TEMPO - a one step alternative to SUNDIALS

Philipp Birken

Lund University

PDESoft 2014
Heidelberg, July 15th, 2014

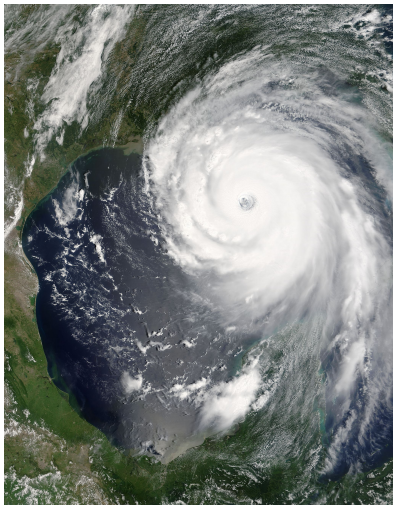
Outline

Outline

Is implicit hard?

- PDE people know little about time integrators and solvers
- Lots of misconceptions
- Implicit time integration smears the solution
- Implicit is less accurate
- Need same order in space and time
- Implicit is hard
- Well...
- True: Implicit offers lots of opportunities to do things wrong
- Solution: Let TEMPO (or SUNDIALS) help you with that!

Unsteady flow problems



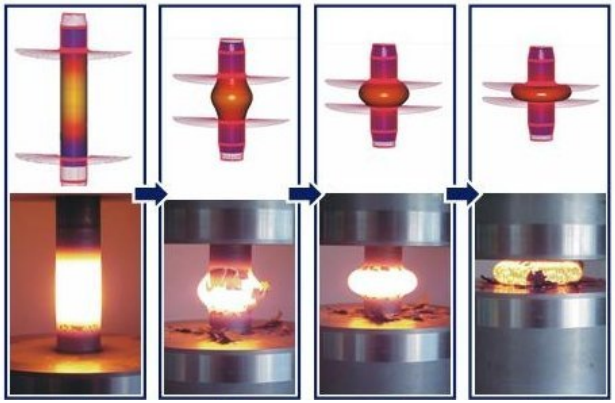
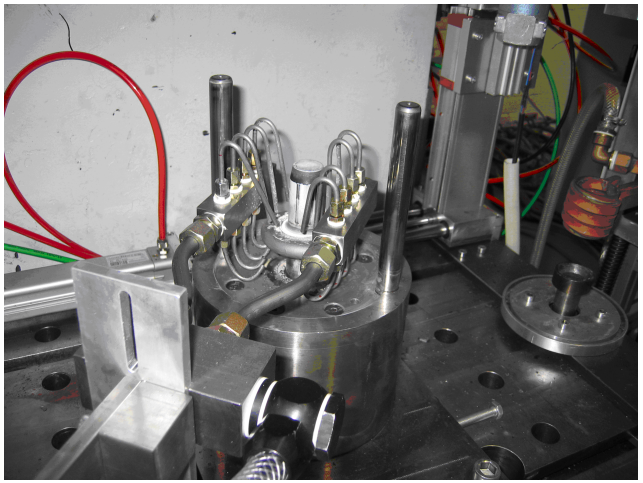



Figure: Hurricane Katrina, NASA, PD; Lillgrund Offshore windfarm, Mariusz Padziora, CC-by-sa 3.0 via Wikimedia Commons; Gas Quenching, Steinhoff

Outline

Mathematical setting

- Time dependent nonlinear systems of PDEs
- Method of Lines
- Possibly discretized using high order methods in space
- Thus initial value value problem

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u}), \quad \mathbf{u}(t_0) = \mathbf{u}_0$$

- Here, $\mathbf{u}$ is large, possibly a hundred millions
- $\mathbf{f}$ is nonlinear with a sparse block Jacobian
- Problem is typically stiff.
- $\rightarrow$ implicit time integration necessary.

Aim: Parallel low storage Solver

- Form of equation is independent of precise time integration scheme

$$\mathbf{u} = \alpha \Delta t \mathbf{f}(\mathbf{u}) + \psi.$$

$\rightarrow$ Need for fast low storage parallel scaling solvers

- Preconditioned inexact Jacobian-Free Newton-Krylov (JFNK)
- Aim: Library that does all this, given an IVP

Figure: Cray Hermit in Stuttgart; Bild: ThE cRaCkEr, CC-by-sa 3.0, via Wikimedia Commons

Implicit Time Integration

- Large stability region required: A-stable methods!
- Unsteady problem: Higher order and time adaptivity.
- BDF often used, but does not fit the profile.
- Bijl et al (01,02): ESDIRK methods competitive!
- One explicit and subsequent backward Euler steps.
- Time adaptivity easy.

