SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers

Carol S. Woodward and Radu Serban

Center for Applied Scientific Computing, LLNL

Work performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48

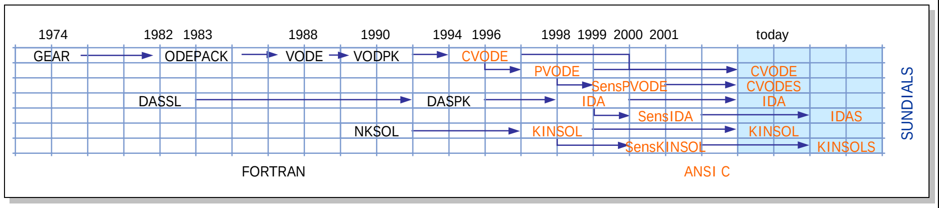
Outline

- **SUNDIALS Overview**
- **SUNDIALS History**
- **The SUNDIALS suite**
  - **CVODE, IDA, KINSOL**
- **Preconditioning**
- **Data Structures**
- **Usage**
- **Fortran interface**
- **Application examples**

- **Sensitivity analysis**
- **Definition and motivation**
- **Approaches**
  - **Forward (FSA), Adjoint (ASA)**
- **FSA in SUNDIALS**
  - **Usage, Methods**
- **ASA in SUNDIALS**
  - **Usage, Implementation**
- **Application examples**
- **Future work**

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

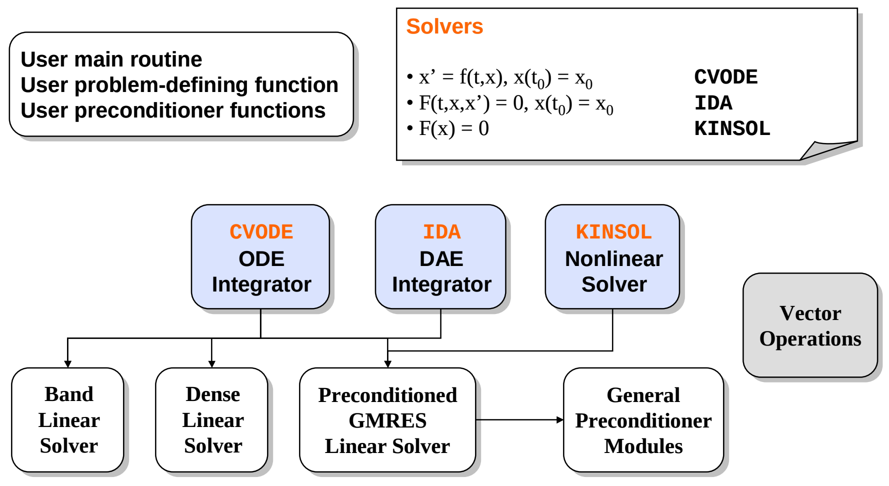
- Fortran solvers written at LLNL:
  - —VODE: stiff/nonstiff ODE systems, with direct linear solvers
  - —VODPK: with Krylov linear solver (GMRES)
  - —NKSOL: Newton-Krylov solver - nonlinear algebraic systems
  - —DASPK: DAE system solver (from DASSL)
- Recent focus has been on parallel solution of 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, Hindmarsh, 98]
- KINSOL: rewrite of NKSOL [Taylor, Hindmarsh, 98]
- IDA: rewrite of DASPK [Hindmarsh, Taylor, 99]
- Sensitivity variants: SensPVODE, SensIDA, SensKINSOL [Brown, Grant, Hindmarsh, Lee, 00-01]
- Organized into a single suite, SUNDIALS, with one ODE solver, CVODE
- New sensitivity capable solvers in SUNDIALS:
  - — CVODES [Hindmarsh, Serban, 02]
  - — IDAS – in development

Structure of SUNDIALS


User main routine
User problem-defining function
User preconditioner functions
Solvers
• $x' = f(t,x)$, $x(t_0) = x_0$ CVODE
• $F(t,x,x') = 0$, $x(t_0) = x_0$ IDA
• $F(x) = 0$ KINSOL
CVODE ODE Integrator
IDA DAE Integrator
KINSOL Nonlinear Solver
Vector Operations
Band Linear Solver
Dense Linear Solver
Preconditioned GMRES Linear Solver
General Preconditioner Modules

