

Lawrence Livermore National Laboratory

SUNDIALS: Suite of Nonlinear and



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

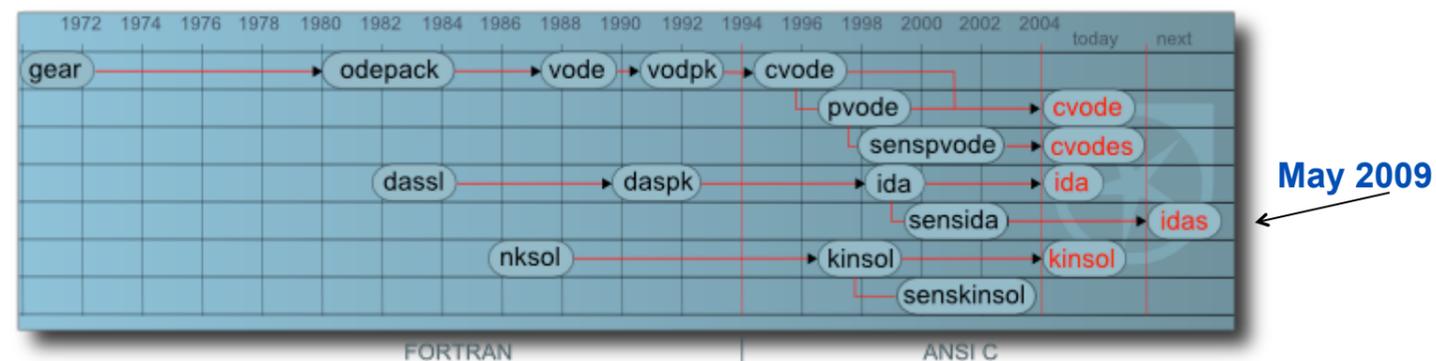
Outline

- 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



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 parallel solution of large-scale problems and on sensitivity analysis



Push to solve large, parallel systems motivated rewrites in C

- **CVODE**: rewrite of VODE/VODEPK [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 and 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

$$\begin{aligned} F(\dot{\mathbf{x}}, \mathbf{x}) &= \mathbf{0} \\ \mathbf{x}(t_0) &= \mathbf{x}_0 \end{aligned}$$

- If $\partial F / \partial \dot{\mathbf{x}}$ is invertible, we solve for $\dot{\mathbf{x}}$ to obtain an ordinary differential equation (ODE)
- 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: $\dot{\mathbf{y}} = \lambda \mathbf{y}, \quad \mathbf{y}_0 = \mathbf{1}$

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

Absolute stability requirement

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

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

Region of absolute stability: $\mathbf{S} = \{z \in \mathbf{C}; |R(z)| \leq 1\}$

where an integrator can be written as

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



Forward and backward Euler show different stability restrictions

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

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

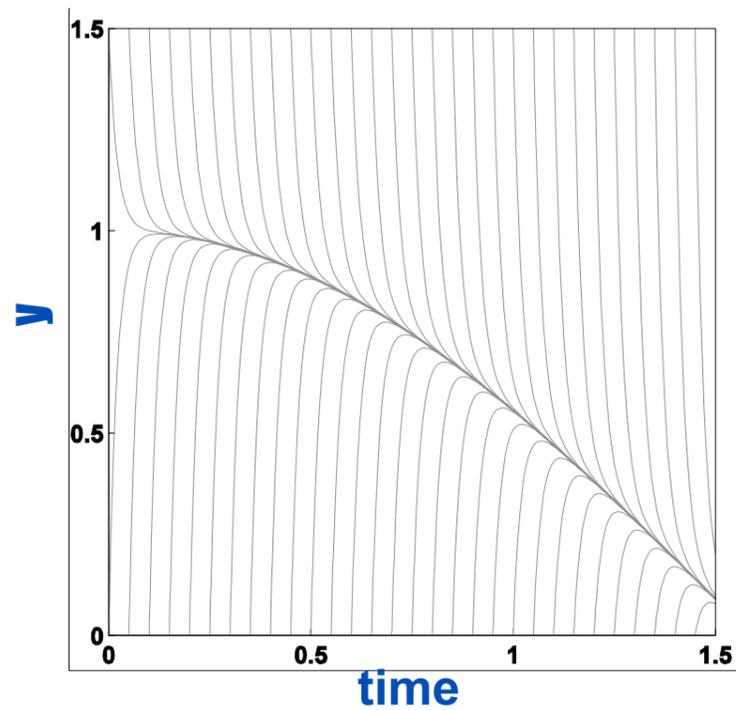
- Backward Euler: $\mathbf{y}_n = \mathbf{y}_{n-1} + h(\lambda \mathbf{y}_n) \Rightarrow R(\mathbf{z}) = \frac{1}{1 - h\lambda}$