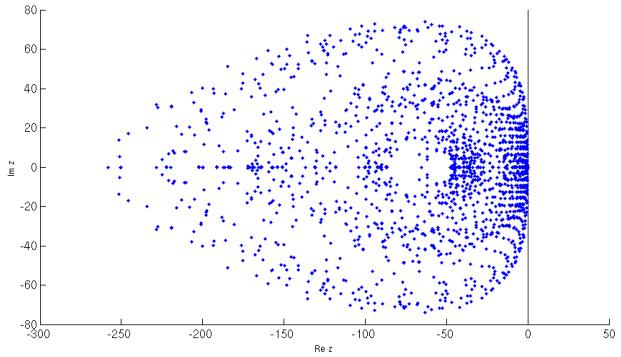


Figure: DG discretization, Re=100, 4th degree polynomial

One step methods

- Advantages: No starting procedure necessary, no problems with change in properties of rhs (e.g. grid adaptivity)
- ESDIRK, SDIRK
- Consist of several steps of implicit Euler with different time step sizes and starting vectors
- Rosenbrock
- Linearized SDIRK
- Linear systems thus have same structure as for SDIRK, ESDIRK

Time adaptivity

- Explicit: Time step bounded by stability
- Implicit: Time step bounded by accuracy
- Time adaptivity crucial for efficiency of code
- Error estimate easy at low cost using embedded methods
- Several time step selectors implemented:
- Standard I controller with limiters
- PI controller PI.4.2
- PC11
- Digital Filter based, H211PI (Söderlind '06)

Inexact Jacobian-free Newton-Krylov

- Inexact Newton method from Eisenstat/Walker '96
- Allows quadratic convergence with very coarse linear system solves
- Solve systems using Krylov method, e.g. GMRES
- Krylov methods don't need Jacobian, only matrix vector products
- Approximate $\frac{\partial \mathbf{f}}{\partial \mathbf{u}} q$ by finite differences:

$$\mathbf{A}\mathbf{q} = \frac{\partial \mathbf{f}(\mathbf{u}^{(k)})}{\partial \mathbf{u}}\mathbf{q} \approx \frac{\mathbf{f}(\mathbf{u}^{(k)} + \epsilon \mathbf{q}) - \mathbf{f}(\mathbf{u}^{(k)})}{\epsilon}.$$

- Immense flexibility
- Method is quadratically convergent in large radius!
- Means that crucial function is the right hand side $\mathbf{f}(\mathbf{u})$
- Low storage solver there

Feedback-Loops for DIRK scheme

Given error tolerances τ, initial time t_0 and time step size Δt_0

- For $i = 1, ..., s$
  - For $k = 0, 1, ...$ until termination criterion with tolerance $\tau/5$ is satisfied or MAX_NEWTON_ITER has been reached
    - Determine Eisenstat-Walker relative tolerance
    - Solve linear system using preconditioned GMRES
- If MAX_NEWTON_ITER has been reached, but the tolerance test has not been passed, repeat time step with $\Delta t_n = \Delta t_n/4$
- Estimate local error and compute new time step size Δt_{n+1}
- $t_{n+1} = t_n + \Delta t_n$

Note: Puts additional bound on time step via nonlinear solver

Preconditioning

- Big issues with regards to low memory and parallel
- Two options available
- First: Compute and store the jacobian and use it to compute a preconditioner
- Read: ILU
- Needs: Function to compute the Jacobian, does the rest itself
- Sparse Block Matrix class available
- Second: Just provide a function pointer that has a vector as in and output and represents the preconditioner
- Read: Multigrid

Outline

Development so far

- Started from the DLR-TAU-Finite-Volume solver for steady flows as done in the nineties by Thomas Sonar, Andreas Meister and others
- Extended to unsteady problems by me ten years later
- Technology for my habilitation thesis "Numerical Methods for the Unsteady Compressible Navier-Stokes Equations"
- Implemented principles in other codes as well myself, e.g. UFLO (Jameson; Stanford), HALO (Gassner, Munz; U Stuttgart, Cologne)
- Got tired of that $\rightarrow$ Library
- Lots of help from students with money from DFG (SFB TRR 30)
- Subversion repository and redmine instance at U Kassel

