Review/Preface to Second Day of DD-15 Tutorial

William D. Gropp

Lab for Advanced Numerical Software

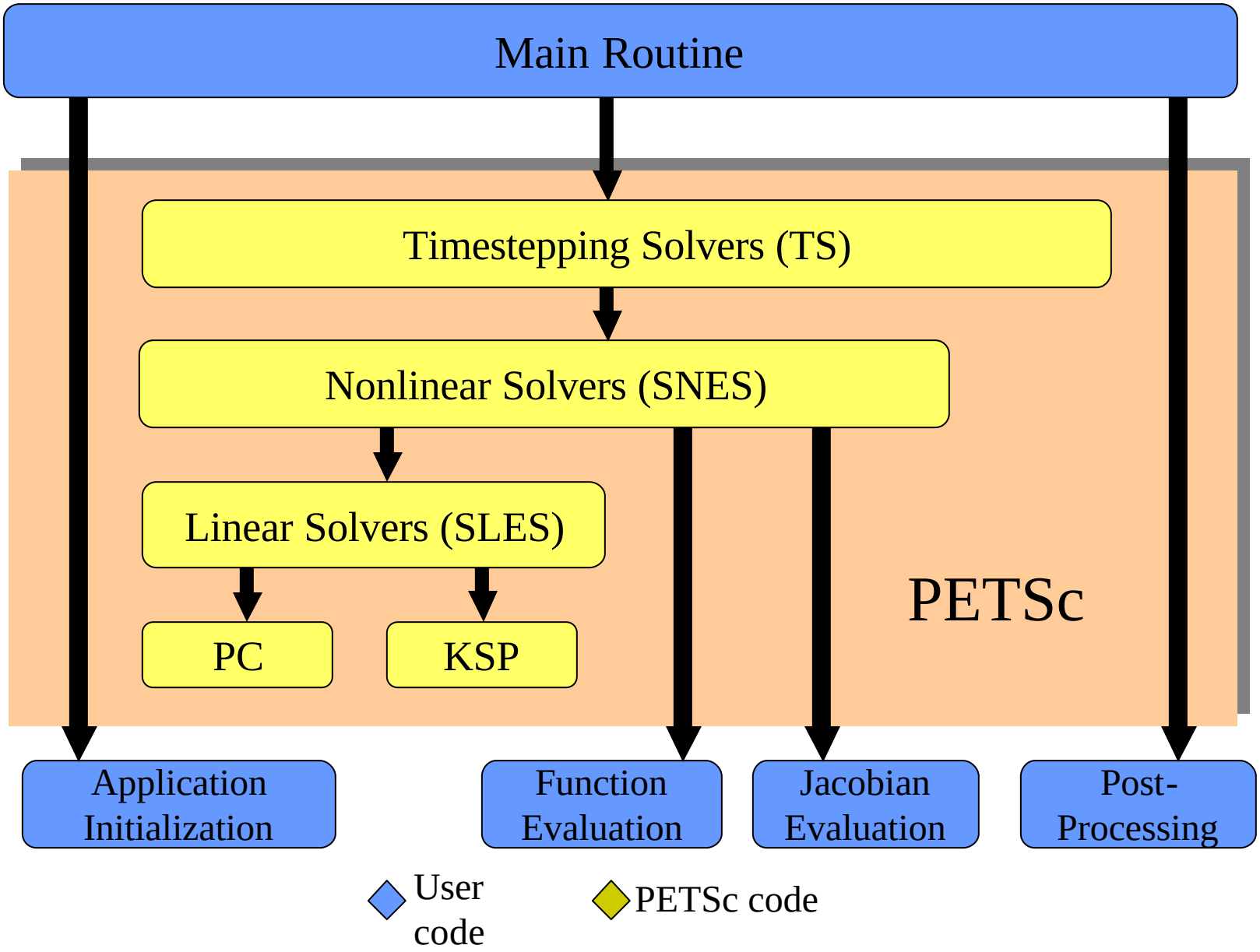
Argonne National Laboratory

David E. Keyes

Department of Applied Physics & Applied Mathematics

Columbia University

PETSc's SLES & SNES: objects and algorithms



PETSc objects

- Vector and matrix objects, Vec and Mat
  - Accessed and used as mathematical objects
  - Code usage independent of runtime choice of data structures
  - Implemented to allow for high performance
    - Split-phase operations
    - Overlap of communication and computation
    - Aggregation, locality, low memory footprint
- Cartesian grid object DA
- Solver objects SLES and SNES
  - Large collection of progressive algorithmic options assembled under abstract interfaces
  - Recursively callable, with separate contexts for sub-objects
  - Command-line tunable
- Auxilliary objects (e.g., Viewer)

