Time Integration

Presented to
ATPESC 2017 Participants

Carol S. Woodward
SUNDIALS Project Lead
Lawrence Livermore National Laboratory

Q Center, St. Charles, IL (USA)
Date 08/07/2017

U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

EXASCALE COMPUTING PROJECT

Outline

- Software
- Definition of ODEs and DAEs
- Stability and stability restrictions
- Implicit vs. explicit methods
- Stiffness
- Linear multistep methods
- Multistage methods and additive multistage methods
- Need for solvers
- Adaptive methods
- Rootfinding
- Data use
- SUNDIALS
- Summary

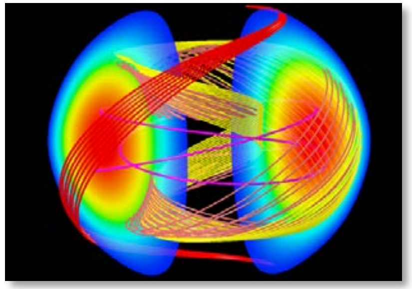
High performance time integration software is available in the DOE in different forms that meet different needs

- ***PETSc:*** TS package, includes DAE and ODE integrators based on variable step multistage methods and additive multistage methods, C
- ***Trilinos:*** Rythmos and Chronos, include ODE and DAE integrators, C++
- ***SUNDIALS:*** Variable step and variable order linear multistep methods, variable step multistage and additive multistage methods, C

While there are numerous integration packages, this talk will emphasize the way SUNDIALS handles each of presented topics

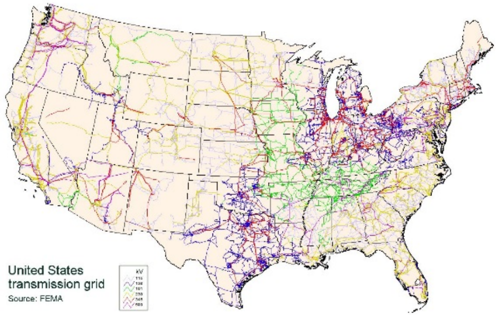
ODEs and DAEs arise in numerous application areas

- Ordinary Differential Equations (ODEs) $\dot{y} = f(t, y)$
  - Method of lines discretization of PDEs: *f* embeds all of the discrete spatial operations
  - Chemical reactions: *f* includes terms for each reaction



Magnetic reconnection

- Differential Algebraic Equations (DAEs) $F(t, y, \dot{y}) = 0, \quad y(0) = y_0$
  - Method of lines discretization of PDEs with algebraic constraints
  - Transmission power system models: *F* includes differential equations for power generators and a large network-based algebraic system constraining power flow
  - Circuit models
  - If $\partial F / \partial \dot{y}$ is invertible, we solve for $\dot{y}$ to obtain an ordinary differential equation (ODE), but this is not always the best approach
  - Else, the system is a differential algebraic equation (DAE)



US Transmission grid
(Wikimedia Commons)

EXASCALE COMPUTING PROJECT

Stability is a key concept when discussing time integration

Dalquist test equation: $\dot{y} = \lambda y, \quad y_0 = 1$

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

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

Absolute stability requirement

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

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

where an integrator can be written as $y_n = R(z)y_{n-1}$, with time advance $z = h\lambda$