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

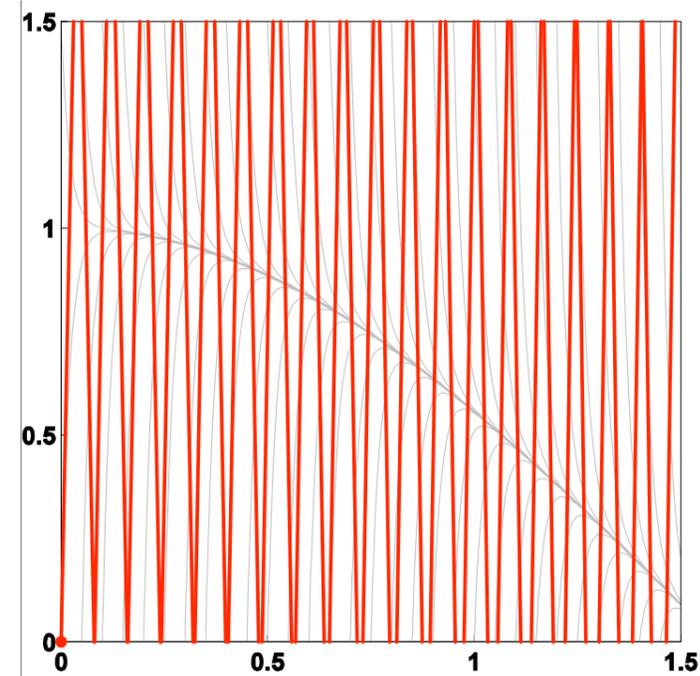


Curtiss and Hirschfelder example

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



Solution curves



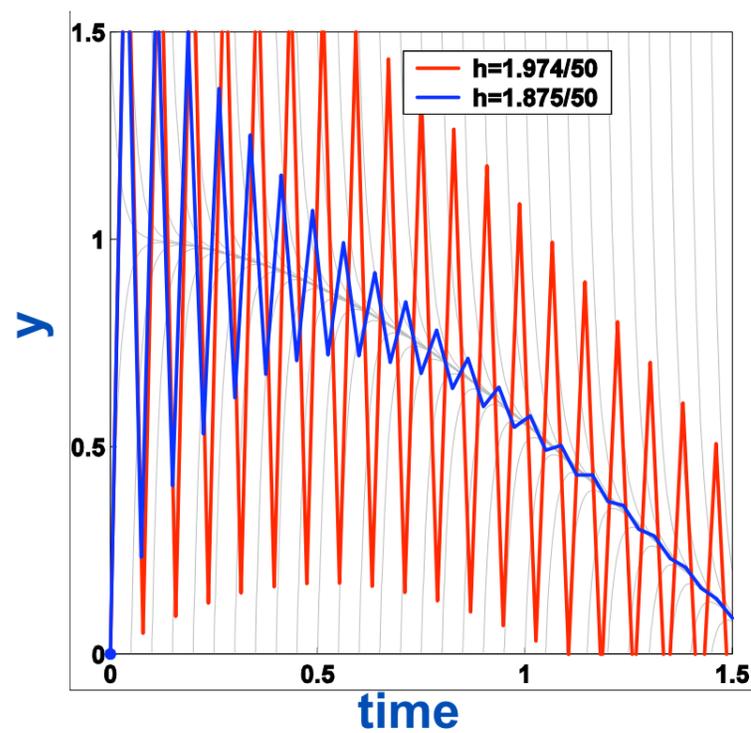
Forward Euler

$h=2.01/50$

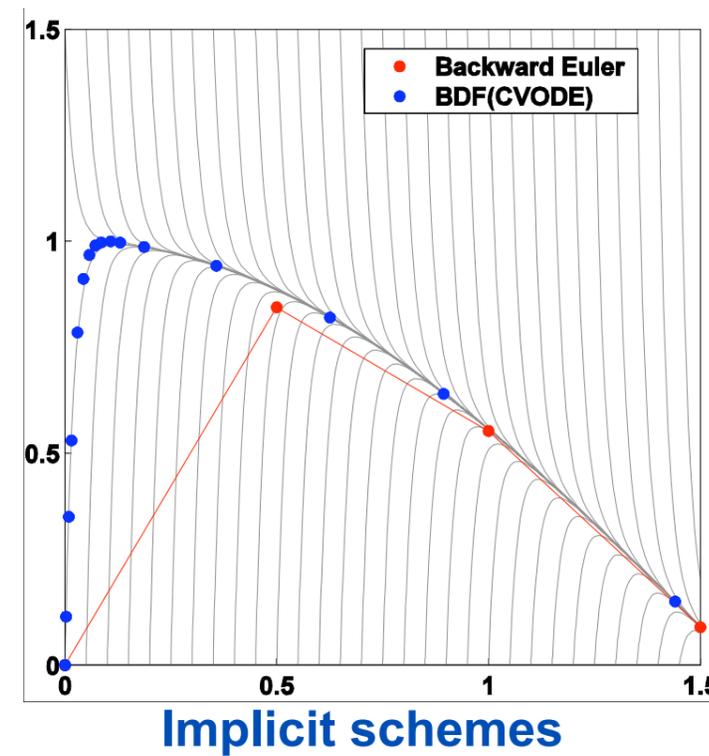


Curtiss and Hirschfelder example

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



Forward Euler



Implicit schemes
 $h=0.5$ for BE



SUNDIALS has implementations of Linear Multistep Methods (LMM)

