Fast Solvers for Incompressible Flow Problems IV

David Silvester
University of Manchester

Aside — parabolic smoothing

- Philip Gresho & David Griffiths & David Silvester Adaptive time-stepping for incompressible flow; part I: scalar advection-diffusion, SIAM J. Scientific Computing, 30: 2018–2054, 2008.

Heat Equation – I

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0, \qquad 0 < x < 1$$

$$u(0,t) = 1, \quad u(1,t) = 0 \qquad\qquad BC$$

$$u(x,0) = 1,\ 0 \le x < 1,\ u(1,0) = 0 \qquad IC$$

Solution.

$$u(x,t) = \begin{cases} \text{erf}\left(\frac{1-x}{\sqrt{4t}}\right) \\ (1-x) + \sum_{j=1}^{\infty} \frac{2}{j\pi} e^{-j^2\pi^2 t} \sin j\pi x \end{cases}$$

Heat Equation – II

Spatial Discretization

Using linear FEM gives the ODE system

$$M\dot{\mathbf{u}} + A\mathbf{u} = \mathbf{f}$$

with M and A both symmetric positive definite matrices.

Discrete solution.

$$\mathbf{u}(t) = (1 - x) \, + \sum_{k=1}^{n_u} a_k e^{-\lambda_k t} \mathbf{v}_k$$

where $0 < \lambda_1 \le \lambda_2 \le \ldots \le \lambda_{n_u}$ and $\{\lambda_k, \mathbf{v}_k\}$ satisfy

$$M\mathbf{v}_k = \lambda_k A\mathbf{v}_k.$$

Heat Equation – III

$$\mathbf{u}(t) = (1-x) + \sum_{k=1}^{n_u} a_k e^{-\lambda_k t}\mathbf{v}_k$$

... suggests two asymptotic extremes ...

- For $t < \frac{1}{\lambda_{n_u}} =: \tau_{\mathtt{mtb}}$ there is a fast transient:
  $\mathbf{u}(t) \sim a_{n_u} e^{-\lambda_{n_u} t}\mathbf{v}_{n_u} +$ slowly varying terms
- For $t \gg 1$ there is a slow transient:
  $\mathbf{u}(t) \sim (1-x) + a_1 e^{-\lambda_1 t}\mathbf{v}_1$

Heat Equation – III

$$\mathbf{u}(t) = (1 - x) \; + \sum_{k=1}^{n_u} a_k e^{-\lambda_k t} \mathbf{v}_k$$

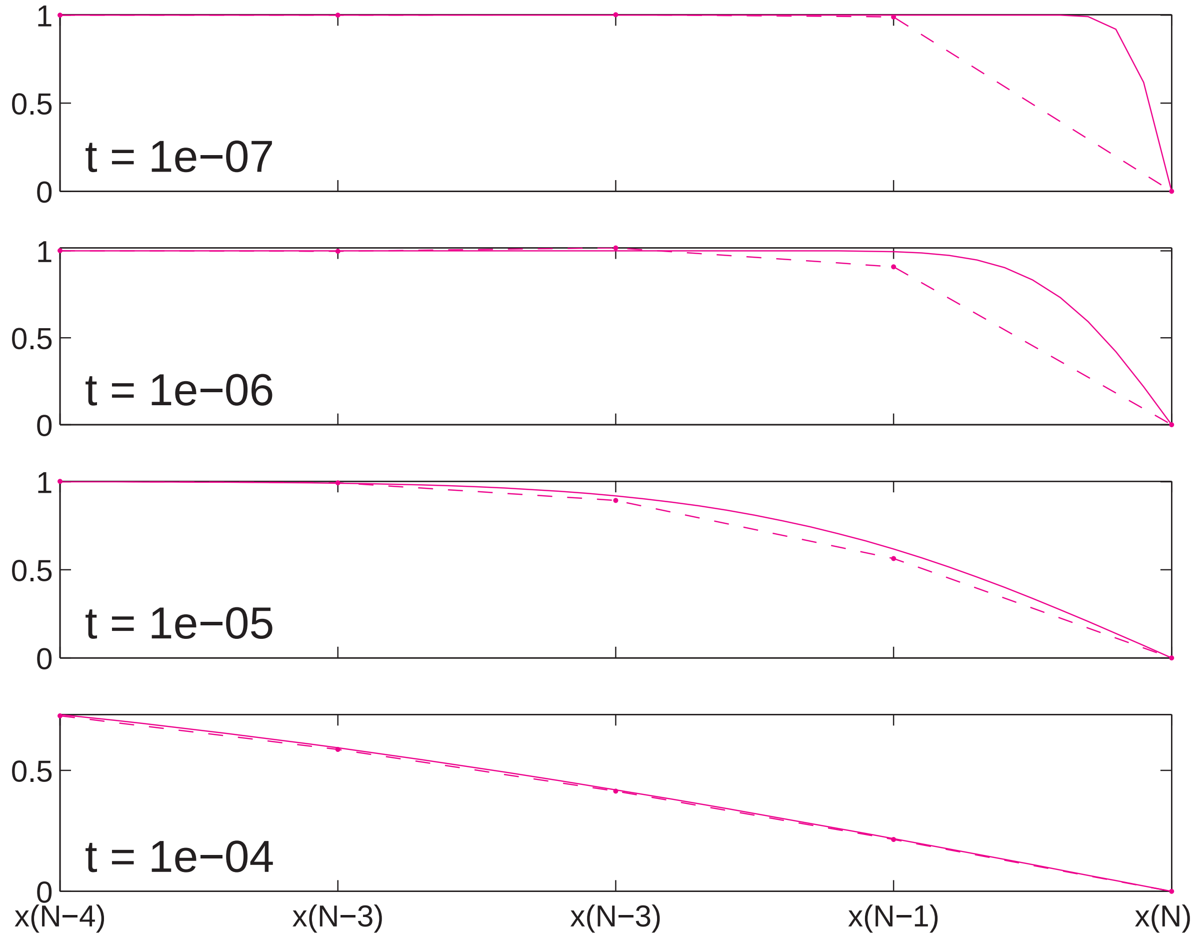
... suggests two asymptotic extremes ...