Forward and backward Euler show different stability restrictions

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

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

Forward Euler is an explicit method

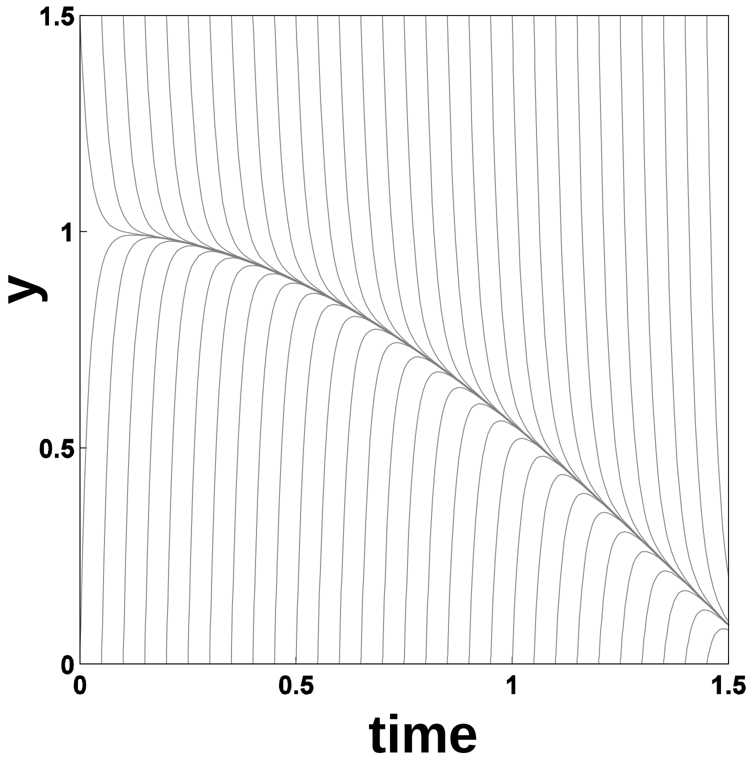
- Backward Euler: $y_n = y_{n-1} + h(\lambda y_n) \Rightarrow R(z) = \dfrac{1}{1 - h\lambda}$

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

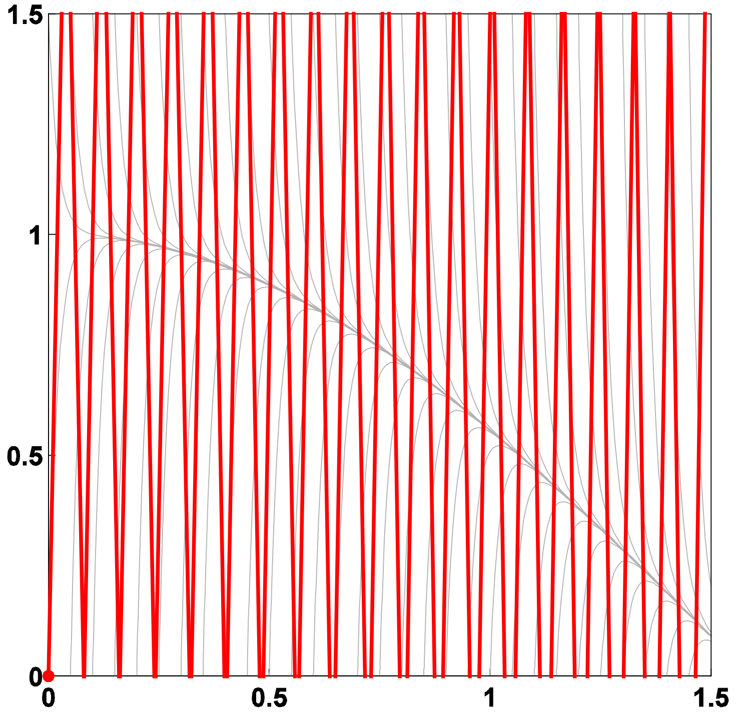
Backward Euler is an implicit method

Curtiss and Hirchfelder example demonstrates what can happen with failure to meet the stability step restriction