The SUNDIALS solvers share common features

- Written in C, Fortran interfaces for CVODE and KINSOL
- Inexact Newton for nonlinear systems
- GMRES for linear solves (dense option for CVODE, IDA, CVODES, and IDAS)
- User supplies system-defining function
- Written in a data structure neutral manner
  - —Do not assume any specific information about data
  - —Vector operations can be supplied
- User supplies preconditioner setup and solve routines
  - —Default band preconditioner available
  - —Can use external preconditioning packages
- Philosophy: Keep codes simple to use

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

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

- Bound time integration error with:

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

The 1/6 factor tries to account for estimation errors

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

- Krylov iterative method finds linear system solution in Krylov subspace: $K(J,r) = \{r, \quad Jr, \quad J^2r, \quad \ldots\}$
- 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
- Dense solver option also available

- Precondition I-γH, where γ is related to the time step size and H is an approximation to J, the Jacobian of f

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
- Newton correction uses the Jacobian:

$$J = \frac{\partial F}{\partial y} + \frac{\alpha_0}{\Delta t}\frac{\partial F}{\partial y'}$$

- 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 with a preconditioned Krylov solver
- Krylov solver: scaled preconditioned GMRES
  - Optional restarts
  - Preconditioning on the right: $(J\,P^{-1})(Ps) = -F$
- Krylov iteration requires matrix-vector products; can be supplied by the user or done by differencing
- Optional constraints: $u_i > 0$, $u_i < 0$, $u_i \geq 0$ or $u_i \leq 0$
- Dynamic linear tolerance selection
- Can scale equations and/or unknowns

Inexact Newton's method gives quadratic convergence near the solution

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

- Repeat for each k
  - Solve, $J(x^k)s^{k+1} = -F(x^k)$ so that,

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

  - Update, $x^{k+1} = x^k + s^{k+1}$
- Until, $\left\| F(x^{k+1}) \right\| \le \text{tol}$
- If x^0 is "close enough" to the solution,

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

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

Linear stopping tolerances can be chosen to prevent “oversolves”

- 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 $\eta^k = \left\| \|F^k\| - \|F^{k-1} - J^{k-1}s^{k-1}\| \right\| / \|F^{k-1}\|$
  - Choice 2 $\eta^k = 0.9\left(\|F^{(k)}\| / \|F^{(k-1)}\| \right)^2$
- Constant value
  - Kelley method $\eta^k = 0.1$
  - ODE literature $\eta^k = 0.05$

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 $I - \gamma\tilde{J}, \quad \tilde{J} \approx J$
- 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 two options for preconditioning:
  - Hooks for user-supplied preconditioning
  - BandPre module – Banded preconditioner (serial)
  - BBDPre module – Band-Block-Diagonal (parallel)

The SUNDIALS NVECTOR module is generic

- 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, of course, 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 code usage (new release) 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 “cvspgmr.h”
#include “nvector_parallel.h”