- For $t < \frac{1}{\lambda_{n_u}} =: \tau_{\mathtt{mtb}}$ there is a fast transient:
  $\mathbf{u}(t) \sim a_{n_u} e^{-\lambda_{n_u} t} \mathbf{v}_{n_u} +$ slowly varying terms
- For $t \gg 1$ there is a slow transient:
  $\mathbf{u}(t) \sim (1 - x) \; + a_1 e^{-\lambda_1 t} \mathbf{v}_1$

$\tau_{\mathtt{mtb}} \approx \frac{h^2}{4}$ is the “Minimum Time of Believability” for spatially discretized convection-diffusion problems—it is the time for discontinuities in IC to grow to size h.

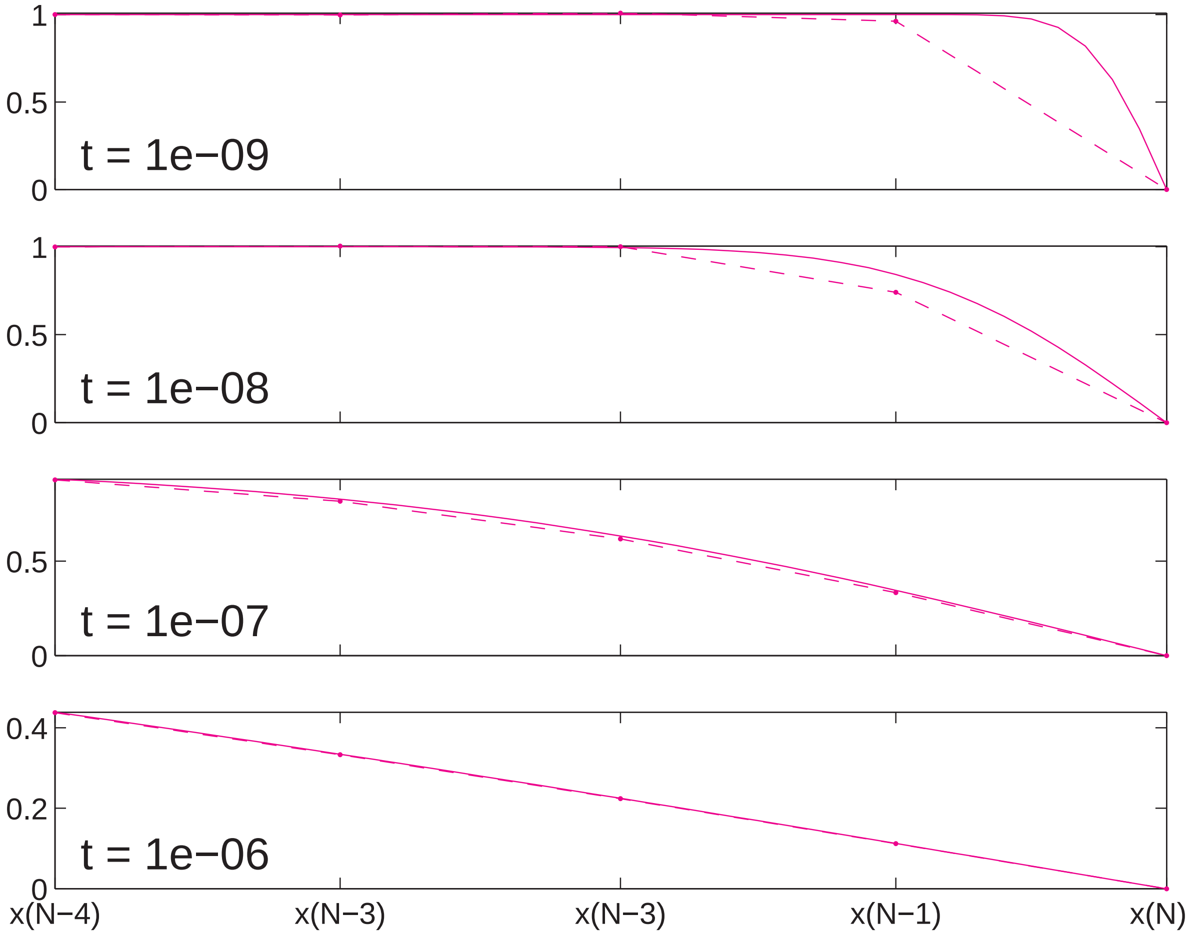
Spatial discretization I

- Uniform: $n_u = 255$, $h = 1/256$, $\tau_{\mathtt{mtb}} \sim 4 \times 10^{-6}$



Spatial discretization II

- Geometric: $h_{\min} = 2 \times 10^{-4}, n_u = 255, \tau_{\mathtt{mtb}} \sim 10^{-8}$



Heat Equation – IV

$$\mathbf{u}(t) = (1-x) \ + \sum_{k=1}^{n_u} a_k e^{-\lambda_k t}\mathbf{v}_k; \qquad \Delta t_n^3 = \frac{12\mathtt{tol}}{\|\dddot{\mathbf{u}}\|}$$

- For $t < \tau_{\mathtt{mtb}}$ there is a fast transient:
  $\mathbf{u}(t) \sim a_{n_u} e^{-\lambda_{n_u} t}\mathbf{v}_{n_u} +$ slowly varying terms
  $\Delta t_n \sim e^{\lambda_{n_u} t/3}$