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



Solution curves

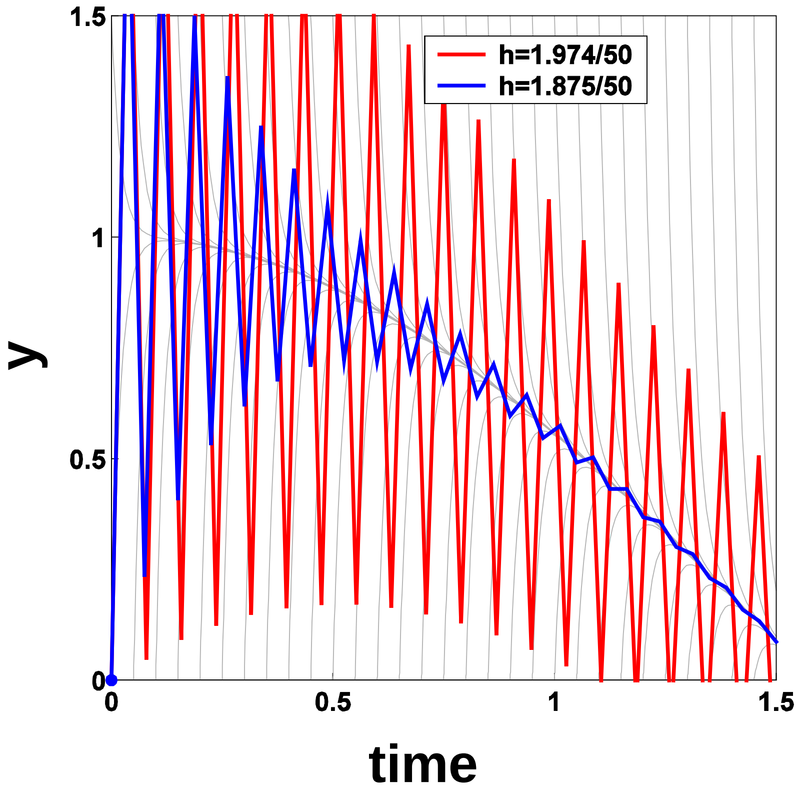


Forward Euler
h=2.01/50

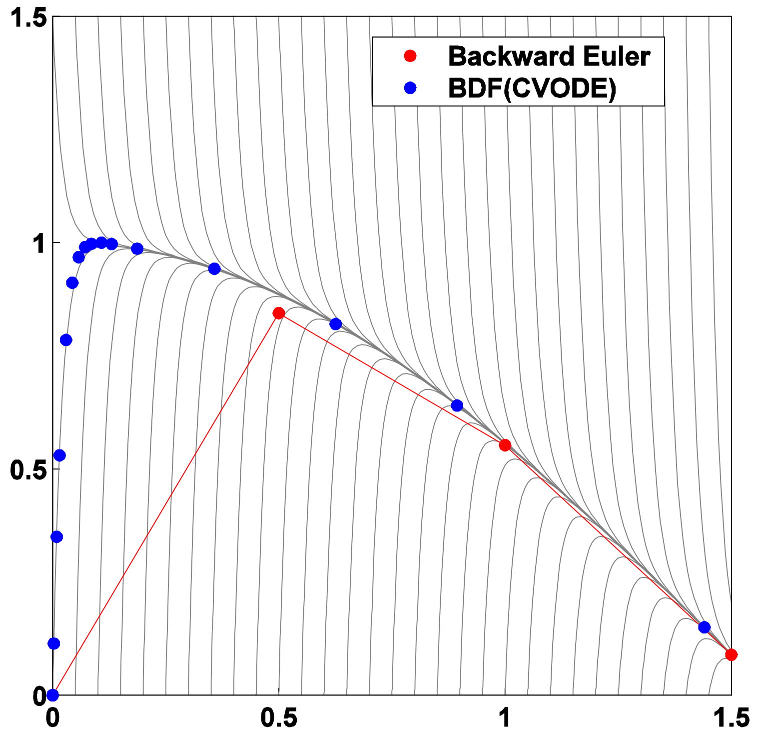
EXASCALE COMPUTING PROJECT

Meeting the restriction with an explicit method or using an implicit method makes a difference!

