Modern Evolution Algorithms

EMIL M CONSTANTINESCU
Mathematics and Computer Science
Argonne National Laboratory
USA

INT-24-2A
QCD at the Femtoscale in the Era of Big Data
June 11, 2024

Funding: This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of NP, Scientific Discovery through Advanced Computing (SciDAC) program and U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under contract DE-AC02-06CH11357

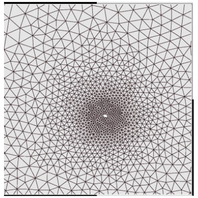
Overview (1)

- Approximation theory:
  - We **approximate** differential equations to compute evolutions
  - We approximate integrals to compute convolutions, …
  - We use approximate models in optimization and nonlinear solvers
  - We approximate distributions with samples or other distributions for inference
  - We approximate approximations to obtain reduced order models, surrogates, emulators
  - ML approximates all sorts of functions used in the above

- Guiding principles: there is no single method that works efficiently for all problems
  - Error estimation: understand the level of errors and help develop better numerical methods
  - Stability: avoid blowups, NANs
  - Invariants' preservation
  - All of the above: **fit-for-purpose**

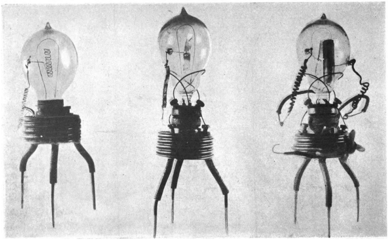
Overview – Problem Formulation (2)

- **Solve evolution equations** $\dot{y} := \frac{\partial y}{\partial t} = f(y)\,,\; y(t_0) = y_0$
- Autonomous only $f(t,y) \Rightarrow f(y)\,, \quad \dot{\widehat{y}} = [f(\widehat{y}), 1]^\top\,,\; \widehat{y} = [y, t]^\top$
- Can solve on manifolds or PDAEs
$$\begin{aligned}\dot{y} &= f(y,z)\,,\; y(t_0) = y_0\,,\; z(t_0) = z_0\,,\; \text{and } f(\partial_x y, \partial_{x,x} y, \cdots) \\ 0 &= g(y,z)\end{aligned}$$
- Or integro-differential eq. (Mori-Zwanzig, QCFs)
$$f(y) = h(y) + \int_{x \text{ or } t} \mathrm{d}\{x \text{ or } t\} r(y)$$
- Discretization on nonuniform grids in *y(x)* and *t*



Overview – What is Modern (3)

- The Runge-Kutta 4 (RK4) method was developed between 1895-1901, a few years before vacuum tubes were invented



- The BDF-2 method was developed in 1952, one year before the first transistor was used in a device

Overview – Modern Numerical Methods (3)

- ‘Integrators’ are classified as explicit or implicit
- Families: multistage (Runge-Kutta), multistep (Adams, BDF), general linear methods, Nystrom, extrapolation, exponential integrators, spectral differed correction, W-methods, …
- Partitioned integrators: semi-explicit, semi-implicit, implicit-explicit, multirate, …
- Adaptive integrators: time-step adaptivity, mesh adaptive (adaptive mesh refinement – AMR); can be static or dynamic AMR
- Invariant preservation: positivity, conservation of total “mass’, symplectic, reversible, monotonic, …
- Can provide error estimates, continuous interpolation/extrapolation

Runge-Kutta 4

- The Runge-Kutta 4 (RK4) method is a remarkable method

$$y_{n+1} = y_n + \triangle t \left(\frac{1}{6} k_1 + \frac{1}{3} k_2 + \frac{1}{3} k_3 + \frac{1}{6} k_4 \right)$$

Stages:

$$k_1 = f(y_n)$$
$$k_2 = f(y_n + \frac{1}{2} \triangle t\, k_1)$$
$$k_3 = f(y_n + \frac{1}{2} \triangle t\, k_2)$$
$$k_4 = f(y_n + \triangle t\, k_3)$$

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Forward Euler

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

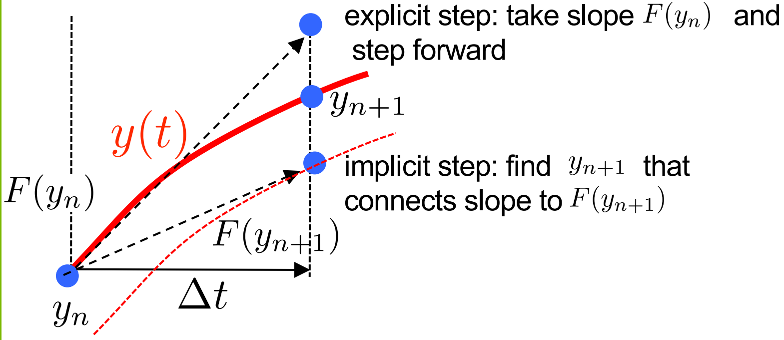
Backward Euler

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

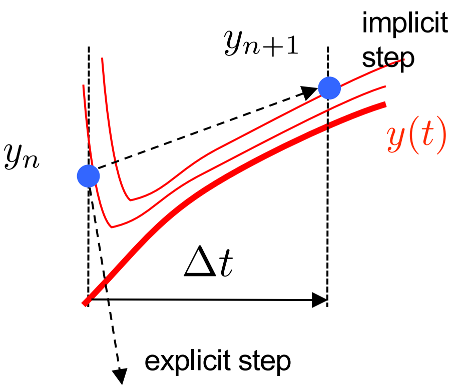
MultistageTime Stepping Basics

Explicit (forward Euler) $y_{n+1} = y_n + \Delta t F(y_n)$

Implicit (backward Euler) $y_{n+1} = y_n + \Delta t F(y_{n+1})$



Stiff system

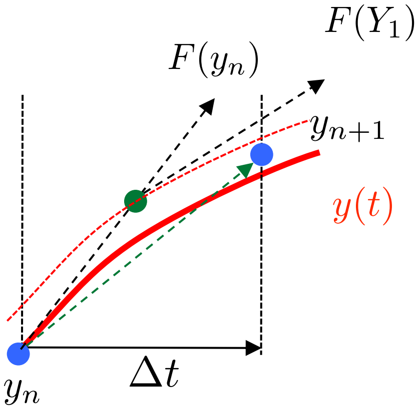


Multistage methods: Runge-Kutta

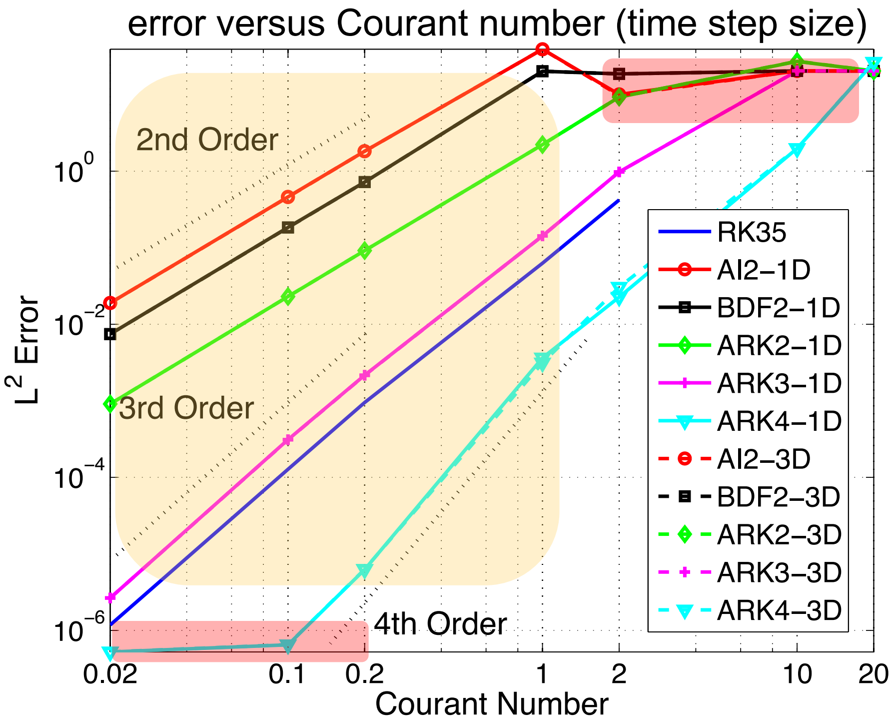
$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