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

- 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:

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

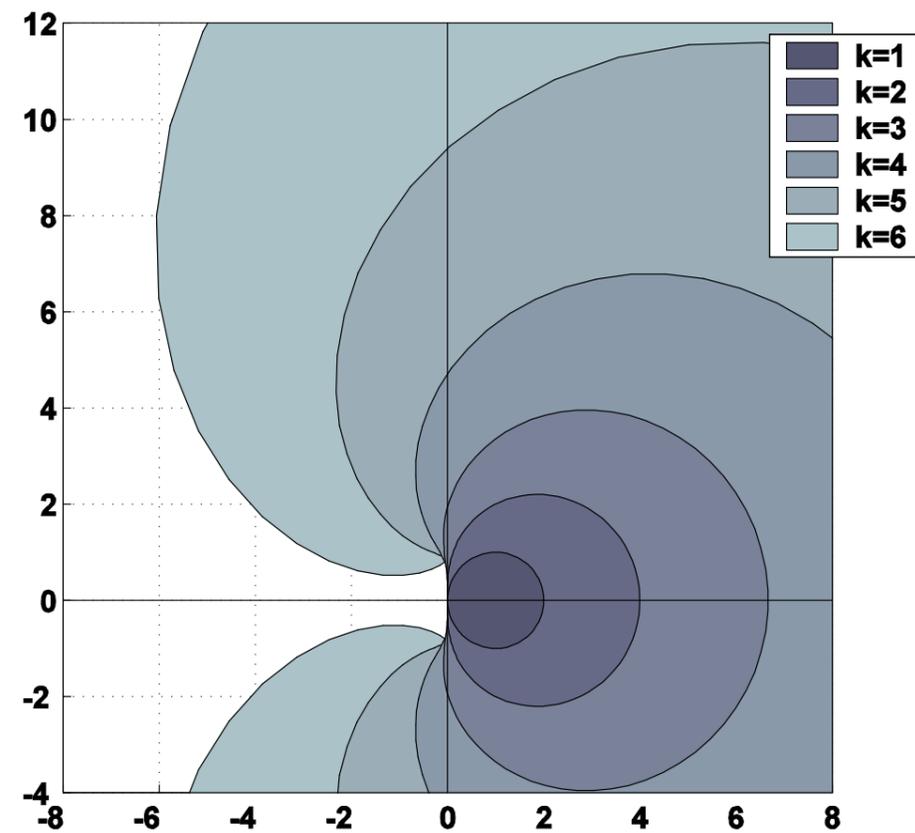
- DAE:

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



Stability is very restricted for higher orders of BDF methods

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



CVODE solves $y'=f(t,y)$

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

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

- 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

$$\mathbf{y}_{n(m+1)} = \beta_0 \mathbf{h}_n \mathbf{f}(\mathbf{y}_{n(m)})$$

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



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 method finds linear system solution in Krylov subspace:
$$K(J,r) = \{r, Jr, J^2r, \dots\}$$
- 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 $F(u) = 0$

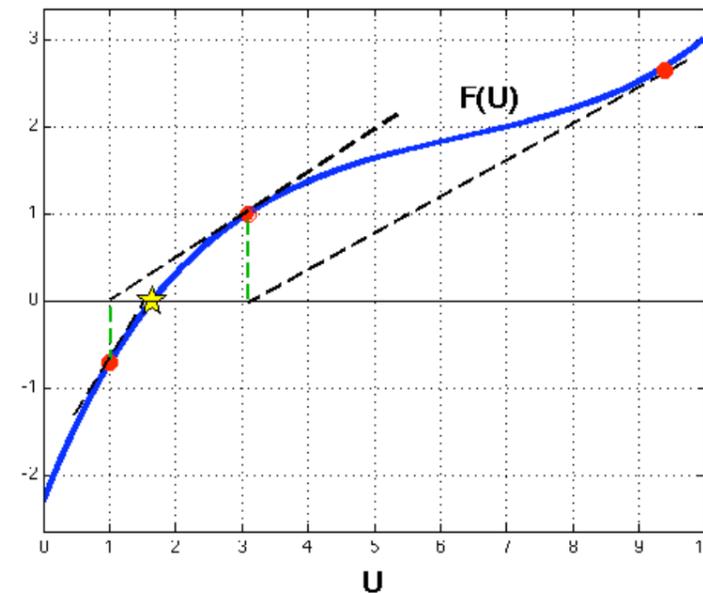
- C rewrite of Fortran NKSOL (Brown and Saad)
- Inexact Newton solver: solves $J \Delta u^n = -F(u^n)$ approximately
- Modified Newton (with direct solves) – this freezes the Newton matrix over a number of iterations
- Krylov solver: scaled preconditioned GMRES, TFQMR, Bi-CGStab
 - Optional restarts 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