Basic and advanced algorithms

- **Linear solvers**
  - **Schwarz: single-level, two-level**
  - **Schur: reduced, full (interface-interior, primal-dual)**
  - **Schwarz-Schur hybrids**
- **Nonlinear solvers**
  - **Newton-Krylov-Schwarz (NKS)**
  - **Jacobian-free Newton-Krylov (JFNK)**
- **Implicit time integrators**
  - **Time-accurate time stepping (PETSc's TS)**
  - **Pseudo-transient continuation (? tc)**

Scalability properties of DD

- **Parallel scaling (time per iteration)**
  - **Good "surface-to-volume" ratio: communication subdominant to computation, Amdahl's law defeated**
  - **Reasonable communication requirements: mostly nearest neighbor, of small size when global**
  - **Can increase processors without bound with increased system size, at power-law rate that depends upon richness of communication network**
- **Convergence scaling (iterations needed for solution)**
  - **Schwarz and Schur can have bounded condition number**
  - **Newton can have quadratic convergence**
  - **Nonlinear robustification techniques**

So far…

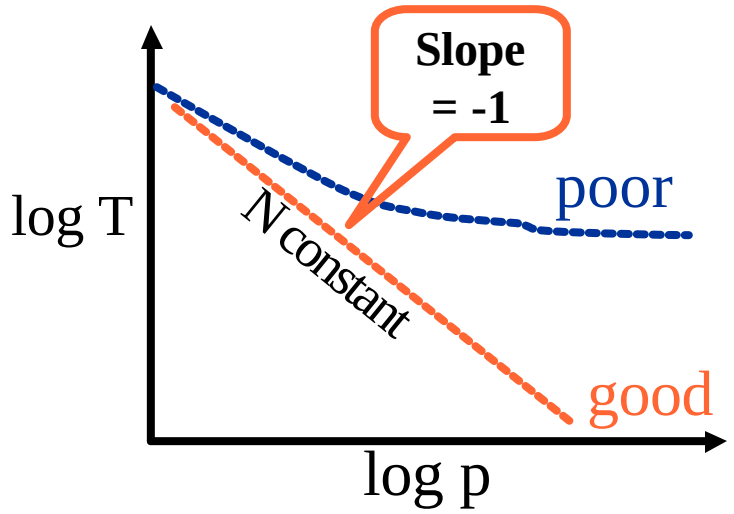
- **Good stories mentioned …**
  - **1999 Gordon Bell Prize for aerodynamic computation using NKS (PETSc, Argonne)**
  - **2002 Gordon Bell Prize for structural dynamics computation using FETI-DP (Salinas, Sandia)**
  - **Combustion model for ? tc**
- **But only simple examples presented in detail …**
  - **Poisson problem**
  - **Bratu problem**
  - **Driven cavity**

This morning …

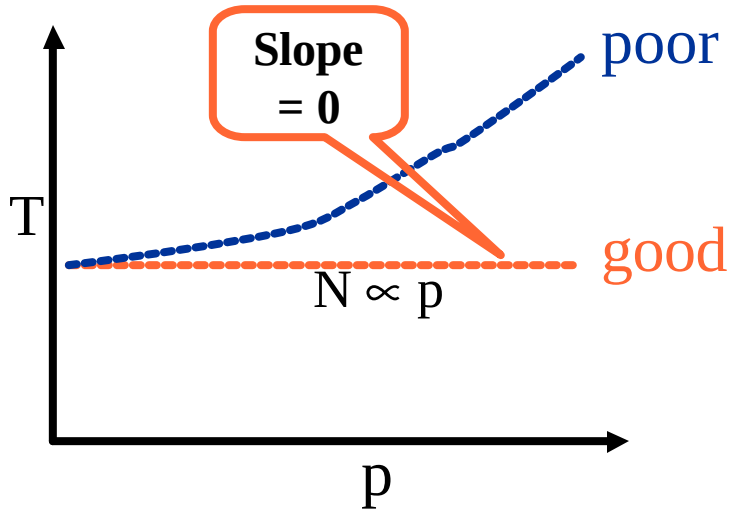
- **Brief review**
- **More advanced algorithms**
- **Programmatic context of current PETSc development environment**
  - **Mathematicians and computer scientists, *come join the fun* ☺**
- **Broader sense of PDE-type scientific applications being addressed through PETSc**
  - **Applications scientists, *come join the fun* ☺**

Two distinct definitions of scalability

- **"Strong scaling"**
  - **execution time decreases in inverse proportion to the number of processors**
  - ***fixed size problem overall***



- **"Weak scaling"**
  - **execution time remains constant, as problem size and processor number are increased in proportion**
  - ***fixed size problem per processor***
  - **also known as "Gustafson scaling"**



Three important usages of "efficiency"

- **For any iterative method,** $time = (\#of\ iters) \times (time\ per\ iter)$
- **"Convergence efficiency"**

$$h_{\text{conv}} = \frac{(\#of\ iters\ \ for\ P_{\text{small}})}{(\#of\ iters\ \ for\ P_{\text{large}})}$$