$$y_{n+1} = y_n + \Delta t \sum_{i=1}^{s} b_i F(t_n + c_i \Delta t, Y_i)$$

$$Y_i = y_n + \Delta t \sum_{j=1}^{s} a_{ij} F(t_n + c_j \Delta t, Y_j)$$

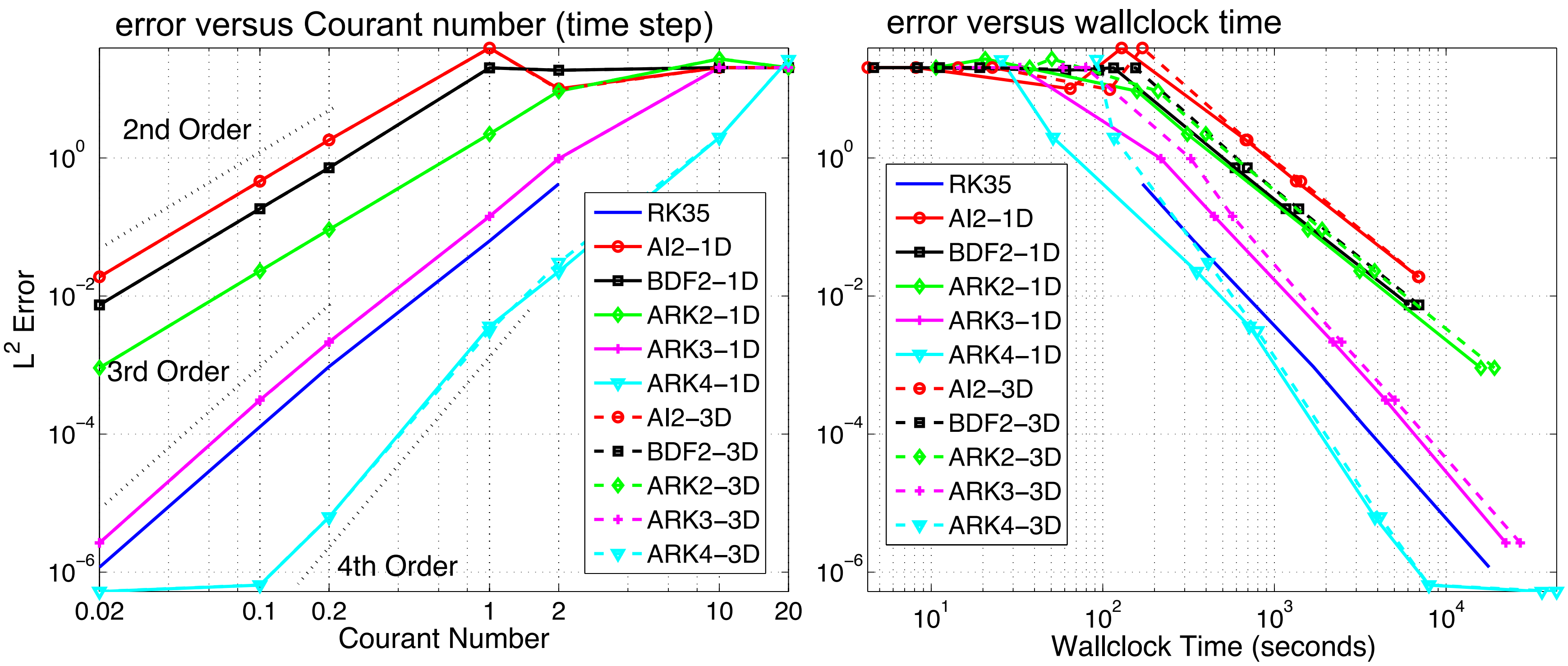


Error and Convergence

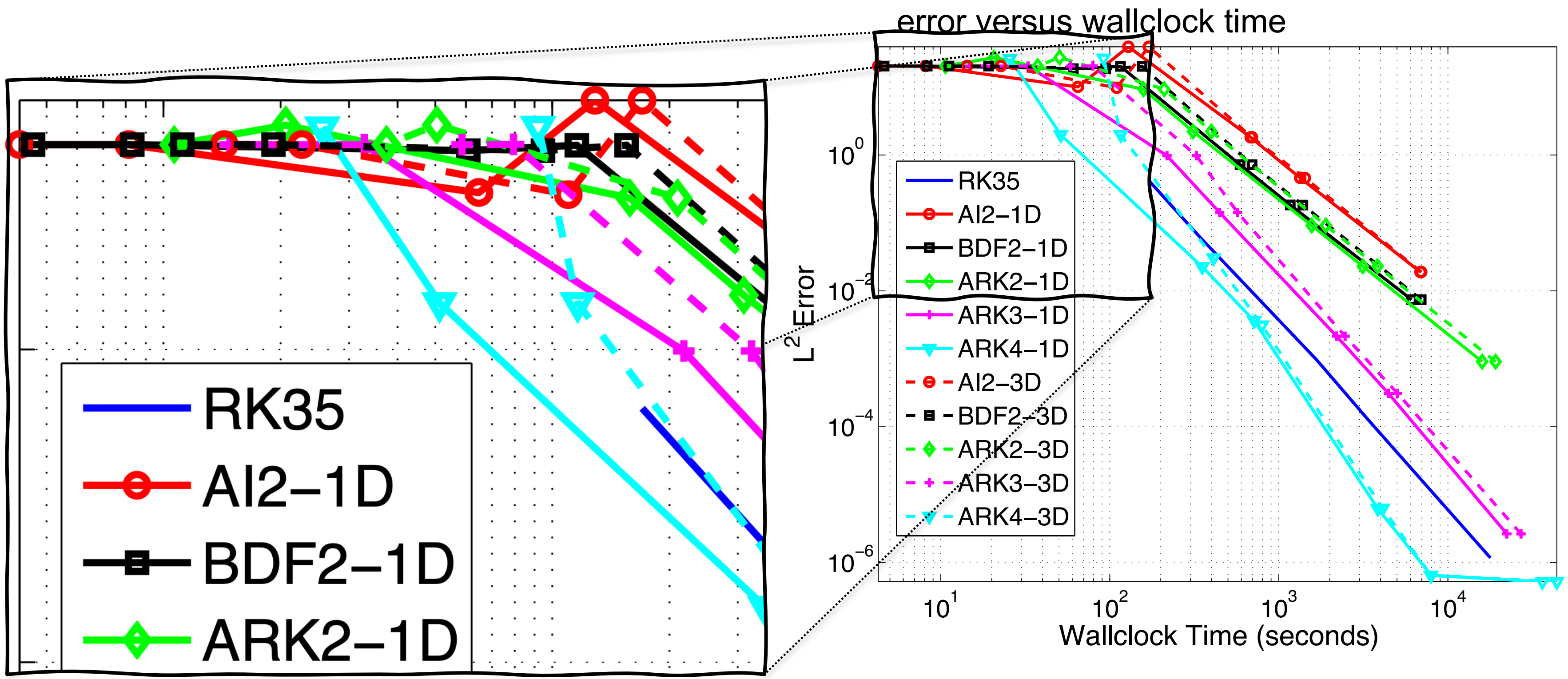


Convergence (error vs time step) for numerical integrators of different orders

Accuracy vs Computational Cost (exclude HPC)