- For $t \gg 1$ there is a slow transient:
  $\mathbf{u}(t) \sim (1-x) \ + a_1 e^{-\lambda_1 t}\mathbf{v}_1$
  $\Delta t_n \sim e^{\lambda_1 t/3}$

Heat Equation – IV

$$\mathbf{u}(t) = (1-x) \ + \sum_{k=1}^{n_u} a_k e^{-\lambda_k t} \mathbf{v}_k; \qquad \Delta t_n^3 = \frac{12\texttt{tol}}{\|\dddot{\mathbf{u}}\|}$$

- For $t < \tau_{\texttt{mtb}}$ there is a fast transient:
  $\mathbf{u}(t) \sim a_{n_u} e^{-\lambda_{n_u} t} \mathbf{v}_{n_u} +$ slowly varying terms
  $\Delta t_n \sim e^{\lambda_{n_u} t/3}$
- What happens in between?
- For $t \gg 1$ there is a slow transient:
  $\mathbf{u}(t) \sim (1-x) \ + a_1 e^{-\lambda_1 t} \mathbf{v}_1$
  $\Delta t_n \sim e^{\lambda_1 t/3}$

Heat Equation – V

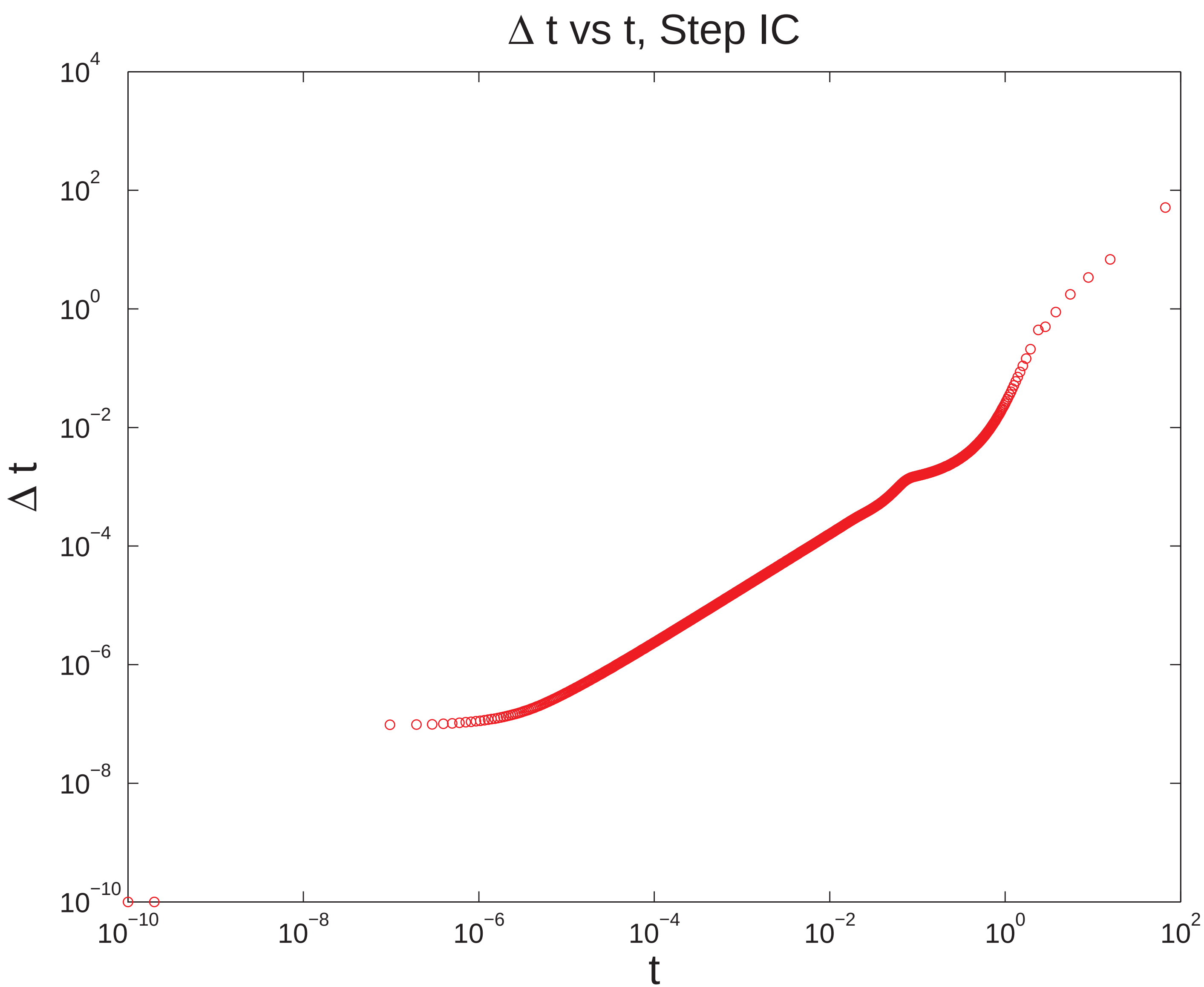
$$u(t) = (1-x) \; + \sum_{j=1}^{\infty} a_j e^{-j^2\pi^2 t} \sin j\pi x$$

Parabolic smoothing (Luskin & Rannacher)