- **"Implementation efficiency"**

$$h_{\text{impl}} = \frac{(time\ per\ iter\ \ for\ P_{\text{small}})}{(time\ per\ iter\ \ for\ P_{\text{large}})} \times \frac{P_{\text{small}}}{P_{\text{large}}}$$

- **Overall efficiency**

$$h = h_{\text{conv}} \times h_{\text{impl}} = \frac{(time\ \ for\ P_{\text{small}})}{(time\ \ for\ P_{\text{large}})} \times \frac{P_{\text{small}}}{P_{\text{large}}}$$

- **"Ideal" is unity for all three measures**

PDE varieties

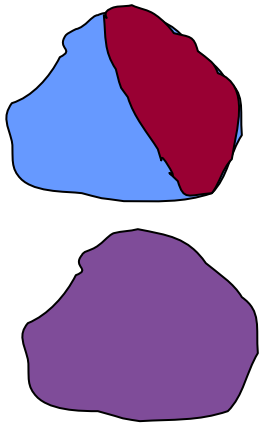
- **Evolution (*time hyperbolic, time parabolic*)**

$$\frac{\partial}{\partial t}(\bullet)+\nabla\cdot(f(\bullet))=\ldots, \quad \frac{\partial}{\partial t}(\bullet)-\nabla\cdot(a\nabla\bullet)=\ldots$$

- **Equilibrium (*elliptic, spatially hyperbolic or parabolic*)**

$$\nabla\cdot(U\bullet-a\nabla\bullet)=\ldots$$

- **Mixed, varying by region**
- **Mixed, of multiple type**
  (e.g., parabolic with elliptic constraint)



Common features

- **Despite enormous variety of mathematical analysis required, most computational algorithms for PDEs can be described with surprisingly few common bulk-synchronous (SPMD) kernels**
  - **Domain-decomposed loops over mesh points manipulating purely local data**
  - **Domain-decomposed loops over mesh points (or edges) manipulating own and near neighbors' data**
  - **Global reductions and recurrences**
- **The large variety of operations, yet simplicity of operation class, invites abstraction and performance optimization: in short, an object oriented library**

Questions?

- Now is a great time for them!
- Next, we go on to nonlinear Schwarz and advanced concepts in nonlinear solver software, to finish Lecture #2 from Thursday afternoon
- Then we start Lecture #3 on applications this morning
- This afternoon's Lecture #4 is on two research topics: physics-based preconditioning and PDE-constrained optimization
- Whatever we don't finish will be posted on line by this evening

Nonlinear Schwarz preconditioning

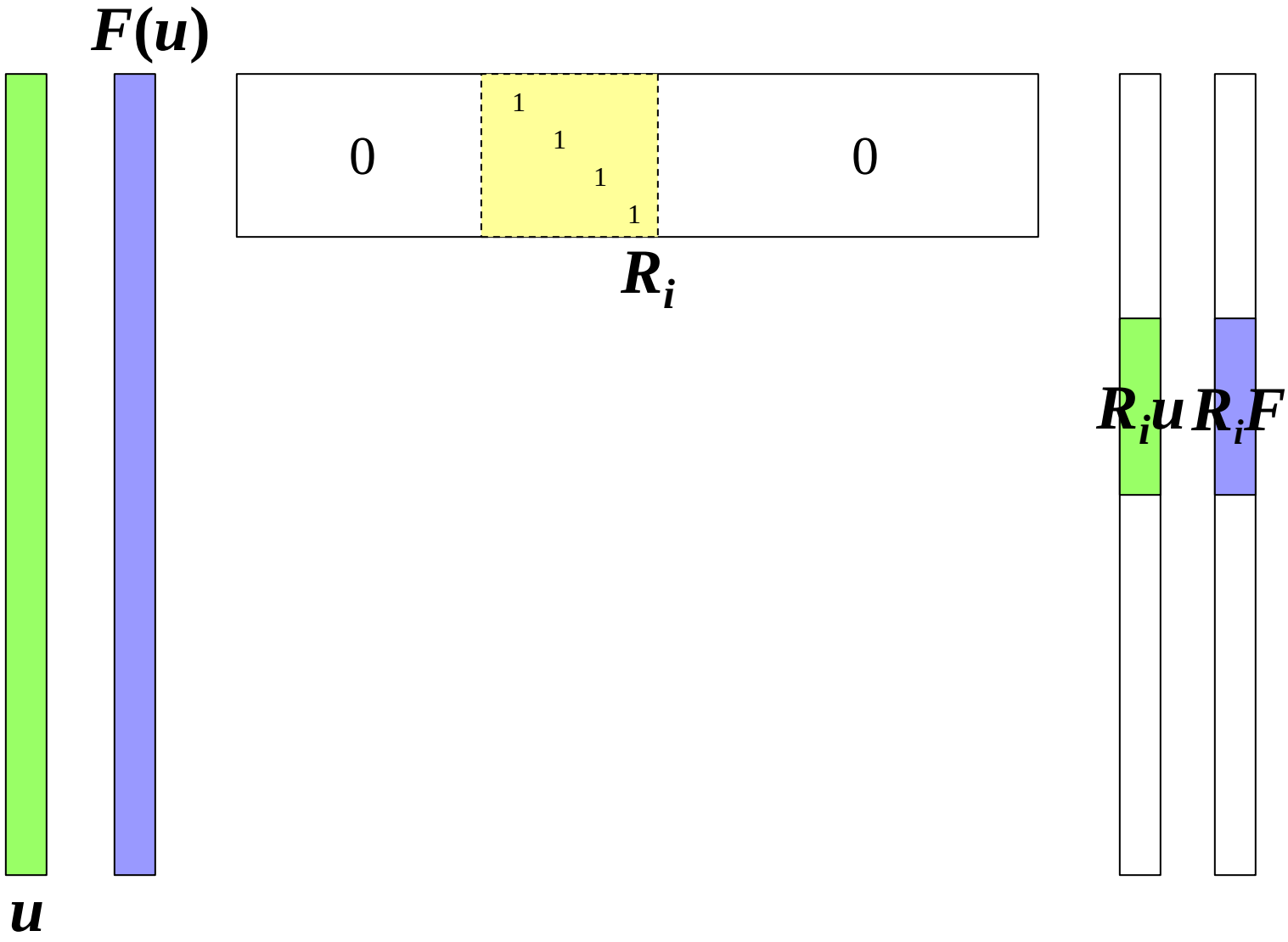
- **Nonlinear Schwarz has Newton both *inside* and *outside* and is fundamentally Jacobian-free**
- **It replaces $F(u)=0$ with a new nonlinear system possessing the same root, $\Phi(u)=0$**
- **Define a correction $d_i(u)$ to the i^{th} partition (e.g., subdomain) of the solution vector by solving the following local nonlinear system:**