Cost vs Accuracy



Runge-Kutta with Local Error Estimation

- Reuse computed stages to get another solution of a different order
- Runge-Kutta Fehlberg: two methods of orders 5 and 4

c_1	a_{11}	a_{12}	$\dots$	a_{1s}
c_2	a_{21}	a_{22}	$\dots$	a_{2s}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	a_{s1}	a_{s2}	$\dots$	a_{ss}
	b_1	b_2	$\dots$	b_s
	b_1^*	b_2^*	$\dots$	b_s^*

0						
1/4	1/4					
3/8	3/32	9/32				
12/13	1932/2197	–7200/2197	7296/2197			
1	439/216	–8	3680/513	-845/4104		
1/2	–8/27	2	–3544/2565	1859/4104	–11/40	
	16/135	0	6656/12825	28561/56430	–9/50	2/55
	25/216	0	1408/2565	2197/4104	–1/5	0

Integrators with Error Control

- All integrators provide an error control mechanism: MATLAB, Python, Julia, PETSc, Trilinos, Sundials, CVODE, …

Error control procedure:

- A step is taken: Δt_n
- Estimate the error with a different (often lower order) method

$$ErrEst_n = C(t_n)\Delta t_n^{\hat{p}+1} + \mathcal{O}(\Delta t_n^{\hat{p}+2})$$

- Aim to have

$$ErrEst_{new} = Tol$$

$$ErrEst_{new} = C(t_n)\Delta t_{new}^{\hat{p}+1} + \mathcal{O}(\Delta t_{new}^{\hat{p}+2})$$

- If tolerance is satisfied, then take a new step; if not, retake the step with $\Delta t_{new} = r\Delta t_n$

$$r = \left(\frac{Tol}{ErrEst_n}\right)^{\frac{1}{\hat{p}+1}}$$

ode23 [Bogacki–Shampine, 1989;Raltson 1965]

$$Y^{(1)} = y_{[n]}; \quad Y^{(2)} = y_{[n]} + \frac{1}{2}\Delta t f(Y^{(1)})$$

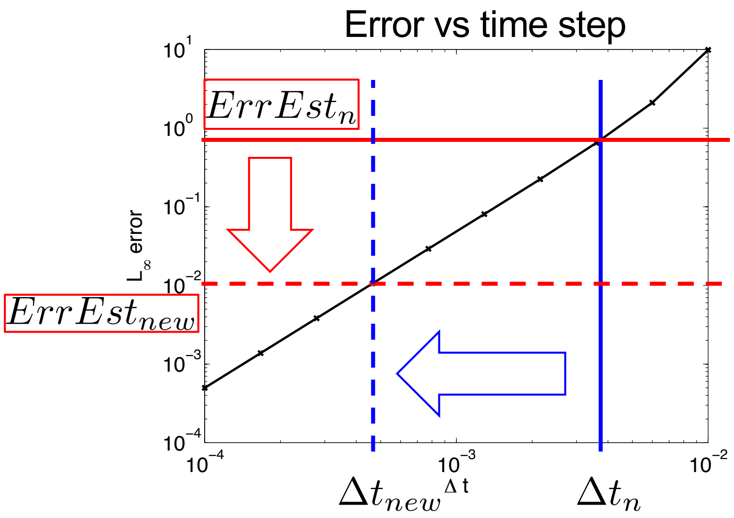
$$Y^{(3)} = y_{[n]} + \frac{3}{4}\Delta t f(Y^{(2)})$$

$$Y^{(4)} = y_{[n]} + \frac{2}{9}\Delta t f(Y^{(1)}) + \frac{1}{3}\Delta t f(Y^{(2)}) + \frac{4}{9}\Delta t f(Y^{(3)})$$

$$y_{[n+1]} = y_{[n]} + \Delta t\left(\frac{2}{9}f(Y^{(1)}) + \frac{1}{3}f(Y^{(2)}) + \frac{4}{9}f(Y^{(3)})\right)$$

$$\tilde{y}_{[n+1]} = y_{[n]} + \Delta t\left(\frac{7}{24}f(Y^{(1)}) + \frac{1}{4}f(Y^{(2)}) + \frac{1}{3}f(Y^{(3)}) + \frac{1}{8}f(Y^{(4)})\right)$$

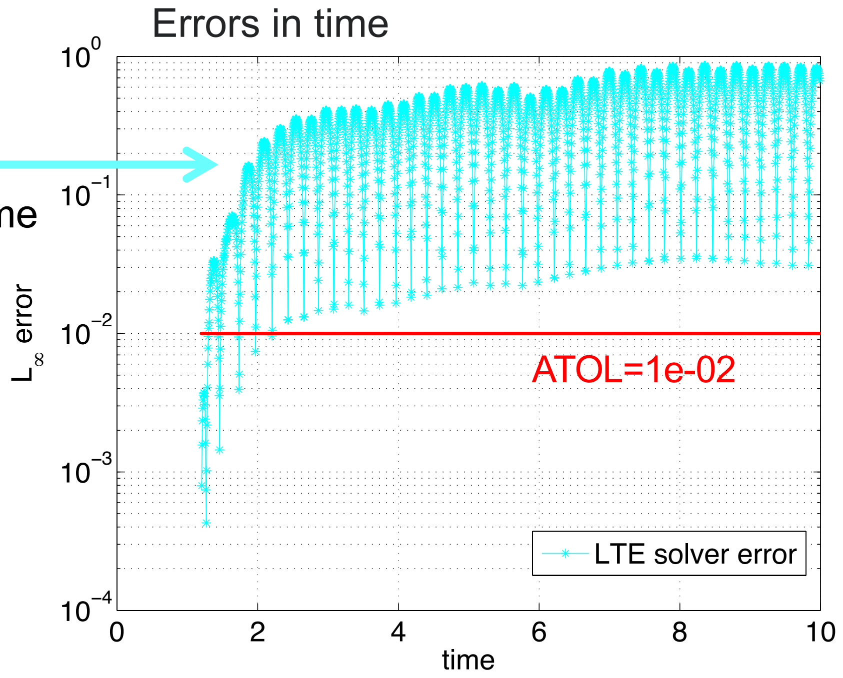
$$ErrEst_n = y_{[n]} - \tilde{y}_{[n]}$$

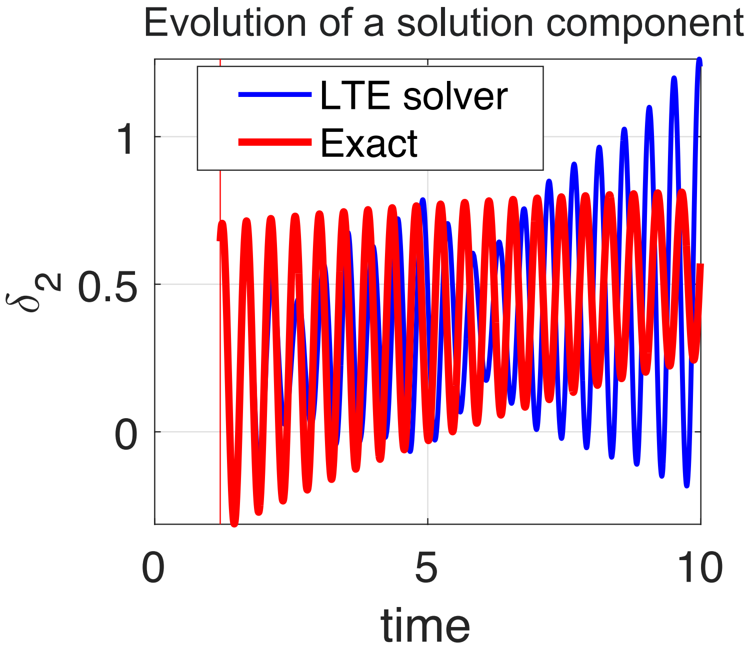


Estimation Can Fail

- Local error estimators do not account for error accumulation, we need global error estimators
- Only local error estimation is present in all software libraries