y = N_VNew_Parallel(comm,n,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);
```

CVODE and KINSOL provide Fortran interfaces

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

- 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.

- Plan a move to the Babel language interoperability tool for access to other languages as well

Some Applications

- CVODE is used in a 3D parallel tokamak turbulence model in LLNL's Magnetic Fusion Energy Division. Typical run: 7 unknowns on a 64x64x40 mesh, with 60 processors
- KINSOL with a *hypre* multigrid preconditioner is used in LLNL's Geosciences Division for an unsaturated porous media flow model. Fully scalable performance has been obtained on up to 225 processors on ASCI Blue.
- All solvers are being used to solve 3D neutral particle transport problems in CASC. Scalable performance obtained on up to 5800 processors on ASCI Red.
- Other applications: disease detection, astrophysics, magnetohydrodynamics
- Many more...

Sensitivity analysis in SUNDIALS

- Definition and motivation
- Approaches
  - FSA
  - ASA
- FSA in SUNDIALS
  - Usage
  - Methods
- ASA in SUNDIALS
  - Usage
  - Implementation
- Application examples

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.**
- **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

Forward Sensitivity Analysis

- **For a parameter dependent system**

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

find s_i=dx/dp$_i$ by simultaneously solving the original system with the N$_p$ sensitivity systems obtained by differentiating the original system with respect to each parameter in turn:

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

- **Gradient of a derived function** $g(t,x,p) \Rightarrow dg/dp = g_x s + g_p$
- **Obtain gradients with respect to p for any derived function**
- **Computational cost - (1+N$_p$)N$_x$ - increases with N$_p$**

Adjoint Sensitivity Analysis

- **index-0 and index-1 DAE**

$$\lambda^* F_{\dot{x}}\Big|_{t=T} = 0 \qquad \frac{dG}{dp} = \int_0^T (g_p - \lambda^* F_p)dt + (\lambda^* F_{\dot{x}})\Big|_{t=0} x_{0p}$$

$$\boxed{\begin{array}{l} \begin{cases} (\lambda^* F_{\dot{x}})' - \lambda^* F_x = -g_x \\ \lambda^* F_{\dot{x}} x_p = \ldots \quad \text{at } t=T \end{cases} \\ \\ G(x,p) = \int_0^T g(t,x,p)dt \\ \dfrac{\mathrm{dG}}{\mathrm{dp}} = \int_0^T (g_p - \lambda^* F_p)dt - (\lambda^* F_{\dot{x}} x_p)\Big|_0^T \end{array}}$$

- **Hessenberg index-2 DAE**

$$\begin{cases} \dot{x} = f^d(x,y,p) \\ 0 = f^a(x,p) \end{cases} \rightarrow \begin{cases} \dot{\lambda} + A^*\lambda + C^*\eta = -g_x^* \\ B^*\lambda = -g_y^* \end{cases}$$

$$A = \frac{\partial f^d}{\partial x}, B = \frac{\partial f^d}{\partial y}, C = \frac{\partial f^a}{\partial x}, \exists (CB)^{-1}$$

impose final conditions of the form $\lambda^*(T) = \xi^* C\Big|_{t=T}$

At $t=T$:

$$\lambda^* B = -g_y \Rightarrow \xi^* CB = -g_y \Rightarrow \xi^* = -g_y (CB)^{-1}$$
$$f^a(x,p) = 0 \Rightarrow Cx_p = -f_p^a \Rightarrow \lambda^* x_p = -\xi^* f_p^a$$

$$\lambda^*(T) = -g_y (CB)^{-1} C\Big|_{t=T} \qquad \frac{dG}{dp} = \int_0^T (g_p + \lambda^* f_p^d + \eta^* f_p^a)dt + \lambda^*(0)x_{0p} - g_y (CB)^{-1} f_p^a\Big|_{t=T}$$

Adjoint Sensitivity Analysis - Sensitivity of g(x,T,p)

$$\left.\frac{dg}{dp}\right|_{t=T} = \frac{d}{dT}\frac{dG}{dp} = \left(g_p - \lambda^* F_p\right)\Big|_{t=T} + \int_0^T \mu^* F_p dt - \left(\mu^* F_{\dot{x}} x_p\right)\Big|_{t=0} - \left.\left(\frac{d(\lambda^* F_{\dot{x}} x_p)}{dT}\right)\right|_{t=T}$$

$$\boxed{\begin{cases}(\mu^* F_{\dot{x}})' - \mu^* F_x = 0 \\ \mu^* = \mathrm{K} \quad \text{at} \quad t = T\end{cases}}$$

Implicit ODE

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

$$A = \frac{\partial F}{\partial \dot{x}}, B = \frac{\partial F}{\partial x}, \exists A^{-1}$$

$$\begin{cases}(A^*\mu)' - B^*\mu = 0 \\ A^*\mu = g_x^* \quad at\ t = T\end{cases}$$

Semi-explicit index-1 DAE

$$\begin{cases}\dot{x} = f^d(x,y) \\ 0 = f^a(x,y)\end{cases}$$

$$A = \frac{\partial f^d}{\partial x}, B = \frac{\partial f^d}{\partial y}, C = \frac{\partial f^a}{\partial x}, D = \frac{\partial f^a}{\partial y}, \exists D^{-1}$$

$$\begin{cases}\dot{\mu} = -A^*\mu - C^*\nu \\ 0 = B^*\mu + D^*\nu \\ \mu = g_x^* - C^*(D^*)^{-1} g_y^* \quad at\ t = T\end{cases}$$

Hessenberg index-2 DAE

$$\begin{cases}\dot{x} = f^d(x,y) \\ 0 = f^a(x)\end{cases}$$

$$A = \frac{\partial f^d}{\partial x}, B = \frac{\partial f^d}{\partial y}, C = \frac{\partial f^a}{\partial x}, \exists (CB)^{-1}$$

$$\begin{cases}\dot{\mu} = -A^*\mu - C^*\nu \\ 0 = B^*\mu \\ \mu = P^*\left[g_x^* - A^*C^*(B^*C^*)^{-1} g_y^* - \dot{C}^*(B^*C^*)^{-1} g_y^*\right] \quad at\ t = T\end{cases}$$

$$P = I - B(CB)^{-1}C$$

Stability of the adjoint system

- **Explicit ODE: proof using Green's function;**

$$\dot{x} = Ax \longrightarrow \dot{\mu} = -A^*\mu$$

- **Semi-explicit index-1 and Hessenberg index-2 DAE: the EUODE of the adjoint system is the adjoint of the EUODE of the original system;**
  Example: Semi-explicit index-1 DAE

$$\begin{tikzcd}
\begin{cases}\dot{x}^d = Ax^d + Bx^a \\ 0 = Cx^d + Dx^a\end{cases} \arrow[r] \arrow[d] & \begin{cases}\dot{\mu}^d = -A^*\mu^d - C^*\mu^a \\ 0 = B^*\mu^d + D^*\mu^a\end{cases} \arrow[d] \\
\dot{x}^d = Ax^d - B(D)^{-1}Cx^d \arrow[r] & \dot{\mu}^d = -A^*\mu^d + C^*(D^*)^{-1}B^*\mu^d
\end{tikzcd}$$

Stability of the adjoint system (contd.)

- **Implicit ODE and index-1 DAE: use bounded transformation**
- **<u>Lemma</u> (Campbell, Bichols, Terrel)**

  Given the time dependent linear DAE system

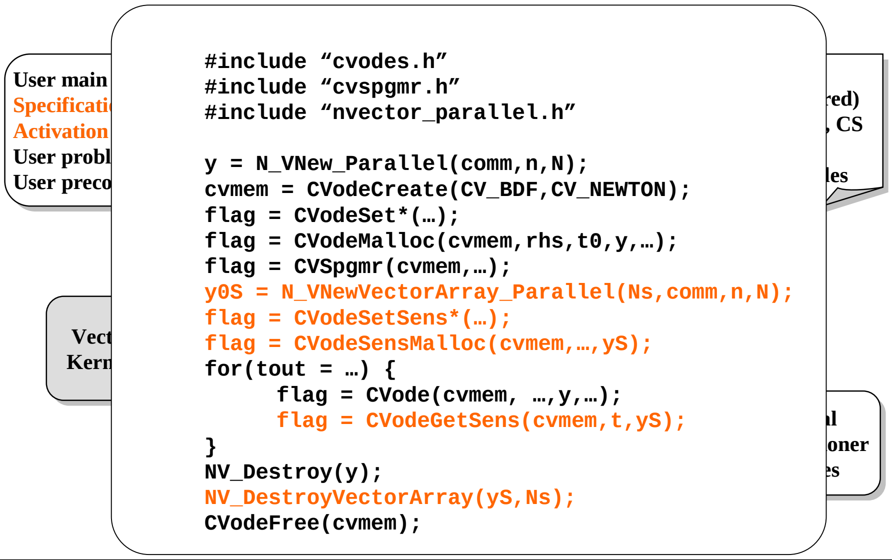
  $$A(t)\dot{x} + B(t)x = f(t)$$

  and nonsingular time dependent differentiable matrices P(t) multiplying the equations of the DAE and Q(t) transforming the variables, the adjoint system of the transformed DAE is the transformed system of the adjoint DAE.
- **<u>Theorem</u>**

  For general index-0 and index-1 DAE systems, if the original DAE system is stable then the augmented DAE system is stable.

$$\boxed{\begin{cases} \dot{\bar\lambda} - F_x^*\lambda = -g_x^* \\ \bar\lambda - F_{\dot x}^*\lambda = 0 \end{cases}}$$

Forward Sensitivity Analysis in SUNDIALS