$$R_i F(u+d_i(u))=0$$

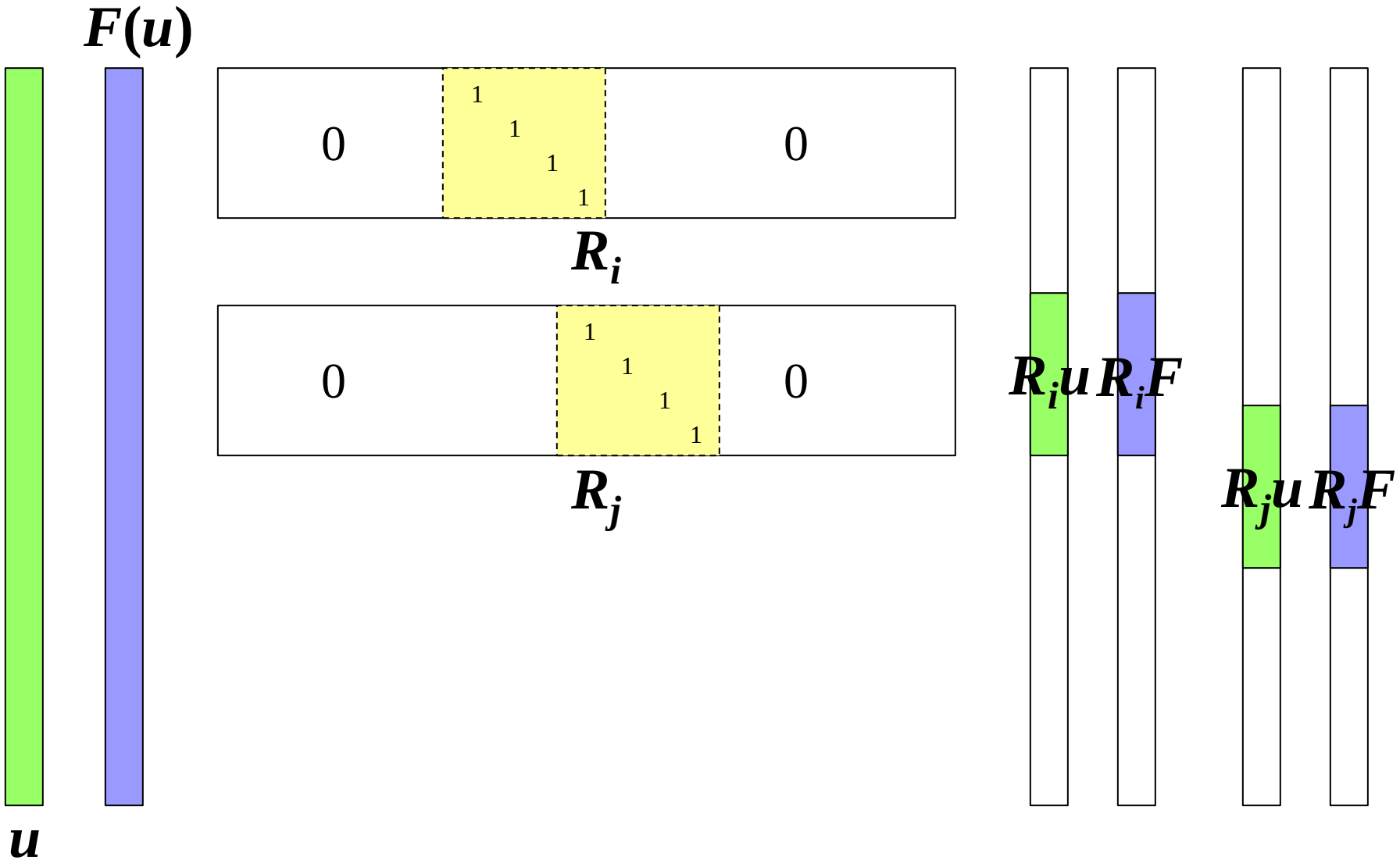
where $d_i(u)\in\Re^n$ is nonzero only in the components of the i^{th} partition