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



Forward Euler



Implicit schemes
h=0.5 for BE

Explicit and implicit approaches should be selected based on needs of the problem

Explicit Methods

✓Easy to conceptualize

✓Easy to code

✓Do not require solvers

X Stability limits on step sizes

X Tracks fastest dynamics

Implicit Methods

✓Less or nonexistent stability limits

✓Steps over fastest dynamics

X Require linear and/or nonlinear solvers

X Solvers generally require coupling over all unknowns

X Code complexity higher

Implicit methods are most useful when fast dynamics of little interest are present, and accuracy requirements would dictate a much larger time step for resolution

For any time-dependent system, need to know if it is stiff before choosing a numerical solution approach

- (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 $(t_1 - t_0) \min_j \Re(\lambda_j) << -1$
- Due to stability requirements, stiff problems generally require implicit approaches

Implicit approaches for stiff problems often require a very robust nonlinear solver for each time step solution

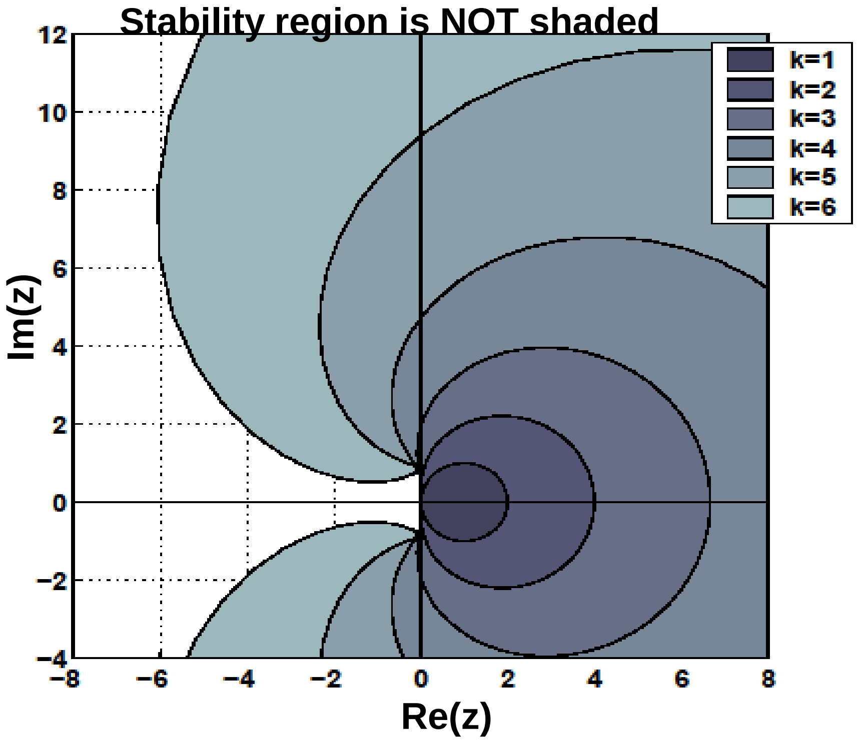
EXASCALE COMPUTING PROJECT

Linear multistep methods construct approximations based on prior states

- Linear Multistep Methods

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

- Nonstiff Implicit: Adams-Moulton
  - $K_1 = 1$, $K_2 = k$, $k = 1,\ldots,12$
- Stiff: Backward Differentiation Formulas [BDF]
  - $K_1 = k$, $K_2 = 0$, $k = 1,\ldots,5$
- Stiff integrators often use a predictor-corrector scheme:



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

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

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

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

EXASCALE COMPUTING PROJECT

Multistage methods construct approximations based on estimates of derivatives at multiple points in a time step

- Multistage methods employ multiple stage solutions

$$\begin{aligned} \mathbf{z_i} &= \mathbf{y_{n-1}} + \mathbf{h_n} \sum_{\mathbf{j=1}}^{\mathbf{s}} \mathbf{a_{i,j} f(t_{n,j}, z_j)}, \quad \mathbf{i = 1, \ldots, s} \\ \mathbf{y_n} &= \mathbf{y_{n-1}} + \mathbf{h_n} \sum_{\mathbf{i=1}}^{\mathbf{s}} \mathbf{b_i f(t_{n,i}, z_i)} \end{aligned}$$