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 $\eta^k = 0.9 \left(\frac{\|\mathbf{F}^{(k)}\|}{\|\mathbf{F}^{(k-1)}\|} \right)$
- 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. 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



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

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

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

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

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

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

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

Computational cost:

Computational cost:

$(1+N_p)N_x$ increases with N_p

$(1+N_g)N_x$ 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

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

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

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

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

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

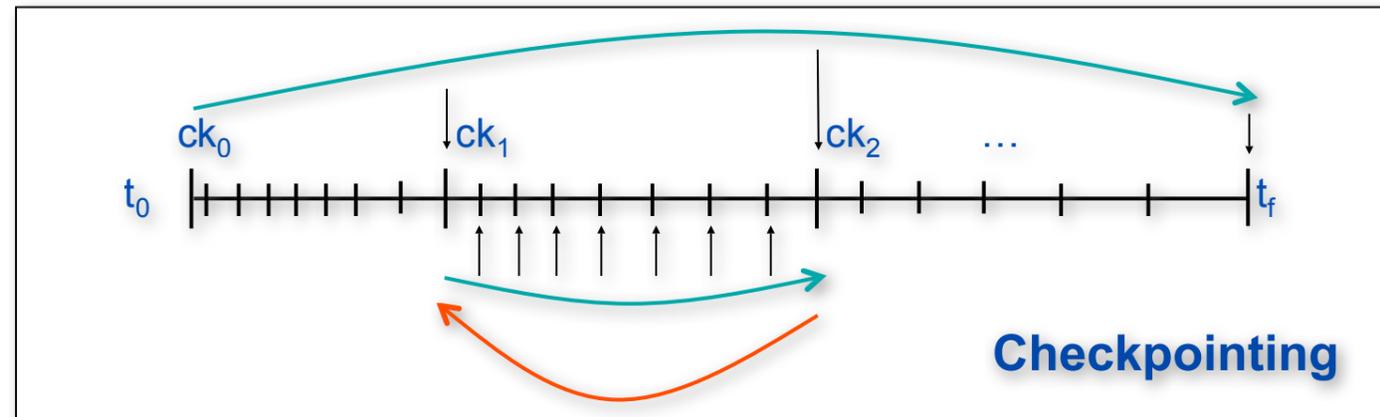
or

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



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 parallelism (if needed) 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