- **Then sum the corrections: $\Phi(u)=\sum_i d_i(u)$ to get an implicit function of u**

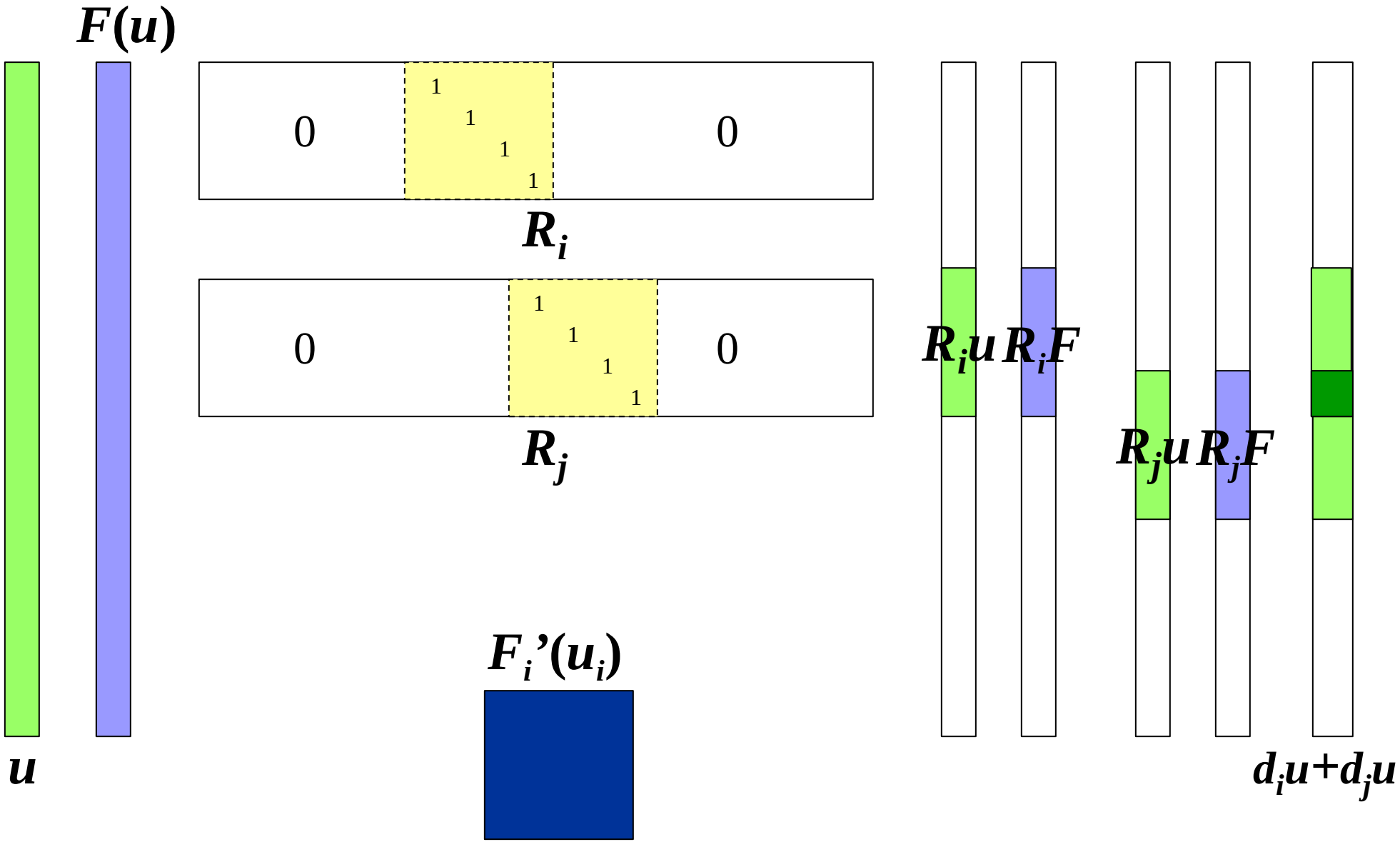
Nonlinear Schwarz – picture



Nonlinear Schwarz – picture



Nonlinear Schwarz – picture



Nonlinear Schwarz, cont.