Global error over time





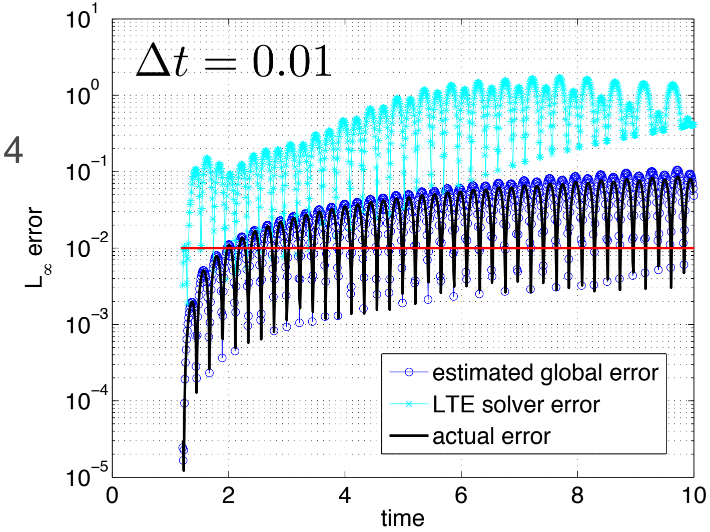
A 3-bus Power Grid System: Error Control Using a 2nd Order GLEE

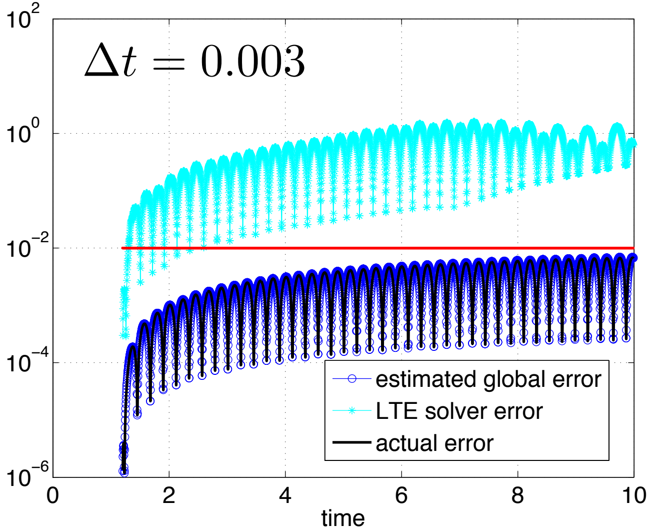
$$\Delta t_{\text{opt}} = \Delta t \left(\frac{\varepsilon(T)}{\text{ATOL}} \right)^{-\frac{1}{p}}$$

- Strategy: two passes (disadvantage – fixed time steps)
- In order to achieve an error of ATOL = 0.01 a time step of 0.0030823 should have been used instead of 0.01

Idea:
1. Take one pass and if not happy, modify time step to achieve error goal
2. control local error and and adjust tolerances after one pass to achieve goal

14





- Proportionality error controller tolerance can also be used

Recap

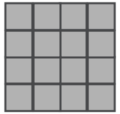
- Many “time” integrators beyond RK4
- Two broad classes: explicit (RK4) and implicit (for stiff systems, e.g., backward Euler)
- High-order integrators provide the best bang for the buck …in principle
- Modern integrators adapt the step size according to an error control mechanism …works well most of the time

Partitioning the Time Integrator

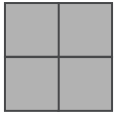
Avoid using a single integrator for a monolithic RHS; employ different integrators for each of the components

We can use a separate integrator for each of the 4 blocks fit for purpose

Some components on the finer mesh may become stiff



Some components don't have interesting dynamics



$$\frac{\partial u_1}{\partial t} = f_{11}(u_1, u_2) + f_{12}(u_1, u_2)$$

$$\frac{\partial u_2}{\partial t} = f_{21}(u_1, u_2) + f_{22}(u_1, u_2)$$

Some components produce instability

Some components produce instability

"Component partitioning"

Most popular players are:

- Implicit/explicit > 100 scale factor :: alleviates stiffness
- Multirate < 100 scale factor :: alleviates global restrictions/local fidelity

"Additive partitioning"

Partitioning the Time Integrator

Avoid using a single integrator for a monolithic RHS; employ different integrators for each of the components

Example of a (bad) implicit-explicit method:

$$\dot{y} = f(y)\,,\ f(y) = f_1(y) + f_2(y); \quad y_{n+1} = y_n + \triangle t f_1(y_n) + \triangle t f_2(y_{n+1})$$

Modern partitioned integrators are high-order, typically required to satisfy coupling conditions.

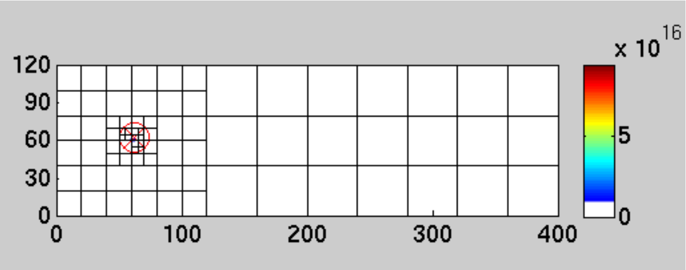
Additive Runge-Kutta: second order, implicit L-stable and second stage-order (stiffly accurate) and conservative; low order embedded and dense output.

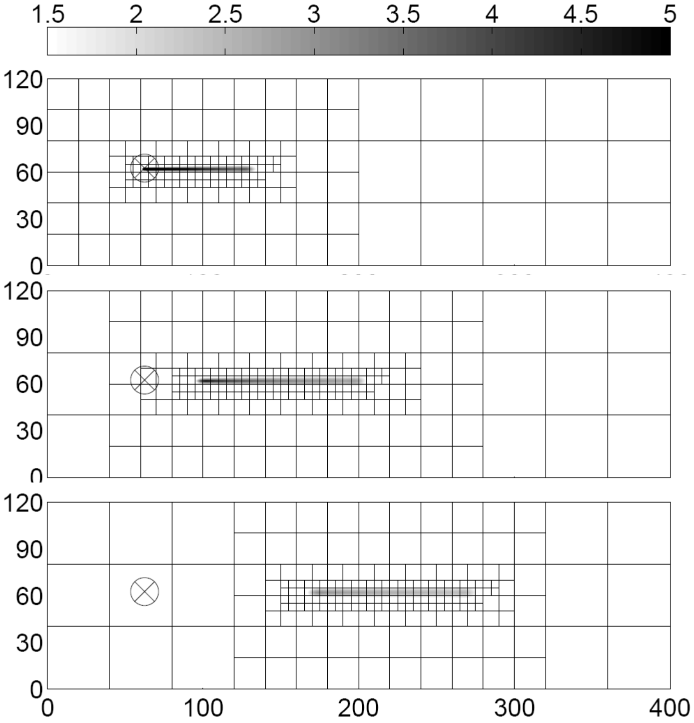
$$\begin{array}{c|ccc} 0 & 0 & & \\ 2-\sqrt{2} & 2-\sqrt{2} & 0 & \\ 1 & 1-a_{32} & a_{32} & 0 \\ \hline & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & 1-\frac{1}{\sqrt{2}} \end{array} \qquad \begin{array}{c|ccc} 0 & 0 & & \\ 2-\sqrt{2} & 1-\frac{1}{\sqrt{2}} & 1-\frac{1}{\sqrt{2}} & \\ 1 & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & 1-\frac{1}{\sqrt{2}} \\ \hline & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & 1-\frac{1}{\sqrt{2}} \end{array}$$