Butcher Tableau

$$\begin{array}{c|cccc} c_1 & a_{1,1} & a_{1,2} & \cdots & a_{1,s} \\ c_2 & a_{2,1} & a_{2,2} & \cdots & a_{2,s} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_s & a_{s,1} & a_{s,2} & \cdots & a_{s,s} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}$$

- The a’s, b’s, c’s, and s define the method, its order of accuracy, and its stability
- Codes with adaptivity in spatial systems or models cannot easily use multi-step methods due to need to interpolate prior step information
- Runge-Kutta (RK) methods are multistage so do not require prior states
- RK methods require multiple nonlinear solves per time step
- Additive RK methods can apply explicit and implicit methods to a split system allowing consistent approximations while using different methods on each

Additive methods address systems with both stiff and nonstiff components

- Split system into stiff, f_I, and nonstiff, f_E, components $M\dot{y} = f_E(t,y) + f_I(t,y)$
- M may be the identity or any nonsingular mass matrix (e.g. FEM)
- Variable step size additive Runge-Kutta Methods combine explicit (ERK) and diagonally implicit (DIRK) RK methods to enable an ImEx integrator
- Let $t_{n,j} = t_{n-1} + c_j \Delta t_n$:

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

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

- Solve for stage solutions, z_i, $i = 1, \ldots, s$, sequentially

Implicit solutions result in nonlinear systems at each time step or stage

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

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

- Stiff systems: Newton iteration

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

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

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

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

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

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

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

Adaptive methods choose time steps to minimize local truncation error and maximize efficiency

- User-defined tolerances:
  - Absolute tolerance on each solution component, $ATOL^i$
  - Relative tolerance for all solution components, RTOL
- Norm calculations are weighted by: $ewt^i = \frac{1}{RTOL|y^i| + ATOL^i}$
- Errors are measured with a weighted root-mean-square norm:

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

- Choose time steps to bound an estimate of the local truncation error; can do with both multistep and multistage methods

The choice of tolerances is critical to accuracy and efficiency for these adaptive methods

- The relative tolerance controls error relative to the size of the solution
  - RTOL = 10^{-4} means that errors are controlled to 0.01%
  - We do not recommend an RTOL above 10^{-3} nor one close to unit roundoff, 10^{-15}
- The absolute tolerances control the absolute size of error when any solution component may be so small that pure relative error control is meaningless
  - Ex: solution starting at a nonzero value but decaying to a noise level, ATOL should be set to the noise level
  - If different components have different noise levels then want ATOL to be a vector
- In general, want to be a bit conservative with these tolerances
  - Rule of thumb: use tolerances 0.01 below desired limits to ensure global errors are below limit
- But not too conservative as the integrator will work harder to meet tight tolerances

To use these adaptive methods effectively, choose tolerances carefully!

EXASCALE COMPUTING PROJECT

Rootfinding capabilities are critical in some applications

- Finds roots of solution-dependent user-defined functions, $g_i(t, y) = 0$ or $g_i(t, y, \dot{y}) = 0$
- Important in applications where problem definition may change based on a function of the solution
- Rootfinding is a critical feature for applications like power grid where solution-dependent system adaptations are common, e.g. voltage limit on a generator
- 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 with a tight tolerance to identify root

EXASCALE COMPUTING PROJECT

Time integrator algorithms do not need to rely on specific data layouts

- All operations within the integrator can be conducted on vectors
- Packages define a vector API; users can use their structures coded to this API
- Within SUNDIALS, each vector implementation defines a content structure and all implemented vector operations, along with routines to clone vectors
- For an implicit method, data layouts are used in
  - Specific vector implementations (streaming and reduction)
  - Solvers (linear and/or nonlinear)
  - Problem-defining function evaluations, f and F, and Jacobian evaluations

Using a time integrator on a given machine requires an efficient implementation of the problem-defining functions, as these typically are the dominant cost