Code framework

- Basic data structure: Vector
- Currently our own vector class
- Question: Should we use a specific other class?
- PETSc class? Blitz++?
- Uses function pointer to represent right hand side
- Used in time integration, Newton and Krylov solver
- Two function pointers to represent IMEX

Classes and structures

- Set all params and function pointers needed in struct TEMPO_time_integrator
- Pass this to constructor of Time Integration class
- One separate constructor for each type of time integration
- Sets up all vectors etc.
- Own classes for JFNK, Krylov stuff
- Actual call is perform_timestep

Alternatives

- SUNDIALS
- Hindmarsh et. al., US National Labs
- Uses adaptive time step and order in BDF setting
- Very similar mathematics otherwise
- ARKOde
- Additive RK methods, soon to be part of SUNDIALS
- Need to check it out!
- TEMPO: +Rosenbrock, computationally more stable

Problems

Copyright

- Origin of TEMPO: In house Flow solver
- Origin of In house flow solver: TAU-Code of the 90s
- Causes hassle now. And is hassle in the first place!

Programming skills

- In my environment, I'm probably the most skilled coder
- I'm good, but not great at all
- Students don't know pointers, libraries, Makefiles, version control, templates, parallel stuff...
- Need more Scientific Computing education!

Outline

Efficiency of JFNK

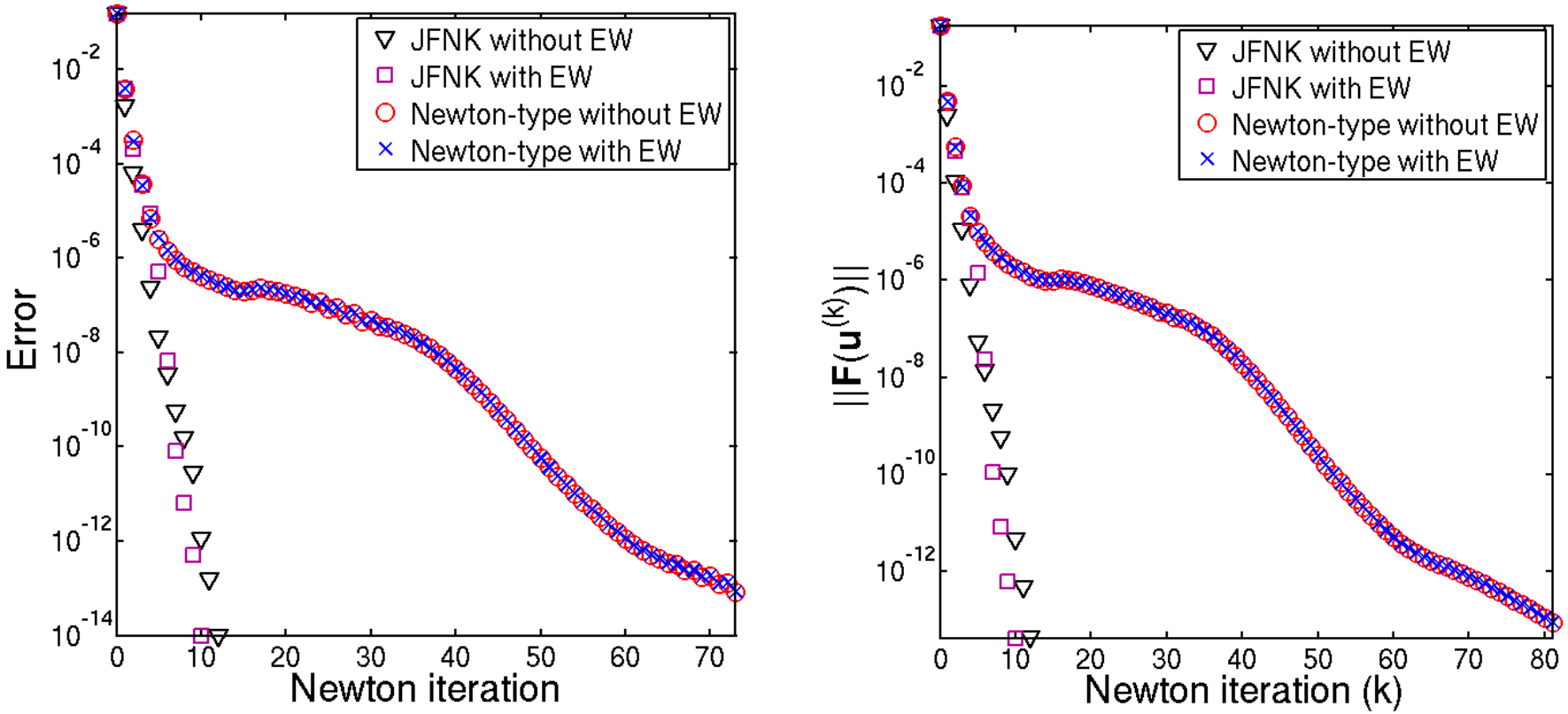


Figure: Relative errors (left) and relative residuals (right).

Efficiency of JFNK

JFNK-FT	JFNK-EW	Newton-type-FT	Newton-type-EW
3.0	4.1	0.7	0.9

Table: Upper bounds of convergence radius for Shu vortex problem in terms of CFL numbers. FT stands for a fixed tolerance, EW for Eisenstat-Walker.