```
#include “cvodes.h”
#include “cvspgmr.h”
#include “nvector_parallel.h”

y = N_VNew_Parallel(comm,n,N);
cvmem = CVodeCreate(CV_BDF,CV_NEWTON);
flag = CVodeSet*(…);
flag = CVodeMalloc(cvmem,rhs,t0,y,…);
flag = CVSpgmr(cvmem,…);
y0S = N_VNewVectorArray_Parallel(Ns,comm,n,N);
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);
```

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 SPGMR, sensitivity systems solved (theoretically) in 1 iteration

FSA – Generation of the sensitivity system

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

CVODES case

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

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

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

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

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

or

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

Adjoint Sensitivity Analysis in SUNDIALS

User main
Activation
User probl
User revers
User preco
User revers

ned

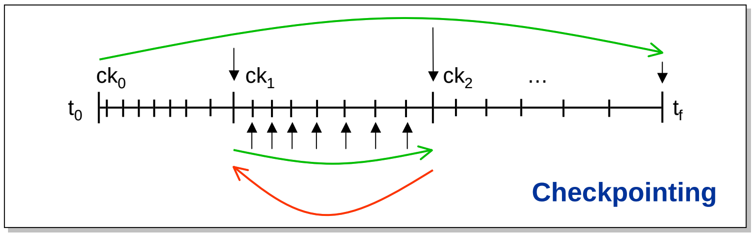
(Modifi
Vecto
Kerne

nl
oner
es