EXASCALE COMPUTING PROJECT

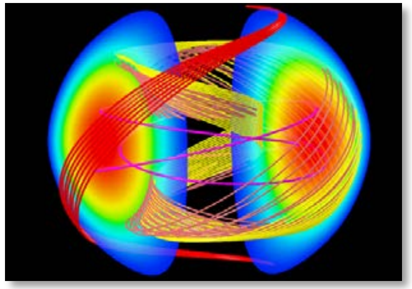
SUNDIALS: SUite of Nonlinear and DIfferential / ALgebraic equation Solvers

- *ODE integrators:*
  - (CVODE/CVODES) variable order and step stiff BDF and non-stiff Adams
  - (ARKode) variable step implicit, explicit, and additive IMEX Runge-Kutta
- *DAE integrators:* (IDA/IDAS) variable order and step stiff integrators
- CVODES and IDAS are equipped with forward and adjoint sensitivity analysis
- *Nonlinear Solver:* (KINSOL) Newton-Krylov, Picard, and accelerated fixed point
- Serial, MPI, openMP, and pthreads vectors; CUDA will be released this fall
- Written in C with interfaces to Fortran
- Designed to be easily incorporated into existing codes
- Modular design allows users to supply their own data structures
- CMAKE-based portable build system
- Freely available (BSD license); >11,000 downloads/year from around the world
- Active user community supported by *sundials-users* email list

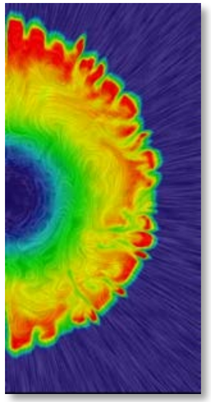
http://www.llnl.gov/CASC/sundials

SUNDIALS has been used worldwide in applications from research and industry

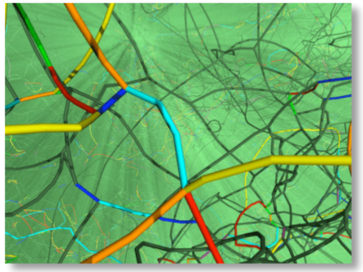
- Power grid modeling (RTE France, LLNL, ISU)
- Simulation of clutches and power train parts (LuK GmbH & Co.)
- Magnetism at the nanoscale (Magpar, Nmag)
- 3D parallel fusion (SMU, U. York, LLNL)
- Spacecraft trajectory simulations (NASA)
- Dislocation dynamics (LLNL)
- Combustion and reacting flows (Cantera)
- Large-scale subsurface flows (Mines, LLNL)
- 3D battery simulation (ORNL - AMPERE)
- Computational modeling of neurons (NEURON)
- Micromagnetic simulations (U. Southampton)
- Released in third party packages:
  - Red Hat Extra Packages for Enterprise Linux (EPEL
  - SciPy – python wrap of SUNDIALS
  - Cray Third Party Software Library (TPSL)



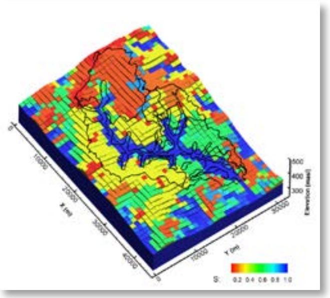
Magnetic reconnection



Core collapse supernova



Dislocation dynamics



Subsurface flow

EXASCALE COMPUTING PROJECT

Summary

- When choosing a time integration method, need to understand
  - What the time scales in the system are
  - Whether the application requires resolving all time scales
  - Whether the system is stiff
  - Whether the system has adaptivity in the spatial or model components
  - What the accuracy requirements are
- The choice of tolerances can impact both accuracy and performance
- Multistep and multistage methods have different characteristics that may make each better suited to an application
- Implicit methods will need algebraic solvers (nonlinear and/or linear)
- Time integrators can be implemented in a data agnostic way allowing for use of application-specific data structures