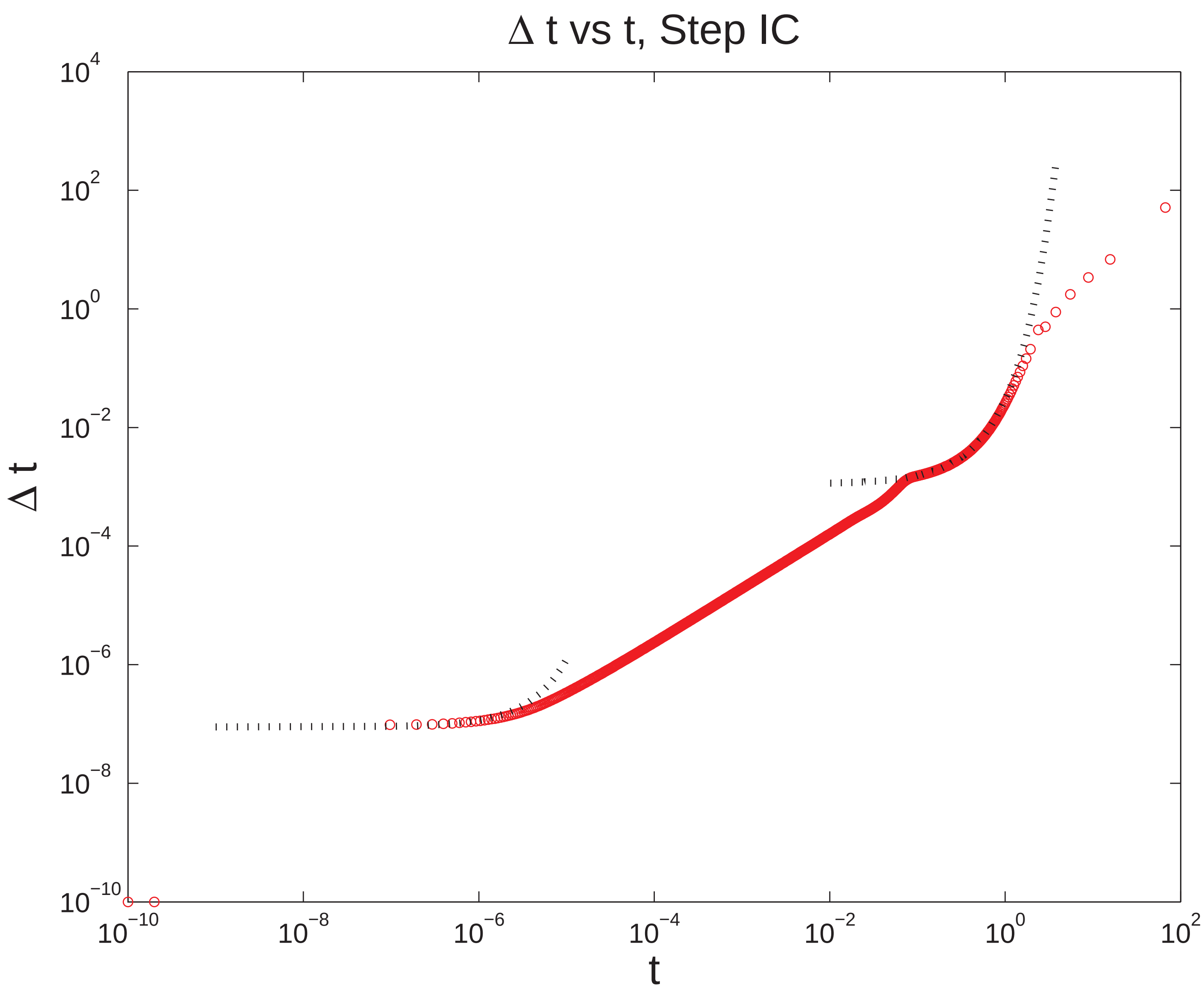
$$\begin{aligned}
\|\ddot{\mathbf{u}}\|^2 &\le C\|\dddot{u}\|^2 \\
&= C \sum_{j=1}^{\infty} j^6 a_j^2 e^{-2j^2\pi^2 t} \\
&\le C \max_j \left(j^{7+\epsilon} a_j^2 e^{-2j^2\pi^2 t}\right) \sum_{j=1}^{\infty} \frac{1}{j^{1+\epsilon}} \le \frac{C}{t^{11/2}}
\end{aligned}$$