- It is simple to prove that if the Jacobian of $F(u)$ is nonsingular in a neighborhood of the desired root then $\Phi(u)=0$ and $F(u)=0$ have the same unique root
- To lead to a Jacobian-free Newton-Krylov algorithm we need to be able to evaluate for any $u,v\in\Re^n$:
  - The residual $\Phi(u)=\sum_i d_i(u)$
  - The Jacobian-vector product $\Phi(u)'v$
- Remarkably, (Cai-Keyes, 2000) it can be shown that

$$\Phi'(u)v \approx \sum_i (R_i^T J_i^{-1} R_i) Jv$$

where $J=F'(u)$ and $J_i = R_i J R_i^T$

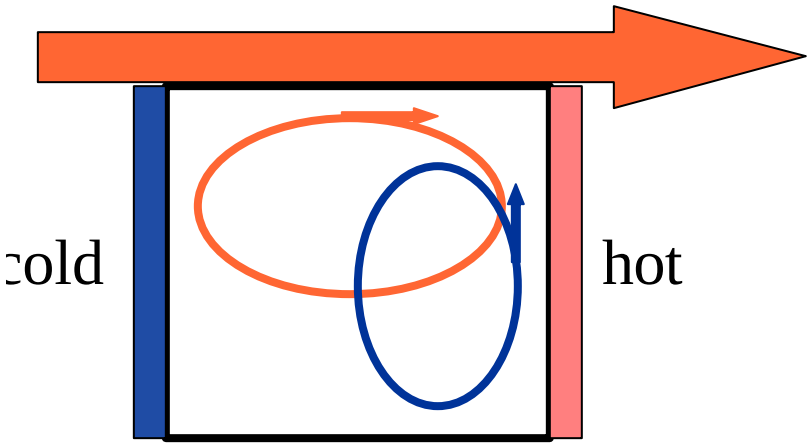
- All required actions are available in terms of $F(u)$!

Nonlinear Schwarz, cont.

Discussion:

- **After the linear version of additive Schwarz was invented, it was realized that it is in the same class of methods as additive multigrid, in terms of operator algebra: projection off the error in a set of subspaces (though linear MG is usually done multiplicatively)**
- **After nonlinear Schwarz was invented, it was realized that it is in the same class of methods as nonlinear multigrid (full approximation scheme MG), in terms of operator algebra (though nonlinear multigrid is usually done multiplicatively)**

Driven cavity in velocity-vorticity coords



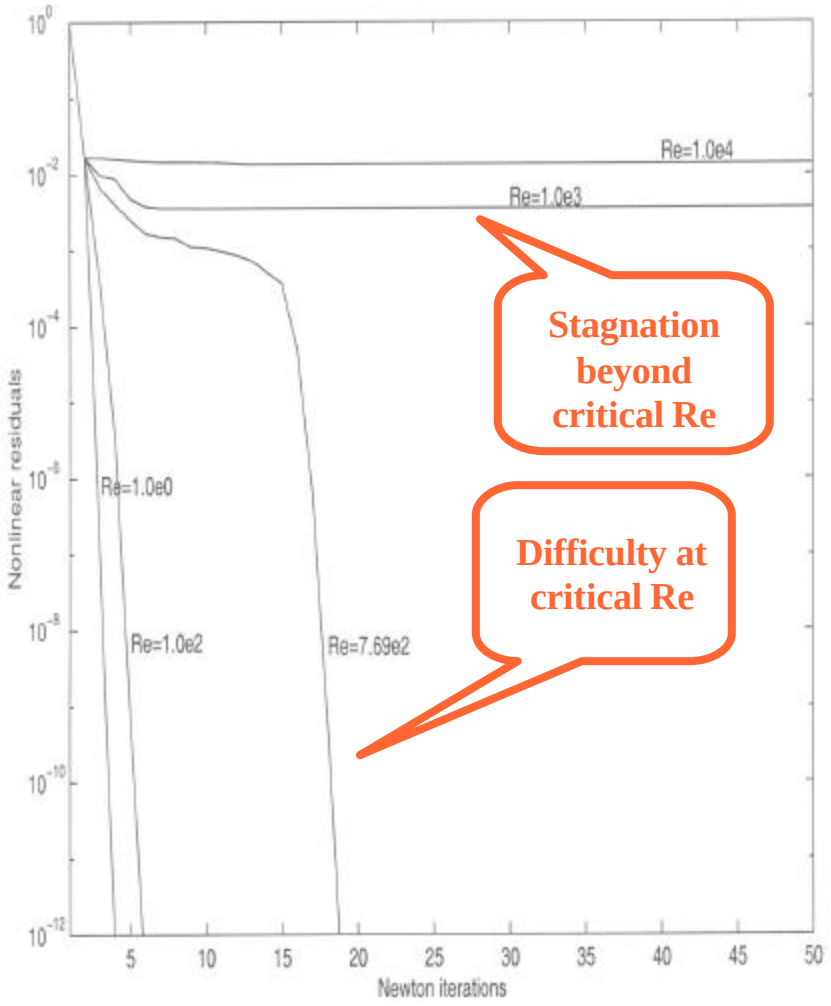
x-velocity $$-\nabla^2 u - \frac{\partial w}{\partial y} = 0$$