```
#include “cvodes.h”
#include “cvodea.h”
#include “cvspgmr.h”
#include “nvector_parallel.h”

y = N_VNew_Parallel(comm,n,N);
cvmem = CVodeCreate(CV_BDF,CV_NEWTON);
CVodeSet*(…);  CVodeMalloc(…);  CVSpgmr(…);

cvadj = CVadjMalloc(cvmem,STEPS);
flag = CVodeF(cvadj,…,&nchk);
yB = N_VNew_Parallel(commB,nB,NB);
CVodeSet*B(…);  CVodeMallocB(…);  CVSpgmrB(…);
for(tout = …) {
      flag = CVode(cvmem, …,y,…);
      flag = CVodeGetSens(cvmem,t,yS);
}

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

ASA – Implementation

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



  — Cubic Hermite 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
  - PDEs: 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!)

Applications

- SensPVODE, SensKINSOL, SensIDA used to determine solution sensitivities in neutral particle transport applications.
- IDA and SensIDA used in a cloud and aerosol microphysics model at LLNL to study cloud formation processes.
- SensKINSOL used for sensitivity analysis of groundwater simulations.
- 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

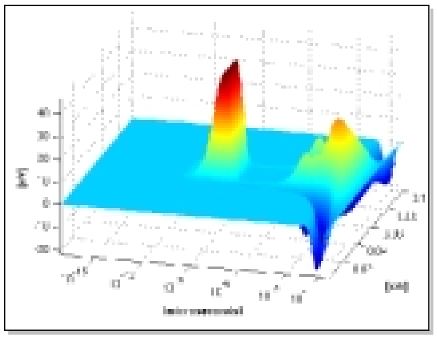
$$\frac{\partial E_R}{\partial t} = \nabla \cdot \left(\frac{c}{3\kappa_R(\rho,T_R) + \|\nabla E_R / E_R\|} \nabla E_R \right) + c\kappa_P(\rho,T_M)\left(aT_M^4 - E_R\right) + \chi(x) c a T_{source}^4$$

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

- 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 beta.
Early time effect of Plank opacity
Later effects of Rosseland opacity

Influence of relative permeability parameters in groundwater simulation

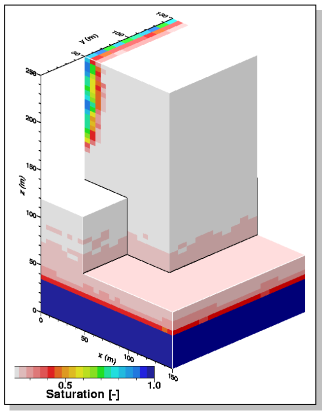
$$\frac{\partial[s(p)\phi\rho]}{\partial t} - \nabla \cdot \left\{ \frac{Kk_r(p)\rho}{\mu}(\nabla p - \rho g \nabla z) \right\} = q$$

$$k_r(p) = \frac{\left\{ 1 - (\alpha|p|)^{n-1}\left[1 + (\alpha|p|)^n\right]^{-m} \right\}^2}{\left[1 + (\alpha|p|)^n\right]^{n/2}}$$

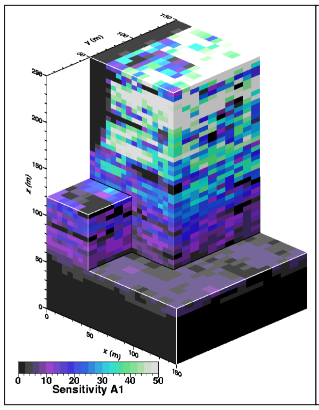
- Sensitivity of water pressure to parameters in the expression for relative permeability:

$$\alpha = a_1 \ln|K| + a_2$$
$$n = b_2$$

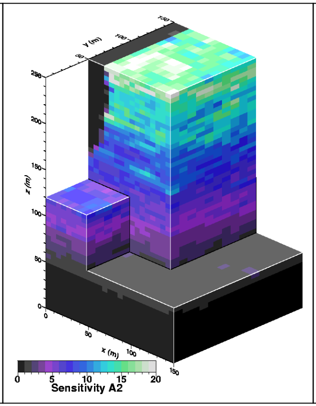
- Problem dimension: $N_x = 18750$, $N_p = 3$
- Software: KINSOL and SensKINSOL