This gives the lower bound: $\Delta t_n \ge C t^{11/12}$

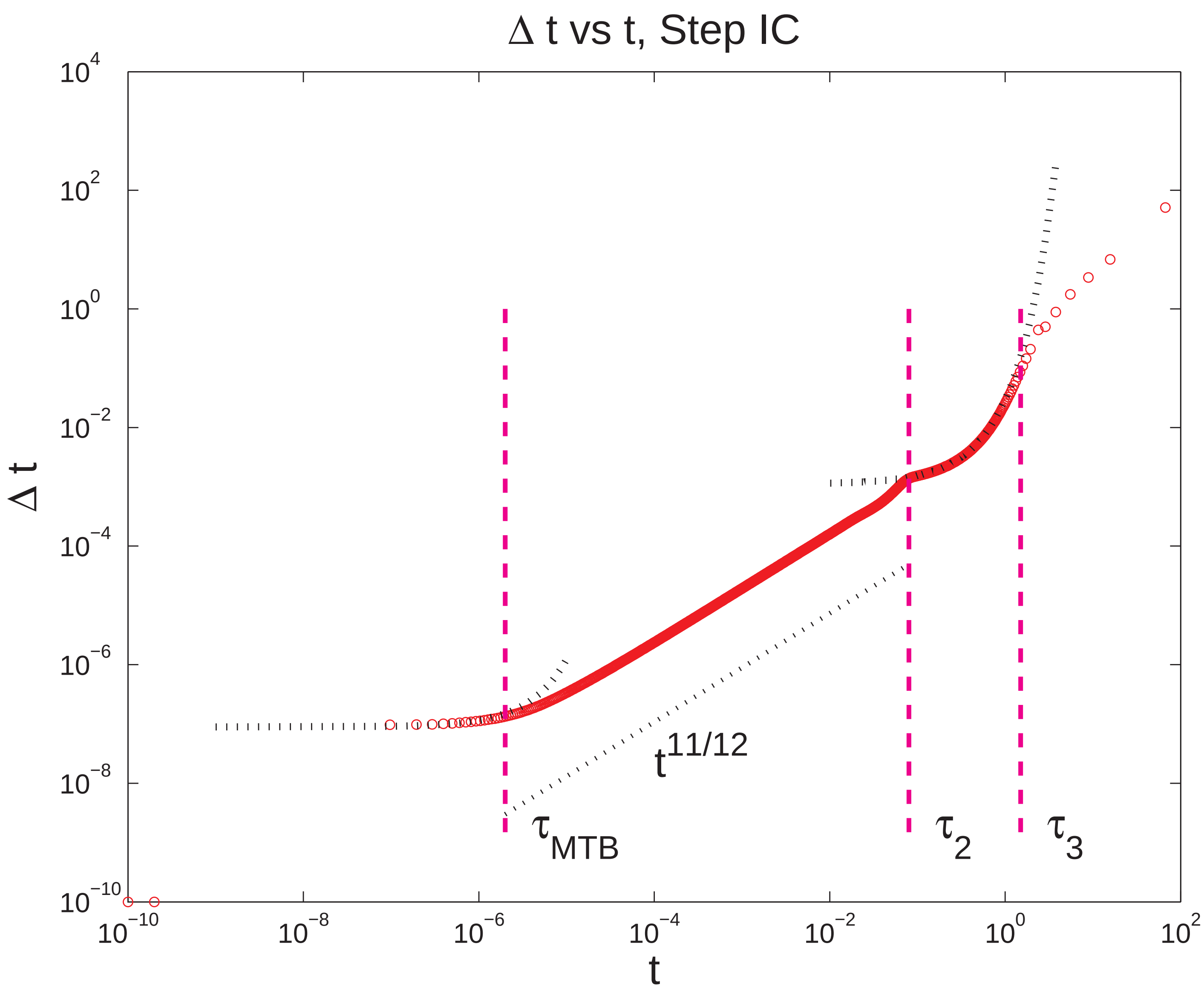
Uniform grid – Time steps



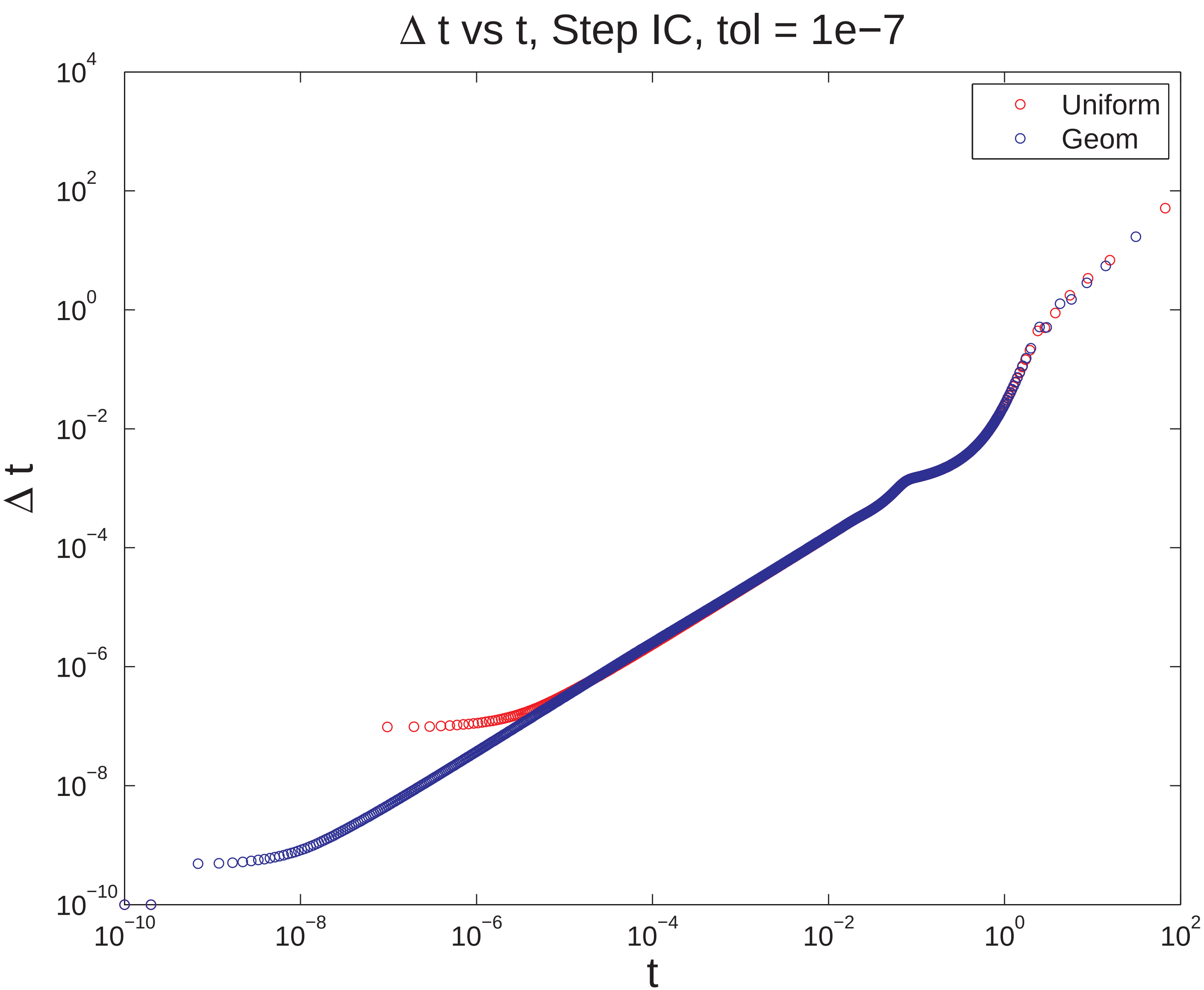
Uniform grid – Time steps



Uniform grid – Time steps



Uniform vs Geometric grid



Lecture IV

- $$-\nabla^2 \vec{u} + \nabla p = \vec{0}; \quad \nabla \cdot \vec{u} = 0$$

- $$\vec{u} \cdot \nabla \vec{u} - \nu \nabla^2 \vec{u} + \nabla p = \vec{0}; \quad \nabla \cdot \vec{u} = 0$$