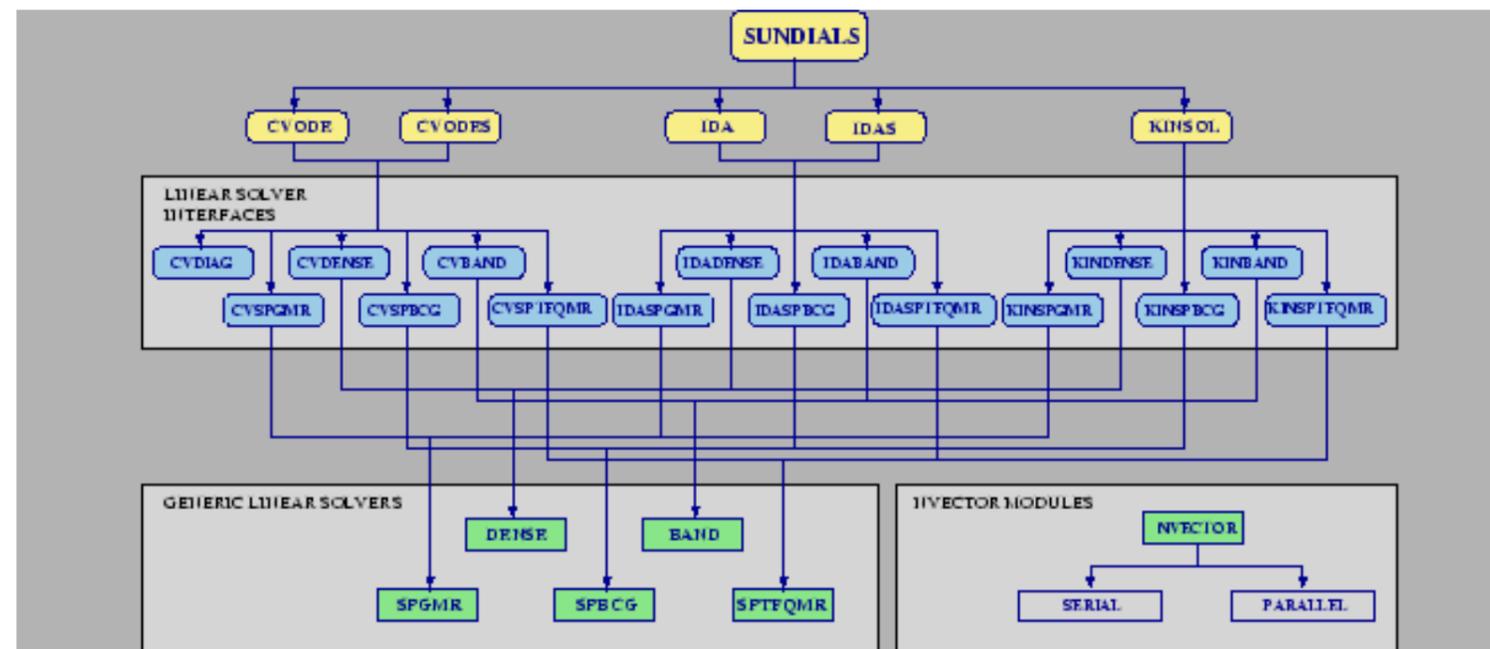


SUNDIALS provides Matlab interfaces

- CVODES, KINSOL, and IDA
- 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 = CVodeMalloc(cvmem,rhs,t0,y,...);
flag = CVSpgmr(cvmem,...);
for(tout = ...) {
    flag = CVode(cvmem, ...,y,...); }