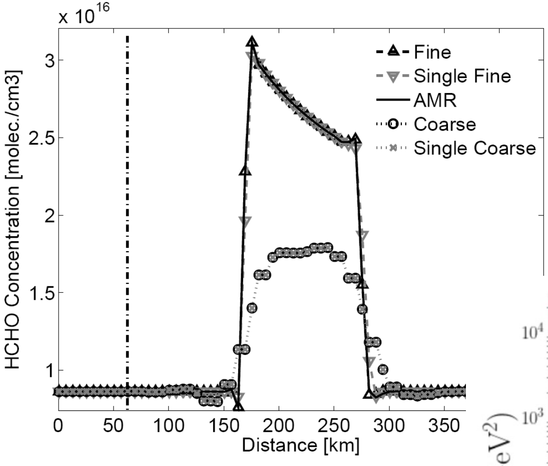
Component Partitioning

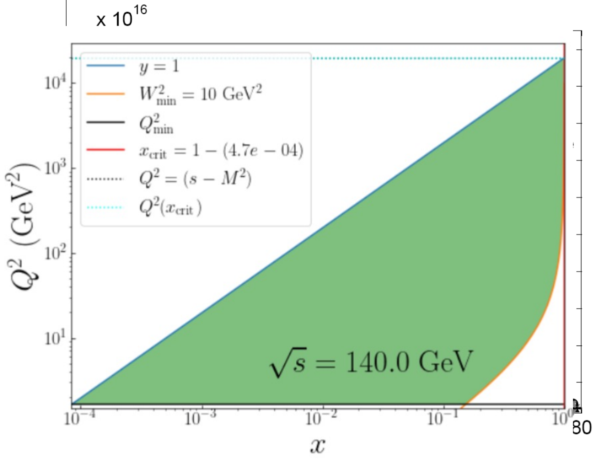
Static or dynamic?

$$\frac{\partial y}{\partial t} = -u\nabla y + K\nabla^2 y$$







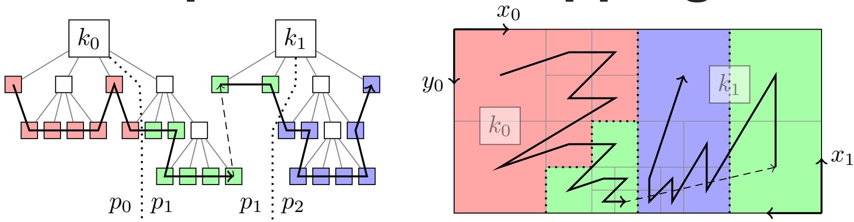


Dynamic AMR for a Relativistic Electron Drift-Kinetic Solver and Scalable PETSc-p4est Implementation and Implicit Time Stepping

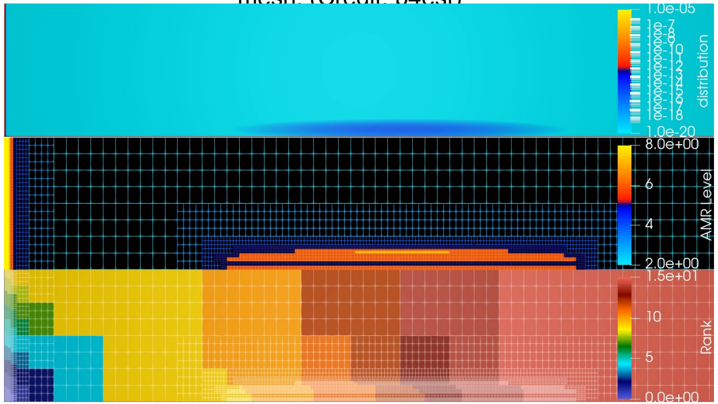
Dynamic adaptive mesh refinement (AMR) in PETSc enables runaway electron simulations (Fokker-Planck PDE) at several orders of magnitude higher resolutions

- We developed a new parallel data management (DM) in PETSc that interfaces AMR capabilities (via p4est library) with physics simulations that require adaptivity
- AMR reduces simulation errors and computational cost by increasing the degrees of freedom only where needed
- Dynamic AMR coarsens and refines meshes to adapt over time as the solution evolves through a dynamical processes
- Fully-implicit time stepping (via PETSc TS) enables accurate solution of stiff dynamical systems

ANL: Johann Rudi, Max Heldman, Emil Constantinescu
LANL: Qi Tang, Xianzhu Tang



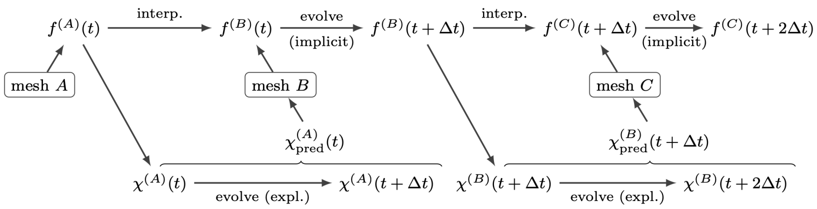
Parallel octree-based AMR. Left: Forest-of-trees topology with 2 trees and leaves are cells of the mesh. Right: Space filling curve to sequentialize cells of mesh. (Credit: p4est)



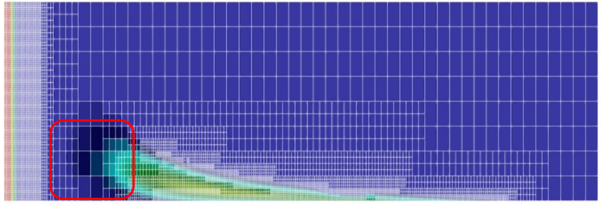
Dynamic AMR in parallel. Each color represents one of 1024 MPI ranks. The aggressive adaptivity required by the application results in 12 levels of difference in refinement, which corresponds to 3 orders of magnitude difference in cell size.