- $$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} - \nu \nabla^2 \vec{u} + \nabla p = 0; \quad \nabla \cdot \vec{u} = 0$$

- $$\left.\begin{array}{c} \dfrac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} - \nu \nabla^2 \vec{u} + \nabla p = \vec{j} T; \quad \nabla \cdot \vec{u} = 0 \\ \dfrac{\partial T}{\partial t} + \vec{u} \cdot \nabla T - \nu \nabla^2 T = 0 \end{array}\right\}$$

Reference

- Howard Elman, Milan Mihajlović and David Silvester.
  Fast iterative solvers for buoyancy driven flow problems
  J. Computational Physics, 230: 3900–3914, 2011.

Buoyancy driven flow

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} - \nu \nabla^2 \vec{u} + \nabla p = \vec{j} T \quad \text{in } \mathcal{W} \equiv \Omega \times (0, T)$$

$$\nabla \cdot \vec{u} = 0 \qquad \text{in } \mathcal{W}$$

$$\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T - \nu \nabla^2 T = 0 \qquad \text{in } \mathcal{W}$$

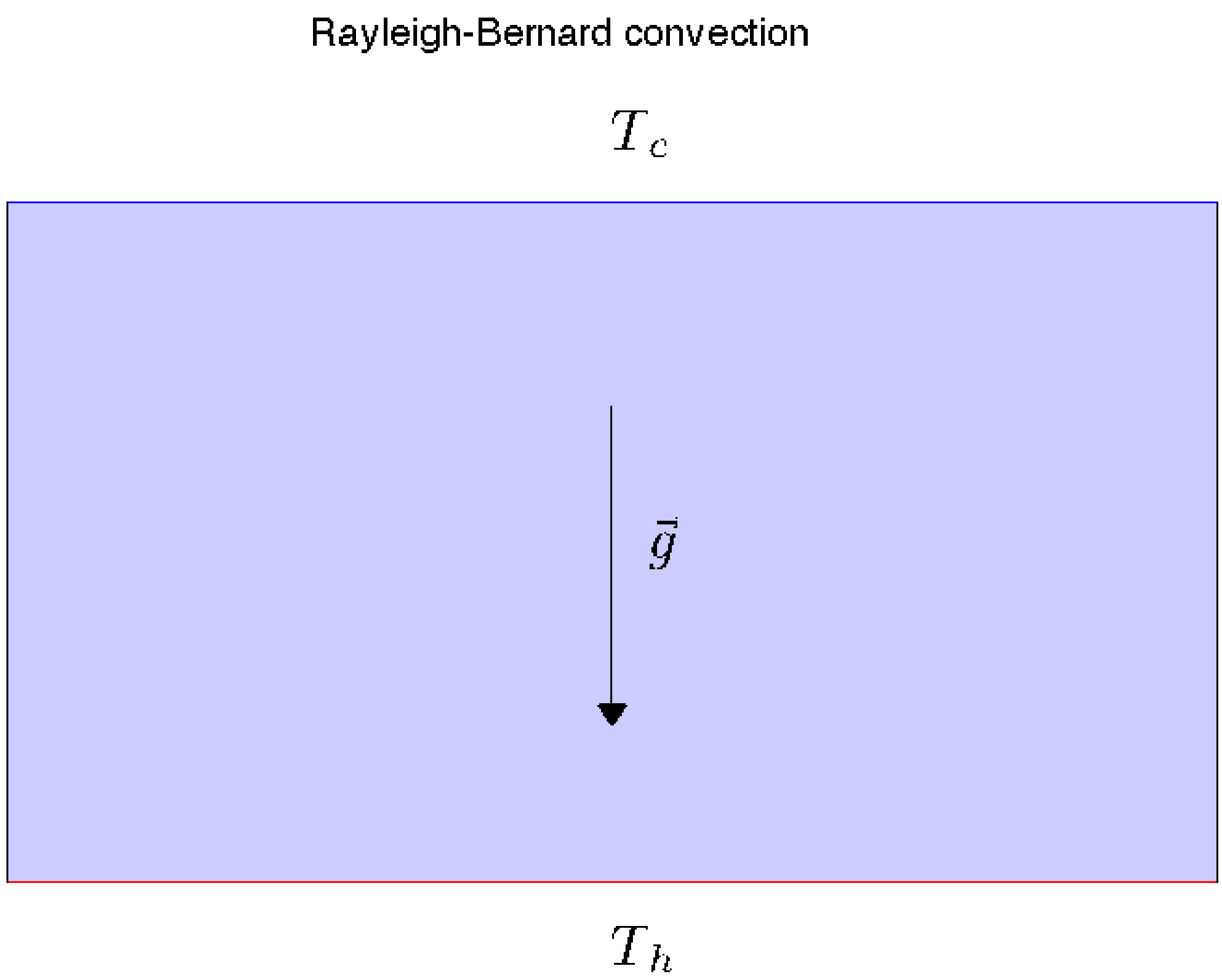
Boundary and Initial conditions

$$\vec{u} = \vec{0} \quad \text{on } \Gamma \times [0, T]; \qquad \vec{u}(\vec{x}, 0) = \vec{0} \quad \text{in } \Omega.$$

$$T = T_g \quad \text{on } \Gamma_D \times [0, T]; \qquad \nu \nabla T \cdot \vec{n} = 0 \quad \text{on } \Gamma_N \times [0, T];$$

$$T(\vec{x}, 0) = T_0(\vec{x}) \quad \text{in } \Omega.$$

avi



“Smart Integrator” (SI)

- Optimal time-stepping: time-steps automatically chosen to “follow the physics”.
- Black-box implementation: few parameters that have to be estimated a priori.
- Algorithm efficiency: solve linear equations at every timestep.