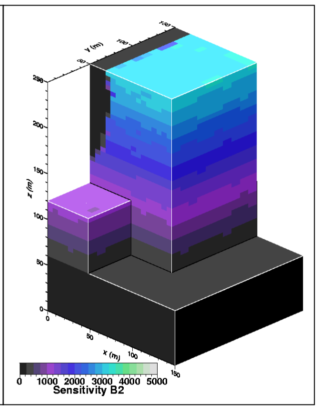
Influence of relative permeability parameters in groundwater simulation - Results



Sensitivity to a_1



Sensitivity to a_2



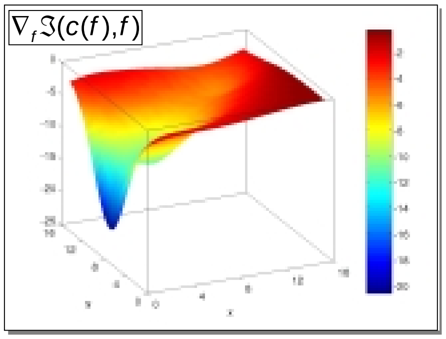
Sensitivity to b_2

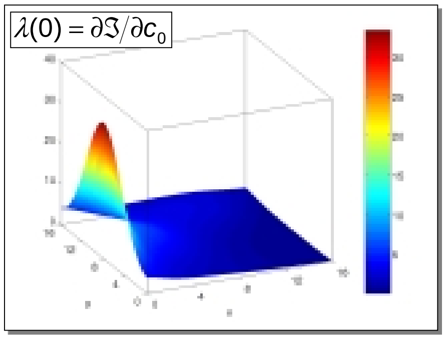
Atmospheric event reconstruction

$$\min_{f,c} \Im(c,f) := \frac{1}{2}\sum_{j=1}^{N_r}\int_t\int_\Omega \left(c - c^*\right)^2 \delta(x - x_j)\, d\Omega dt + \frac{1}{2}\beta R(f)$$

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

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

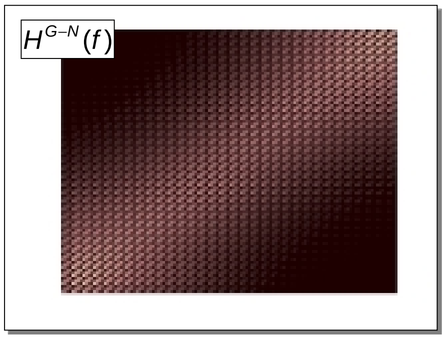


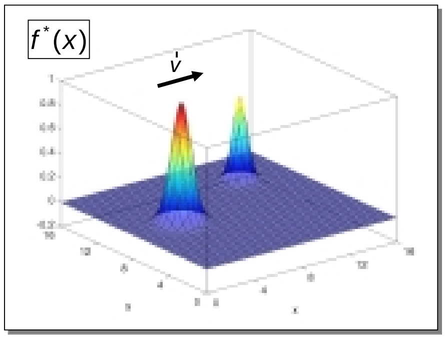


Atmospheric event reconstruction

- **CVODES – for gradient and Hessian-vector products**
- **KINSOL – for NLP solution**

- **Problem dimensions: N_{ODE}=4096, N_{NLP}=1024**





Current and future work

- **More Krylov solvers for the Jacobian systems**
- **IDAS (forward and adjoint sensitivity variant of IDA)**
- **Automatic generation of derivative information**
  - **Complex-step tools for forward sensitivity and/or Jacobian**
  - **Incorporation of AD tools (forward/reverse)**
- **Improved checkpointing / alternatives to checkpointing**
  - **Storage of integrator decision history**
  - **Use of ROM**
- **BABEL / CCA components**
- **New release – expected mid-November**

Availability

Open source BSD license
`www.llnl.gov/CASC/sundials`

Publications
`www.llnl.gov/CASC/nsde`

SUNDIALS
SUite of Nonlinear and DIfferential/ALgebraic equation Solvers

<u>The SUNDIALS Team</u>

- Peter Brown
- Aaron Collier
- Keith Grant
- Alan Hindmarsh
- Steven Lee
- Dan Reynolds
- Radu Serban
- Dan Shumaker
- Carol Woodward

<u>Past contributors</u>

- Scott Cohen
- Allan Taylor