Scalable Implicit Solvers with Dynamic Mesh Adaptation for a Relativistic Drift-Kinetic Fokker-Planck-Boltzmann Model

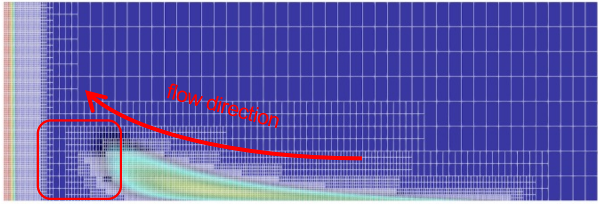
Algorithm for dynamic AMR with prediction. The evolution of an auxiliary function χ is evolved in time separately, indicating where to adapt the mesh



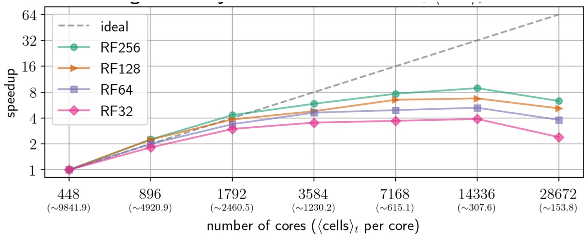
Without prediction



With AMR prediction



Refinement levels of the dynamically adapted mesh (white lines) without prediction vs. AMR with prediction. Note the refined mesh ahead of the flow



Frontera strong scaling – preliminary results

ANL: Johann Rudi, Max Heldman, Emil Constantinescu
LANL: Qi Tang, Xianzhu Tang

Error control and stage predictors

- **Error control**: level of confidence in the numerical solution accuracy: extrapolation & embedded methods (reuse of internal stages)

$$\tilde{y}_{[n+1]} = y_{[n]} + \Delta t \sum_{i=1}^{s} \tilde{b}_i f(Y_{[i]}) + \Delta t \sum_{i=1}^{s} \widehat{\tilde{b}}_i g(Y_{[i]})$$

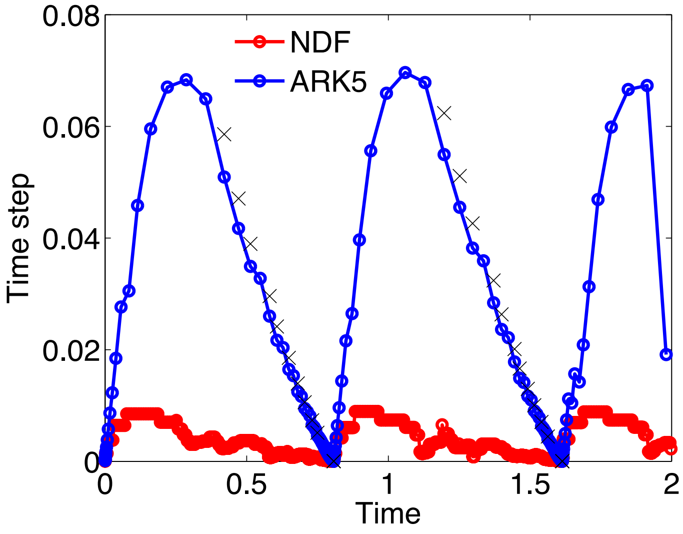
- **Stage predictors** provide "hot-starts" for the solver and reduce the number of iterations

- Develop predictors based on dense output:

$$y(t_{[n]} + \theta \Delta t) = y(t_{[n]}) + \Delta t \sum_{i=1}^{s} b_i^*(\theta) f(Y^{(i)}) + \widehat{b}_i^*(\theta) g(Y^{(i)})$$

- Example: stiff van der Pol oscillator:

$$\begin{bmatrix} \dot{y}_1 \\ y_2 \end{bmatrix} = \varepsilon \begin{bmatrix} 0 \\ (1 - y_1^2) y_2 - y_1 \end{bmatrix} + \begin{bmatrix} y_2 \\ 0 \end{bmatrix}_{\varepsilon = 10^6}$$

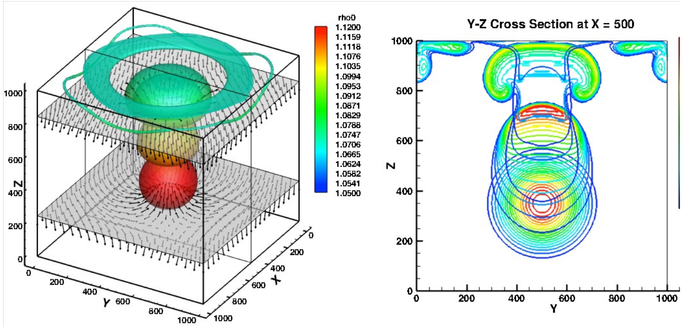


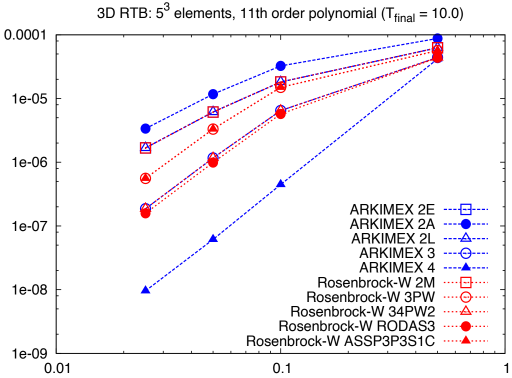
Method order	Predictor order	Newton iterations
3		**104 K**
	2	63 K
4		**38 K**
	2	31 K
	3	26 K
5		**25 K**
	3	20 K

Time Integration for Atmospheric Flows

Results with NUMA – PETSc Interface

Test problem: Rising thermal bubble (benchmark atmospheric flow test case)





Reduce computational cost by stage prediction for nonlinear implicit solves

Meth	Function calls		Nonlinear iter.		Linear iter.		Error	
ARK	W/ pred	W/o pred	W/ pred	W/o pred	W/ pred	W/o pred	W/ pred	W/o pred
2A	**27,156**	32,894	**801**	1,200	**24,755**	29,684	**3.371e-06**	3.371e-06
2E	**17,427**	49,110	**1,601**	2,396	**13,423**	42,721	**1.677e-06**	1.677e-06
3	**29,834**	84,585	**2,402**	3,599	**23,827**	74,987	**1.864e-07**	1.865e-07
4	**36,429**	85,503	**4,000**	4,706	**26,426**	73,379	**9.592e-09**	9.593e-09
5	**32,349**	90,737	**4,138**	5,998	**28,109**	75,536	**2.399e-09**	2.399e-09

Recap

- Many “time” integrators beyond RK4
- Two broad classes: explicit (RK4) and implicit (for stiff systems, e.g., backward Euler)
- High-order integrators provide the best bang for the buck …in principle
- Modern integrators adapt the step size according to an error control mechanism …works well most of the time
- Modern integrators handle adaptivity in time and in “space”
- Computational advantages result from partitioning systems and integration of each partition with different custom methods
- We can reuse the calculated stages to form continuous high-order interpolators
- Interpolators can be used to seed the next step solution when using implicit integrators

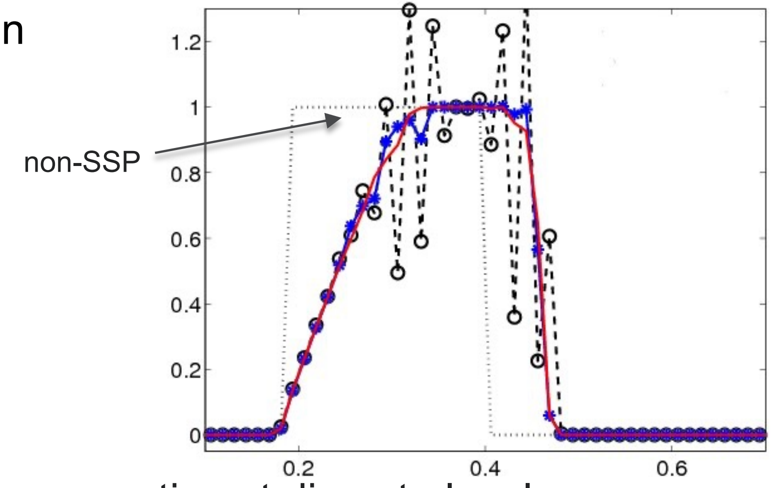
Properties We May Like to Have

1. Preservation of linear or quadratic invariants => **Conservation**
   – Require all methods to be conservative: $\omega^\top y(t) = \text{const.}\,,\ \forall t \Rightarrow \omega^\top y_n = \omega^\top y(0)\,,\ \forall n$