“Smart Integrator” (SI)

- Optimal time-stepping: time-steps automatically chosen to “follow the physics”.
- Black-box implementation: few parameters that have to be estimated a priori.
- Algorithm efficiency: solve linear equations at every timestep.
- Solver efficiency: see later ...

Trapezoidal Rule (TR) time discretization

Subdivide $[0,T]$ into time levels $\{t_i\}_{i=1}^N$. Given $(\vec{u}^n, p^n, T^n)$ at time t_n, $k_{n+1} := t_{n+1} - t_n$, compute $(\vec{u}^{n+1}, p^{n+1}, T^{n+1})$ via

$$\frac{2}{k_{n+1}}\,\vec{u}^{n+1} - \nu\nabla^2\,\vec{u}^{n+1} + \vec{u}^{n+1}\cdot\nabla\,\vec{u}^{n+1} + \nabla p^{n+1} - \vec{j}T^{n+1} = \frac{2}{k_{n+1}}\,\vec{u}^n + \frac{\partial\,\vec{u}}{\partial t}^n \quad \text{in } \Omega$$

$$-\nabla\cdot\,\vec{u}^{n+1} = 0 \qquad \text{in } \Omega$$

$$\vec{u}^{n+1} = \vec{0} \qquad \text{on } \Gamma$$

$$\frac{2}{k_{n+1}}T^{n+1} - \nu\nabla^2 T^{n+1} + \vec{u}^{n+1}\cdot\nabla T^{n+1} = \frac{2}{k_{n+1}}T^n + \frac{\partial T}{\partial t}^n \qquad \text{in } \Omega$$

$$T^{n+1} = T_g^{n+1} \qquad \text{on } \Gamma_D$$

$$\nu\nabla T^{n+1}\cdot\,\vec{n} = 0 \qquad \text{on } \Gamma_N.$$

Linearization

Subdivide $[0, T]$ into time levels $\{t_i\}_{i=1}^N$. Given $(\vec{u}^n, p^n, T^n)$ at time t_n, $k_{n+1} := t_{n+1} - t_n$, compute $(\vec{u}^{n+1}, p^{n+1}, T^{n+1})$ via

$$\frac{2}{k_{n+1}}\vec{u}^{n+1} - \nu\nabla^2\vec{u}^{n+1} + \vec{w}^{n+1}\cdot\nabla\vec{u}^{n+1} + \nabla p^{n+1} - \vec{j}T^{n+1} = \frac{2}{k_{n+1}}\vec{u}^n + \frac{\partial\vec{u}}{\partial t}^n \quad \text{in } \Omega$$

$$-\nabla\cdot\vec{u}^{n+1} = 0 \quad \text{in } \Omega$$

$$\vec{u}^{n+1} = \vec{0} \quad \text{on } \Gamma.$$

$$\frac{2}{k_{n+1}}T^{n+1} - \nu\nabla^2 T^{n+1} + \vec{w}^{n+1}\cdot\nabla T^{n+1} = \frac{2}{k_{n+1}}T^n + \frac{\partial T}{\partial t}^n \quad \text{in } \Omega$$

$$T^{n+1} = T_g^{n+1} \quad \text{on } \Gamma_D$$

$$\nu\nabla T^{n+1}\cdot\vec{n} = 0 \quad \text{on } \Gamma_N,$$

with $\vec{w}^{n+1} = (1 + \frac{k_{n+1}}{k_n})\vec{u}^n - \frac{k_{n+1}}{k_n}\vec{u}^{n-1}$.

Adaptive time stepping components

- Starting from rest, $\vec{u}^0 = \vec{0}$, and given a steady-state temperature boundary condition $T(\vec{x}, t) = T_g$, we model the impulse with a time-dependent boundary condition:

$$T(\vec{x}, t) = T_g(1 - e^{-5t}) \quad \text{on } \Gamma_D \times [0, T].$$

  We also choose a very small initial timestep, typically, $k_1 = 10^{-9}$.

Adaptive time stepping components

- Starting from rest, $\vec{u}^0 = \vec{0}$, and given a steady-state temperature boundary condition $T(\vec{x}, t) = T_g$, we model the impulse with a time-dependent boundary condition:

$$T(\vec{x}, t) = T_g(1 - e^{-5t}) \quad \text{on } \Gamma_D \times [0, T].$$

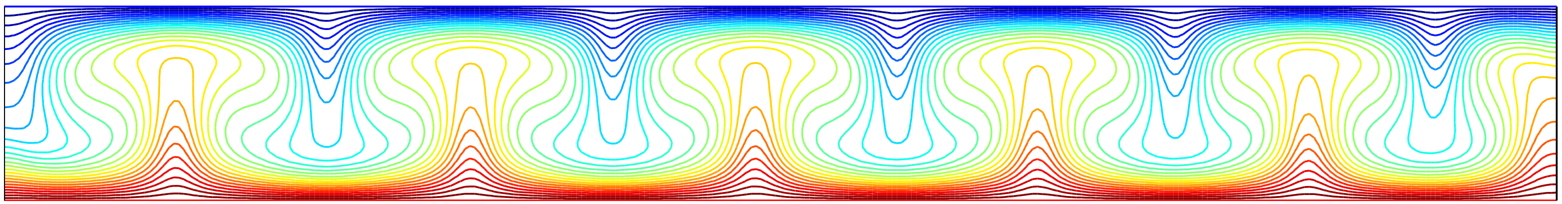
  We also choose a very small initial timestep, typically, $k_1 = 10^{-9}$.