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

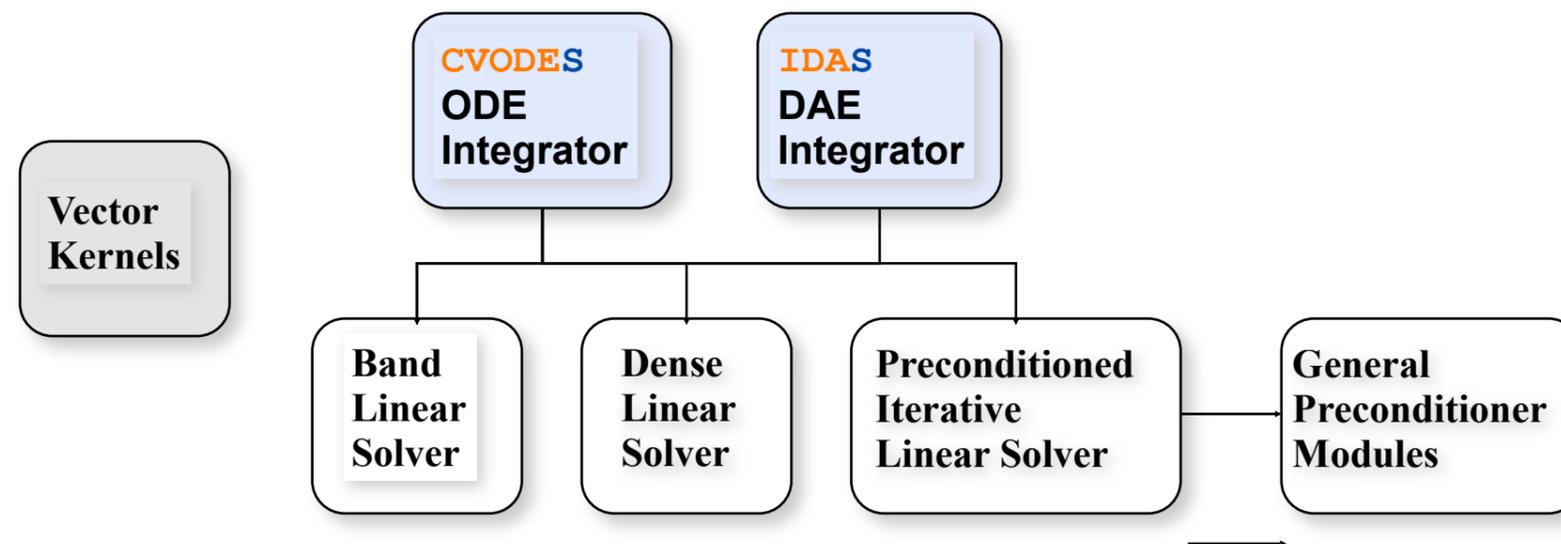


Forward Sensitivity Analysis in SUNDIALS

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 = N_VNew*(n,...);
cvmem = CVodeCreate(CV_BDF,CV_NEWTON);
flag = CVodeSet*(...);
flag = CVodeMalloc(cvmem,rhs,t0,y,...);
flag = CVSpgmr(cvmem,...);
yS = N_VNewVectorArray_*(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);
```

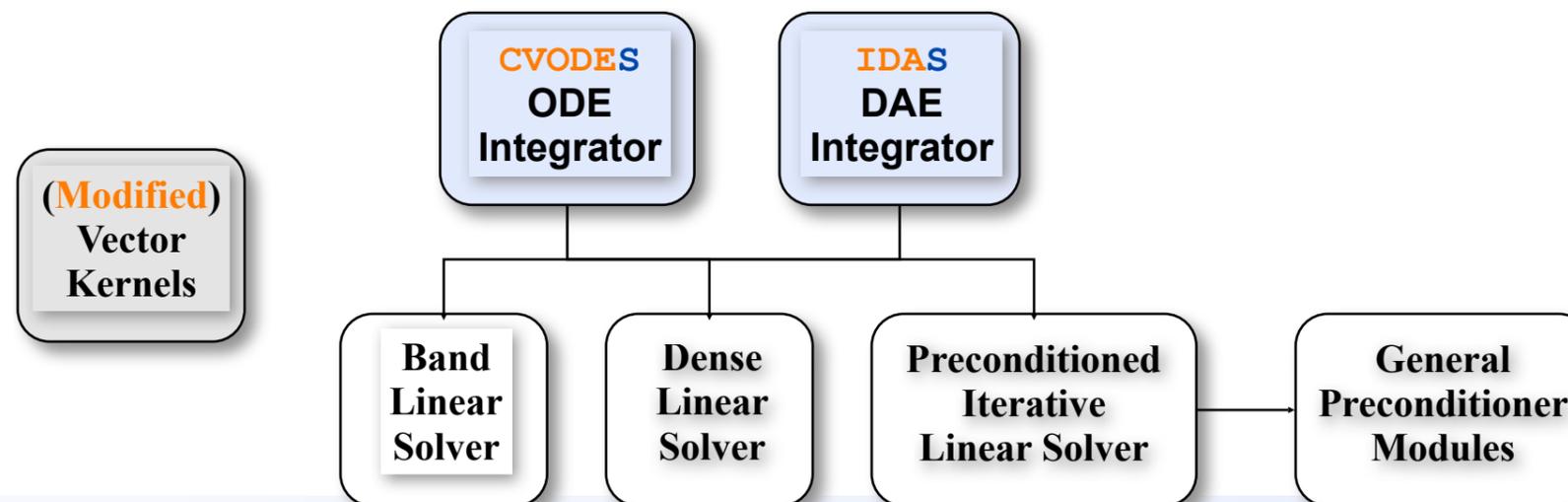


Adjoint Sensitivity Analysis in SUNDIALS

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);
```



Applications of SUNDIALS

- CVODE and KINSOL are being used in parallel fusion simulations at PPPL
- 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 in LLNL's Magnetic Fusion Energy Division.
- KINSOL with a HYPRE multigrid preconditioner is being applied within CASC 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

- IDA and SensIDA used in a cloud and aerosol microphysics model at LLNL to study cloud formation processes.
- 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).
- ...



Influence of Opacity Parameters in Radiation-Diffusion Models

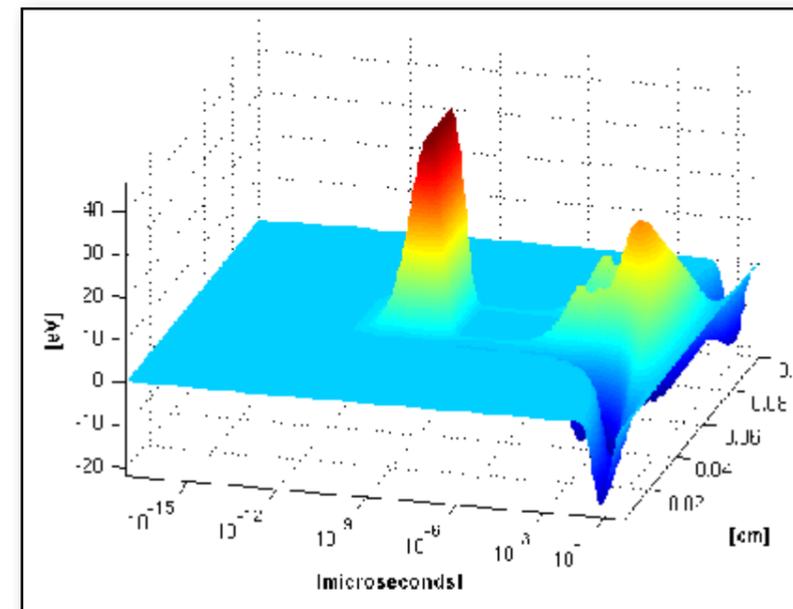
$$\frac{\partial E_R}{\partial t} = \nabla \cdot \left(\frac{c}{3\kappa_R(\rho, T_R) + \|\nabla E_R / E_R\|} \nabla E_R \right) + c\kappa_P(\rho, T_M)(aT_M^4 - E_R) + \chi(x)caT_{source}^4$$

$$\frac{\partial E_M}{\partial t} = -c\kappa_P(\rho, T_M)(aT_M^4 - E_R)$$

- Opacities and EOS are often given through look-up tables
Consider exponential opacities of the form

$$\kappa(\rho, T) = \omega \rho^\alpha T^\beta$$

- Problem dimension: $N_x = 100$, $N_p = 1$
- Find sensitivities of temperatures w.r.t. opacity parameters (**SensPVODE**)