2. **SSP** (strong stability preserving): CFL-like condition

$$||u(t,x)||_{\text{TV}} = \sum_{n=0}^{N-1} |u(t,x_{n+1}) - u(t,x_n)|$$

The higher the order, the worse results



3. **Entropy-stable** and **entropy-preserving**
   – Support entropy-preserving and entropy-stability properties at discrete level
   – The relaxation method applied to IMEX and multirate

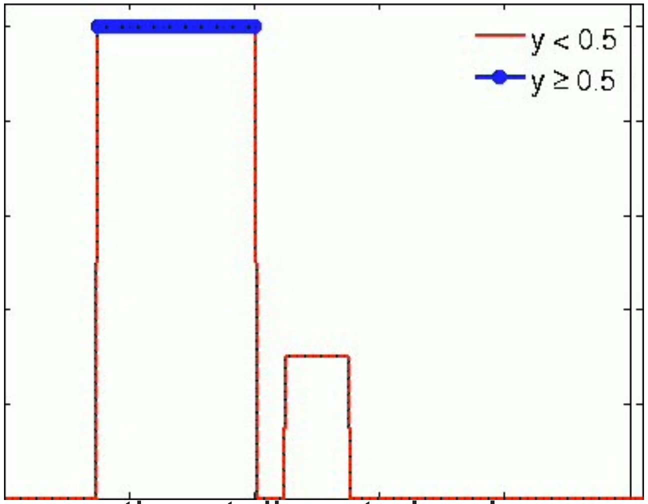
Properties We May Like to Have

1. Preservation of linear invariants => **Conservation**
   - Require all methods to be conservative: $\omega^\top y(t) = \text{const.}\,,\ \forall t \Rightarrow \omega^\top y_n = \omega^\top y(0)\,,\ \forall n$

2. **SSP** (strong stability preserving): CFL-like condition

$$||u(t,x)||_{\text{TV}} = \sum_{n=0}^{N-1} |u(t,x_{n+1}) - u(t,x_n)|$$

$$\frac{\partial y}{\partial t} + y\frac{\partial y}{\partial x} = 0$$



3. **Entropy-stable** and **entropy-preserving**
   - Support entropy-preserving and entropy-stability properties at discrete level
   - By the relaxation method applied to IMEX and multirate

Symplecticity:

- Condition: $b_i a_{ij} + b_j a_{ji} = b_i b_j$

Problem: $\ddot{q} = f(q)$

Störmer-Verlet:

$$p_{n+1/2} = p_n + \frac{\Delta t}{2} f(q_n)$$
$$q_{n+1} = q_n + \Delta t p_{n+1/2}$$
$$p_{n+1} = p_{n+1/2} + \frac{\Delta t}{2} f(q_{n+1})$$

Störmer-Verlet as Runge-Kutta

0	0	0
1	1/2	1/2
	1/2	1/2

1/2	1/2	0
1/2	1/2	0
	1/2	1/2

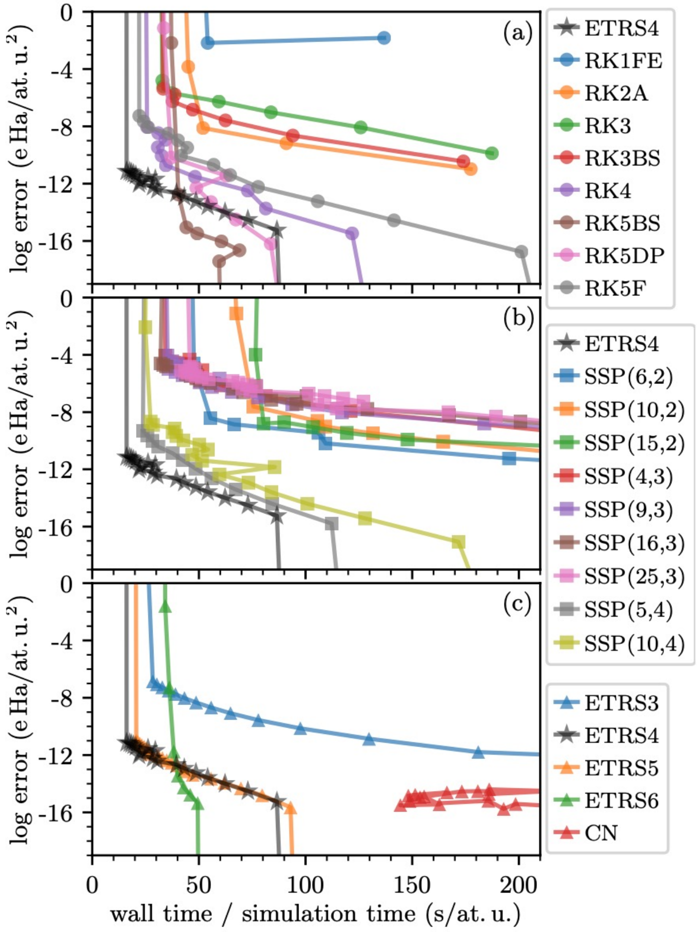
the power of abstraction

$$\{q,p\}_{[n+1]} = e^{\frac{1}{2}\Delta t f(.)} e^{\Delta t p} e^{\frac{1}{2}\Delta t f(.)} \{q,p\}_{[n]}$$

Reversibility:

Time-reversible schemes (time-reversal symmetry)

TD-DFT – invariant wrt the direction of time



Recap

- Many “time” integrators beyond RK4
- Two broad classes: explicit (RK4) and implicit (for stiff systems, e.g., backward Euler)
- High-order integrators provide the best bang for the buck …in principle
- Modern integrators adapt the step size according to an error control mechanism …works well most of the time
- Modern integrators handle adaptivity in time and in “space”
- Computational advantages result from partitioning systems and integration of each partition with different custom methods
- We can reuse the calculated stages to form continuous high-order interpolators
- Interpolators can be used to seed the next step solution when using implicit integrators
- Some integrators preserve symplecticity, monotonicity, and positivity in addition to the above

Solvers' Ecosystems

- Solvers available in small packages addons (Python, Jax, …) are limited/not sophisticated
- Matlab/Julia solvers are well-tested and developed but do not scale
- DOE software libraries can be used for prototyping and scaling
  - PETSc – Argonne solver library provides a hundreds of solvers; scale to HPC
  - Trilinos (developed at Sandia)
  - SUNDIALS (and extensions) developed at Livermore
  - All provide access to many sophisticated methods
- Adaptive meshing:
  - P4est (Parallel AMR on Forests of Octrees)
  - ParMETIS (Parallel Graph Partitioning and Fill-reducing Matrix Ordering)
  - FLASH <- Paramesh (see Anshu's talk)

Portable Extensible Toolkit for Scientific Computation

Open-source numerical library for large-scale parallel computation

Portability

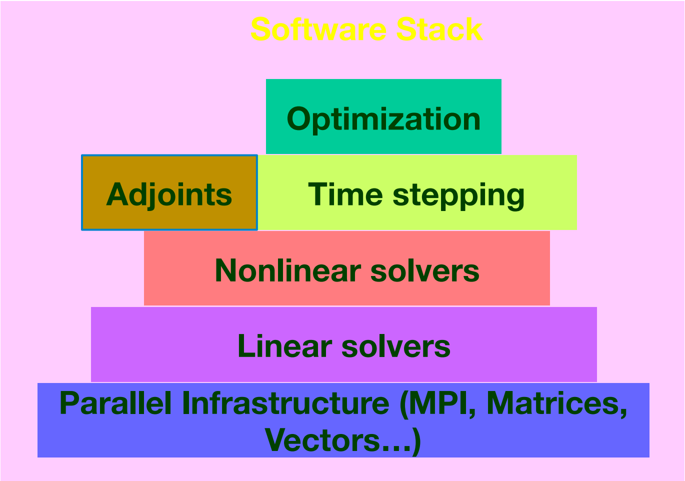
- Unix, Linux, MacOS, Windows, GPUs
- 32/64 bit, real/complex, ...
- C, C++, Fortran, Python, Julia, Matlab