Simulation: Flanged Shaft

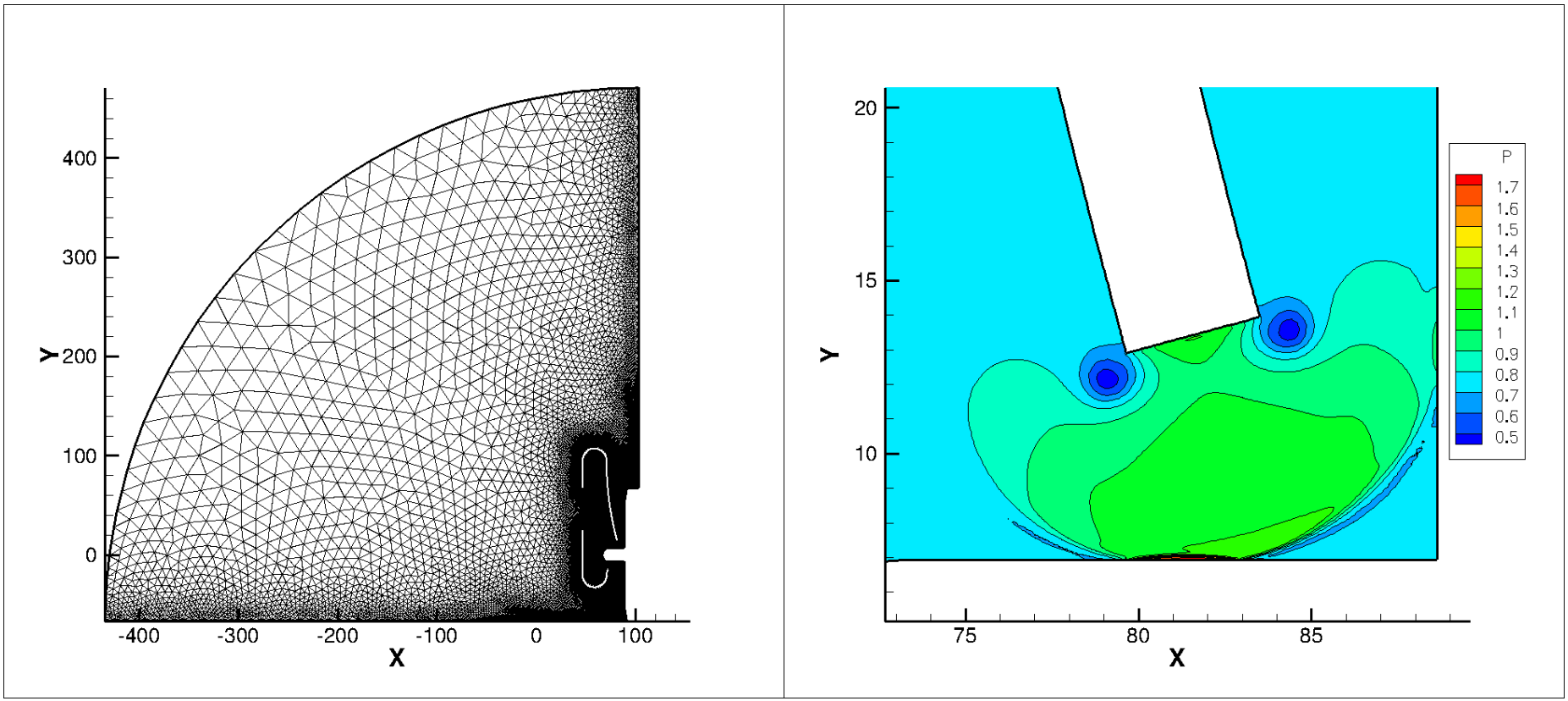


Figure: Pressure contours after 5 seconds. Left: Zoom on region around lower tube and shaft. Right: Zoom on region around upper tube and shaft.

Efficiency of Time Integration schemes

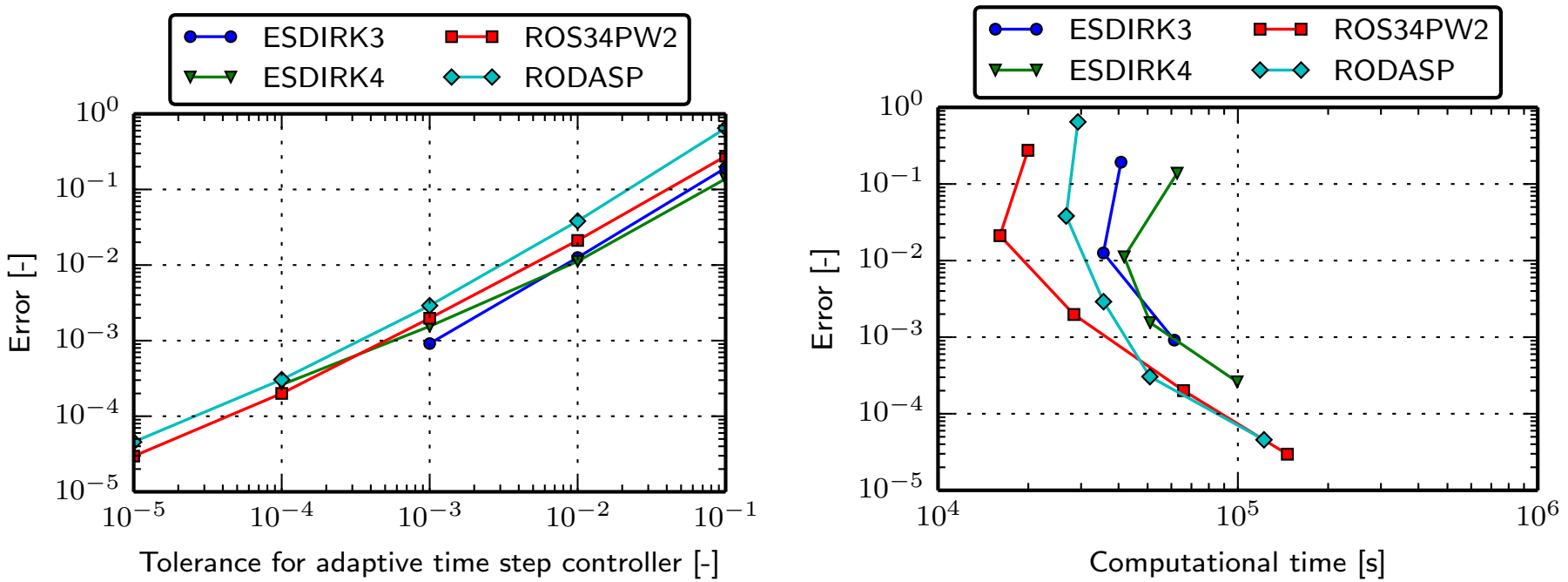


Figure: Adaptive time stepping: accuracy and computational efficiency for different time integration schemes for the cooling of a flanged shaft test case

Summary and Outlook

- TEMPO solves stiff problems
- ESDIRK, SDIRK and Rosenbrock
- Jacobian-free framework
- Time adaptivity
- Efficiency from smart choices of tolerances
- Interface: Function pointer with in and out vector
- Next up: Copyright, then publish library
- Soon: Stage value extrapolation for starting guess in Newton
- Later: Output for videos via Dense Output Formulas