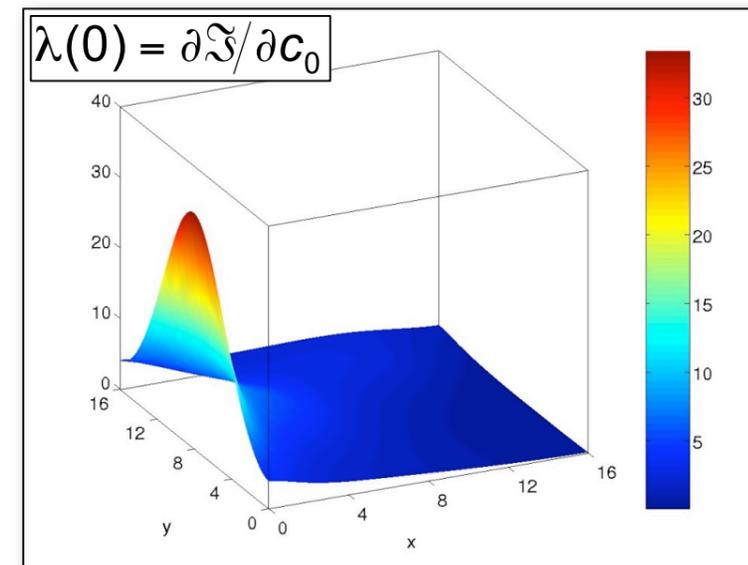
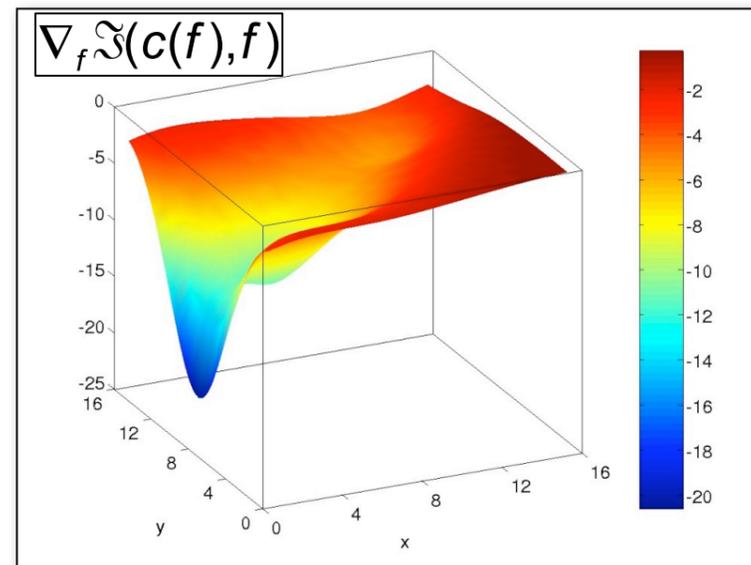
Scaled sensitivity of T_R w.r.t. β .
Early time effect of Plank opacity
Later effects of Rosseland opacity

Atmospheric Event Reconstruction

$$\min_{f,c} \mathfrak{S}(c,f) := \frac{1}{2} \sum_{j=1}^{N_r} \iint_{\Omega} (c - c^*)^2 \delta(x - x_j) d\Omega dt + \frac{1}{2} \beta R(f)$$

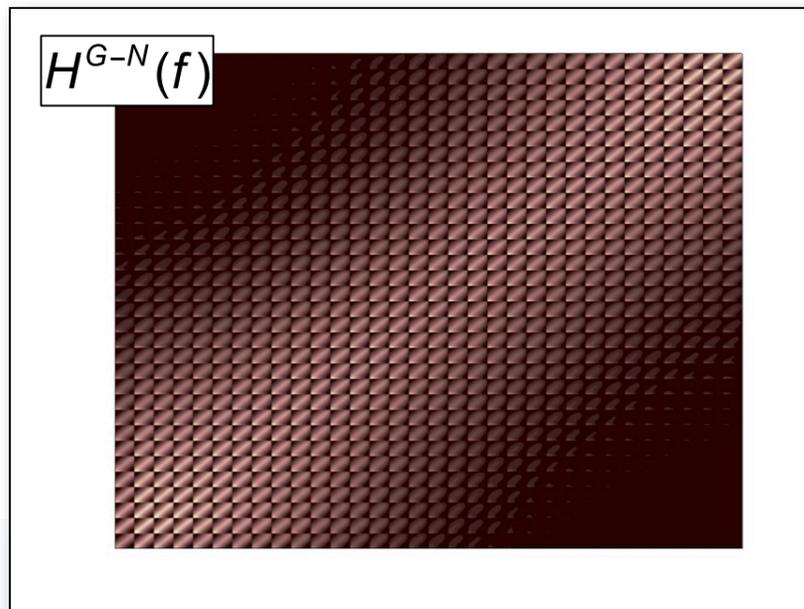
$$\begin{aligned} c_t - k\Delta c + \nabla c \cdot \mathbf{v} + f &= 0, \text{ in } \Omega \times (0, T) \\ \nabla c \cdot \mathbf{n} &= 0, \text{ on } \partial\Omega \times (0, T) \\ c &= c_0(x), \text{ in } \Omega \text{ at } t = 0 \end{aligned}$$