Extensibility

- ParMETIS, SuperLU_Dist, MUMPS, hypre, UMFPACK, Sundials, Elemental, ScaLAPACK, UMFPack, …

Toolkit

- Iterative solvers and preconditioners
- Parallel nonlinear solvers
- Time-stepping (ODE and DAE) solvers
- Adjoint sensitivity analysis
- Event support
- Support for network data structures
- Optimization



Scalable Solver Suite for ODEs/DAEs/PDEs

TS Name	Reference	Class	Type	Order
euler	forward Euler	one-step	explicit	1
ssp	multistage SSP [Ket08]	Runge-Kutta	explicit	≤ 4
rk*	multiscale	Runge-Kutta	explicit	≥ 1
beuler	backward Euler	one-step	implicit	1
cn	Crank-Nicolson	one-step	implicit	2
theta*	theta-method	one-step	implicit	≤ 2
bdf	Backward Differentiation Formulas	one-step	implicit	≤ 6
alpha	alpha-method [JWH00]	one-step	implicit	2
gl	general linear [BJW07]	multistep-multistage	implicit	≤ 3
eimex	extrapolated IMEX [CS10]	one-step	IMEX	≥ 1, adaptive
dirk	DIRK	diagonally implicit Runge-Kutta	implicit	≥ 1
arkimex	See IMEX Runge-Kutta schemes	IMEX Runge-Kutta	IMEX	$1 - 5$
rosw	See Rosenbrock W-schemes	Rosenbrock-W	linearly implicit	$1 - 4$
glee	See GL schemes with global error estimation	GL with global error	explicit and implicit	$1 - 3$
mprk	Multirate Partitioned Runge-Kutta	multirate	explicit	$2 - 3$
basicsymplectic	Basic symplectic integrator for separable Hamiltonian	semi-implicit Euler and Velocity Verlet	explicit	$1 - 2$
irk	fully implicit Runge-Kutta	Gauss-Legrendre	implicit	$2s$

Name	Reference	Stages (IM)	Order (Stage)	IM	SA	Embed	DO	Remarks
a2	based on CN	2 (1)	2 (2)	A-Stable	yes	yes (1)	yes (2)	
l2	SSP2(2,2,2) [PR05]	2 (2)	2 (1)	L-Stable	yes	yes (1)	yes (2)	SSP SDIRK
ars122	ARS122 [ARS97]	2 (1)	3 (1)	A-Stable	yes	yes (1)	yes (2)	
2c	[GKC13]	3 (2)	2 (2)	L-Stable	yes	yes (1)	yes (2)	SDIRK
2d	[GKC13]	3 (2)	2 (2)	L-Stable	yes	yes (1)	yes (2)	SDIRK
2e	[GKC13]	3 (2)	2 (2)	L-Stable	yes	yes (1)	yes (2)	SDIRK
prssp2	PRS(3,3,2) [PR05]	3 (3)	3 (1)	L-Stable	yes	no	no	SSP
3	[KC03]	4 (3)	3 (2)	L-Stable	yes	yes (2)	yes (2)	SDIRK
bpr3	[BPR11]	5 (4)	3 (2)	L-Stable	yes	no	no	SDIRK
ars443	[ARS97]	5 (4)	3 (1)	L-Stable	yes	no	no	SDIRK
4	[KC03]	6 (5)	4 (2)	L-Stable	yes	yes (3)	yes	SDIRK
5	[KC03]	8 (7)	5 (2)	L-Stable	yes	yes (4)	yes (3)	SDIRK

Unconstrained

Algorithm	Associated Type	Objective	Gradient	Hessian	Constraints	Jacobian
Nelder-Mead	TAONM	X				
Conjugate Gradient	TAOCG	X	X			
Limited Memory Variable Metric (quasi-Newton)	TAOLMVM	X	X			
Orthant-wise Limited Memory (quasi-Newton)	TAOOWLQN	X	X			
Bundle Method for Regularized Risk Minimization	TAOBMRM	X	X			
Newton Line Search	TAONLS	X	X	X		
Newton Trust Region	TAONTR	X	X	X		

Optimization in PETSc

Bound Constrained

Algorithm	Associated Type	Objective	Gradient	Hessian	Constraints	Jacobian	Constraint Type
Bounded Conjugate Gradient	TAOBNCG	X	X				Box constraints
Bounded Limited Memory Variable Metric (Quasi-Newton)	TAOBLMVM	X	X				Box constraints
Bounded Quasi-Newton Line Search	TAOBQNLS	X	X				Box constraints
Bounded Newton Line Search	TAOBNLS	X	X				Box constraints
Bounded Newton Trust-Region	TAOBNTR	X	X				Box constraints
Gradient Projection Conjugate Gradient	TAOGPCG	X	X				Box constraints
Bounded Quadratic Interior Point	TAOBQPIP	X	X				Box constraints
Tron	TAOTRON	X	X	X			Box constraints

Optimization in PETSc

Constrained

Algorithm	Associated Type	Objective	Gradient	Hessian	Constraints	Jacobian	Constraint Type
Interior Point Method	`TAOIPM`	X	X	X	X	X	General Constraints
Barrier-Based Primal-Dual Interior Point	`TAOPDIPM`	X	X	X	X	X	General Constraints

Complementarity

Algorithm	Associated Type	Objective	Gradient	Hessian	Constraints	Jacobian	Constraint '
Active-Set Feasible Line Search	`TAOASFLS`				X	X	Complemen
Active-Set Infeasible Line Search	`TAOASILS`				X	X	Complemen
Semismooth Feasible Line Search	`TAOSSFLS`				X	X	Complemen
Semismooth Infeasible Line Searchx	`TAOSSILS`				X	X	Complemen

Nonlinear Least Squares

Algorithm	Associated Type	Objective	Gradient	Hessian	Constraints	Jacobian	Constraint Type
POUNDERS	`TAOPOUNDERS`	X					Box Constraints

PDE-Constrained

Algorithm	Associated Type	Objective	Gradient	Hessian	Constraints	Jacobian	Constraint Type
Linearly Constrained Lagrangian	`TAOLCL`	X	X	X	X	X	PDE Constraints

Summary

- Many "time" integrators beyond RK4
- Two broad classes: explicit (RK4) and implicit (for stiff systems, e.g., backward Euler)
- High-order integrators provide the best bang for the buck …in principle
- Modern integrators adapt the step size according to an error control mechanism …works well most of the time
- Modern integrators handle adaptivity in time and in "space"
- Computational advantages result from partitioning systems and integration of each partition with different custom methods
- We can reuse the calculated stages to form continuous high-order interpolators
- Interpolators can be used to seed the next step solution when using implicit integrators
- Some integrators preserve symplecticity, monotonicity, and positivity in addition to the above
- Open-source software that implements these algorithms+ is available from DOE

Funding: This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of NP, Scientific Discovery through Advanced Computing (SciDAC) program and U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under contract DE-AC02-06CH11357