y-velocity $$-\nabla^2 v + \frac{\partial w}{\partial x} = 0$$

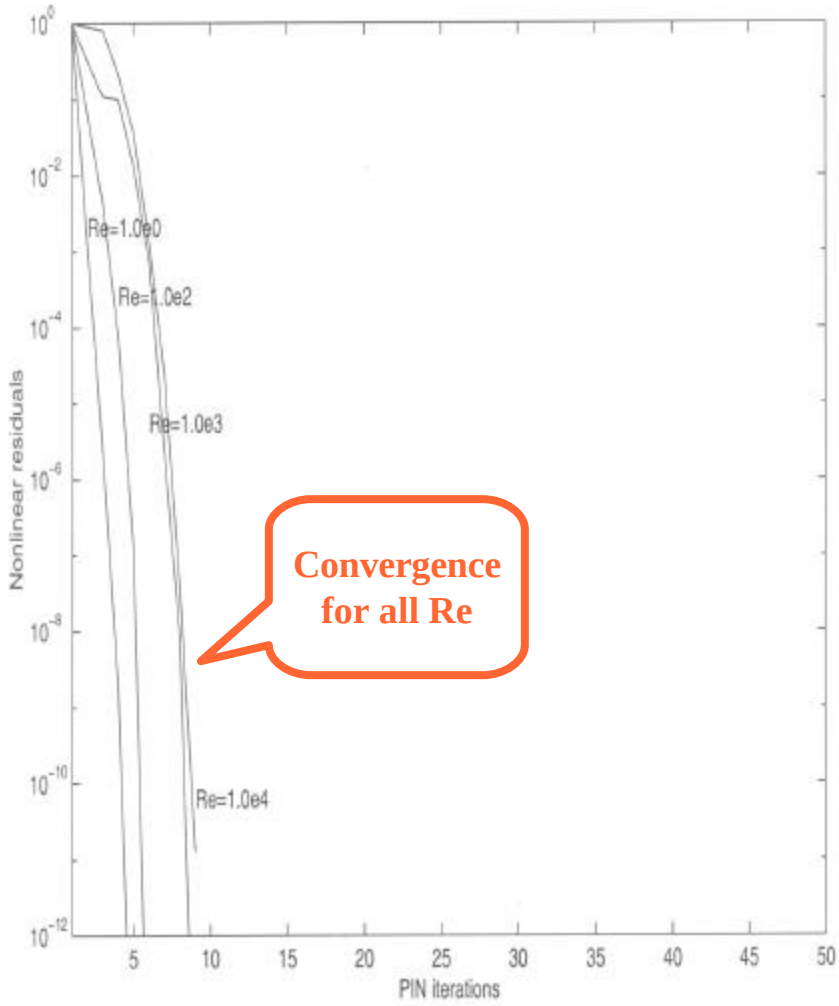
vorticity $$-\nabla^2 w + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} - \mathrm{Gr}\frac{\partial T}{\partial x} = 0$$

internal energy $$-\nabla^2 T + \mathrm{Pr}(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}) = 0$$