$$\begin{aligned} -\lambda_t - k\Delta\lambda - \nabla\lambda \cdot \mathbf{v} &= g(c, c^*), \text{ in } \Omega \times (0, T) \\ (k\nabla\lambda + \lambda\mathbf{v}) \cdot \mathbf{n} &= 0, \text{ on } \partial\Omega \times (0, T) \\ \lambda(x) &= 0, \text{ in } \Omega \text{ at } t = T \end{aligned}$$

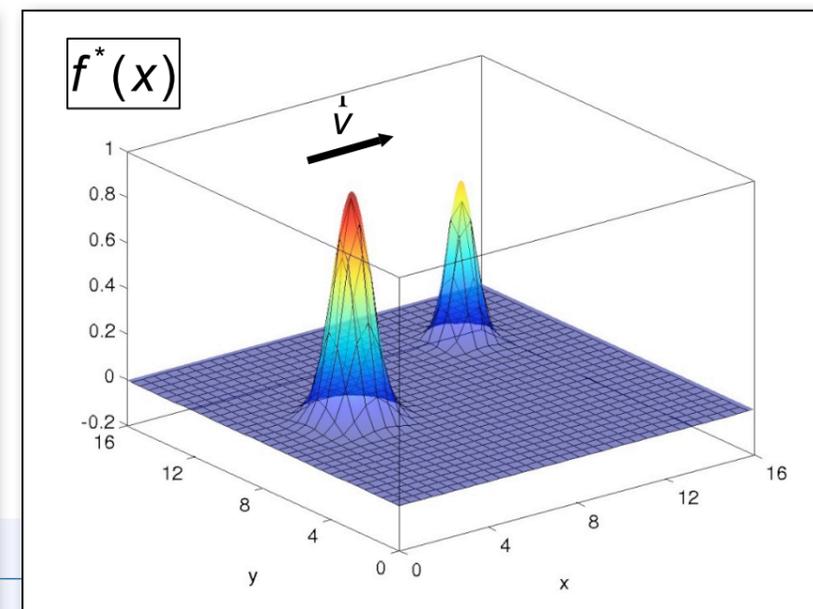


Atmospheric Event Reconstruction

- **CVODES** – for gradient and Hessian-vector products
- **KINSOL** – for NLP solution
- Problem dimensions: $N_{\text{ODE}}=4096$, $N_{\text{NLP}}=1024$



Option:UCRL#



Option:Additional Information

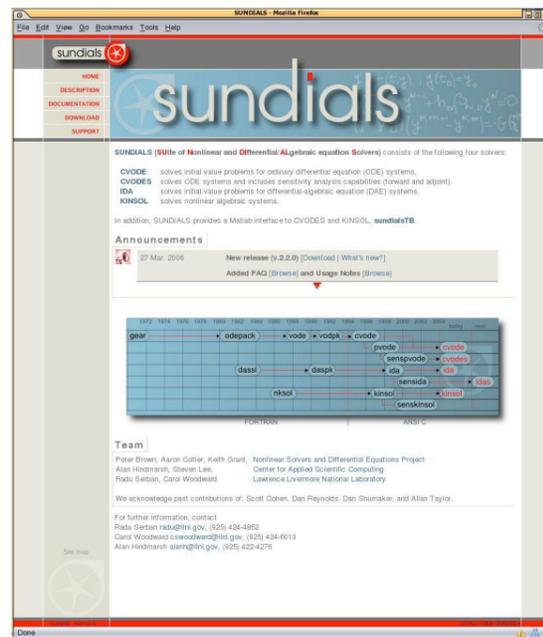
Availability

Open source BSD license

www.llnl.gov/CASC/sundials

Publications

www.llnl.gov/CASC/nsde



The SUNDIALS Team

Peter Brown

Aaron Collier

Keith Grant

Alan Hindmarsh

Steven Lee

Radu Serban

Carol Woodward

Web site:

Individual codes download

SUNDIALS suite download

User manuals

User group email list