Experimental example of nonlinear Schwarz



Newton's method

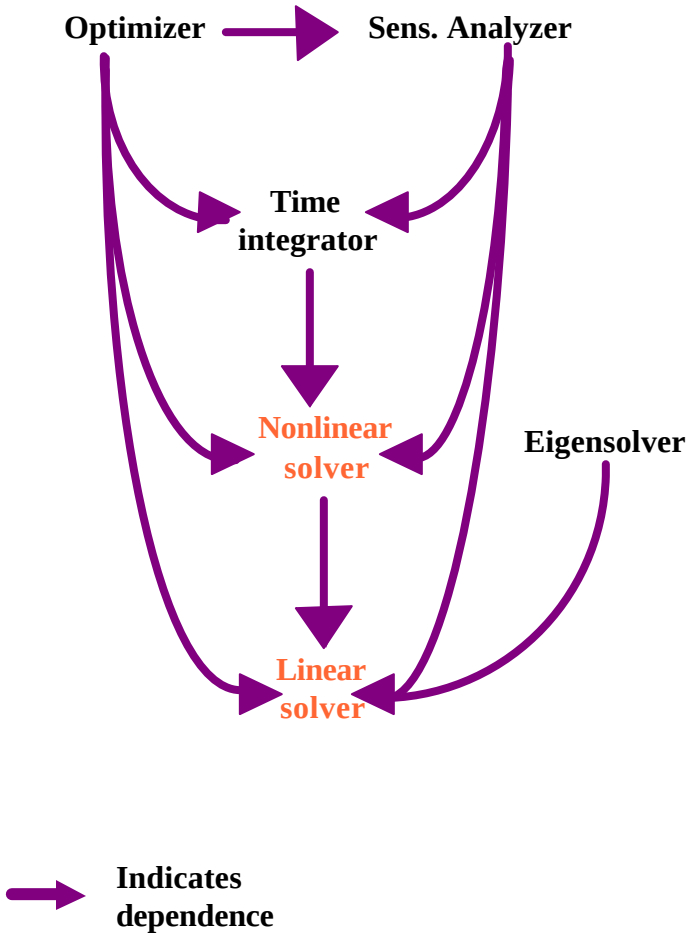


Additive Schwarz Preconditioned Inexact Newton (ASPIN)

Common software infrastructure for nonlinear PDE solvers – coming in PETSc

Vision:

- **The user codes to the problem they are solving (by providing $F(u)$), *not* the algorithm used to solve the problem**
- **Implementation of various algorithms reuse common concepts and code when possible, *without* losing efficiency**



Encompassing …

- **Newton's method**
  - **Jacobian-free methods**
  - **w/Jacobian-based preconditioned linear solvers (Newton-PCG)**
  - **w/multigrid linear solvers (Newton-MG)**
- **Nonlinear multigrid**
  - **a.k.a. Full approximation scheme (FAS)**
  - **a.k.a. MG-Newton**

Software engineering ingredients

- Standard solver interfaces (SLES, SNES)
- Solver libraries
- Automatic differentiation (AD) tools
- Code generation tools

Algorithm review

$$F(u) = 0, \quad \textbf{Jacobian } A(u)$$

Newton

$$u \leftarrow \bar{u} - A^{-1}(\bar{u})F(\bar{u})$$

Newton–SOR (1 inner linear sweep over i)

$$u_i \leftarrow \bar{u}_i - A_{ii}^{-1}(\bar{u})\{F_i(\bar{u}) - \sum_{j<i} A_{ij}(\bar{u})[\bar{u}_j - u_j]\}$$

SOR–Newton (1 inner nonlinear sweep over i)

$$u_i \leftarrow \bar{u}_i - A_{ii}^{-1}(u)F_i(u)$$

Cute observation

SOR-Newton

$$u_i \leftarrow \bar{u}_i - A_{ii}^{-1}(u) F_i(u)$$

… with approximations

$$A_{ii}(u) \approx A_{ii}(\bar{u})$$

$$F_i(u) \approx F_i(\bar{u}) + \sum A_{ij}(\bar{u})[u_j - \bar{u}_j]$$

… gives Newton-SOR

$$u_i \leftarrow \bar{u}_i - A_{ii}^{-1}(\bar{u})\{F_i(\bar{u}) - \sum_{j<i} A_{ij}(\bar{u})[\bar{u}_j - u_j]\}$$

Þ **(Gauss-Seidel) matrix-free linear relaxation**

is very similar to nonlinear relaxation

Function and Jacobian evaluation

- **FAS requires them pointwise**
- **Newton requires them globally**
- **Newton-MG requires both**
  - **Newton (outside) globally**
  - **MG (inside) locally**

Enter Automatic Differentiation

- Given code for $F(u)$ AD can compute
  - $A(u)$ and
  - $A(u)*w$ efficiently
- Given code for $F_i(u)$ AD can compute
  - $A_{ii}(u)$ and
  - $\sum_j A_{ij}(u)w_j$ efficiently

Automated Code generation ("in-lining")

- **Newton method requires a user-provided function and an AD Jacobian**
- **Big performance hit if handled directly with components**
  - **A *outer loop* over an *inner subroutine* that calls a function at each point is inefficient globally**
  - **An *outer subroutine* that *loops over all inner points* is wasteful locally (information thrown away)**
- **Need to have it *both ways*, depending upon algorithmic context; user should not see this level of performance-induced coding complexity**

Coarse grid correction can also be handled

Coarse grid objects generated together with fine (e.g., PETSc's DMMG)

- **Newton-MG**

$$A_H(R\bar{u})c_H = RF(\bar{u})$$

$$u \leftarrow u - R^T c_H$$

- **MG-Newton**

$$F_H(R\bar{u} + c_H) - F_H(R\bar{u}) + RF(\bar{u}) = 0$$

Conclusion

The algorithmic/mathematical building blocks for Newton-MG and MG-Newton are essentially the same.

Thus the software building blocks should be also (and they will be in the next release of PETSc, version 3.0).