- The following parameters must be specified:

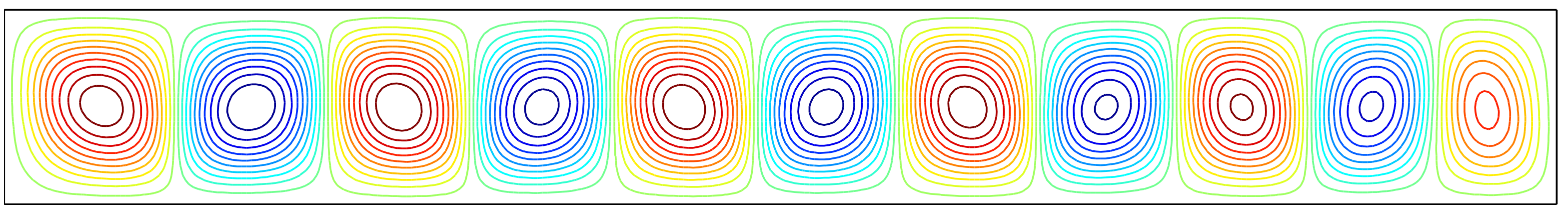
time accuracy tolerance	ε_t (10^{-5})
GMRES tolerance	`itol` (10^{-6})
GMRES iteration limit	`maxit` (50)

Problem I: Rayleigh–Bernard

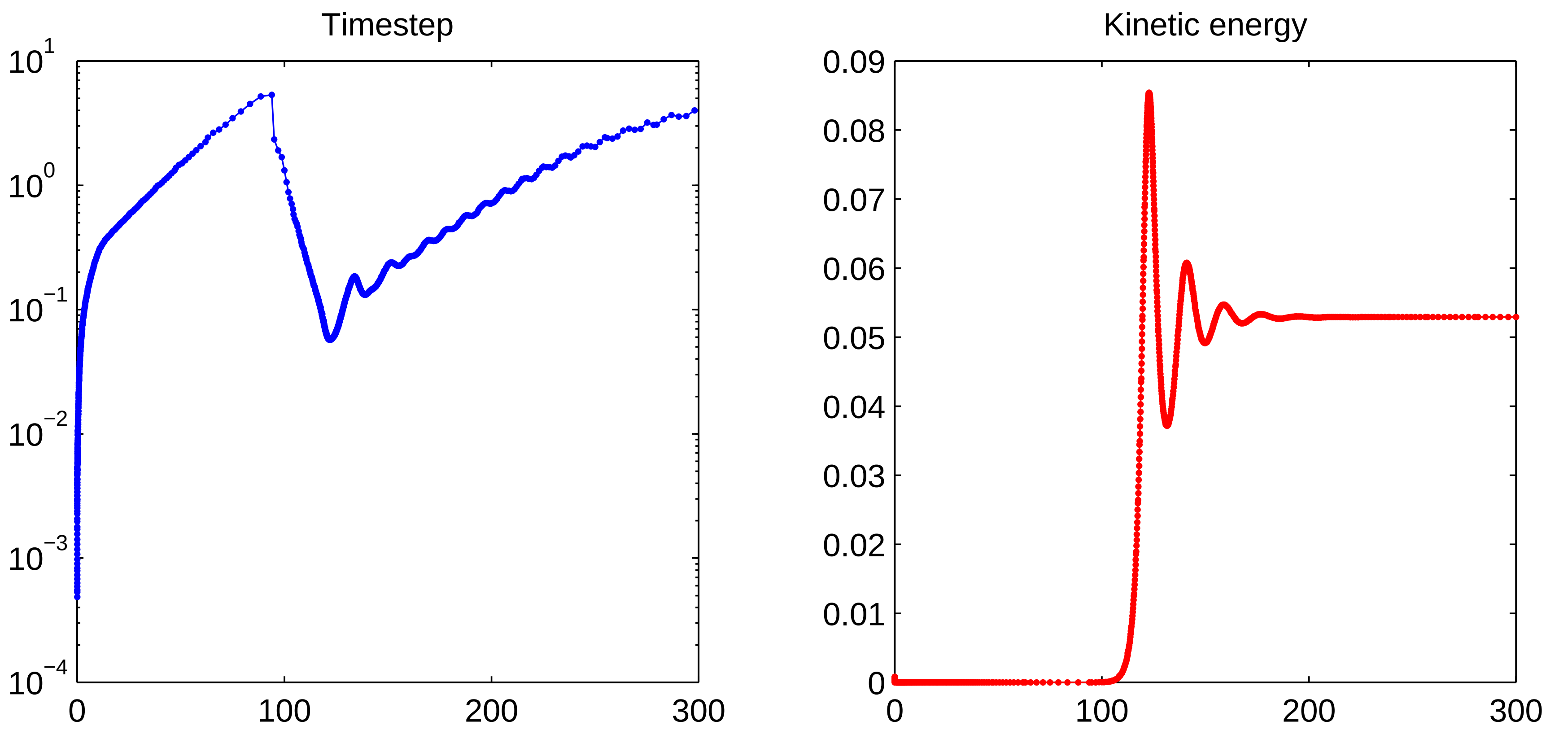
Isotherms : Ra=15000; Pr=7.1; time=300



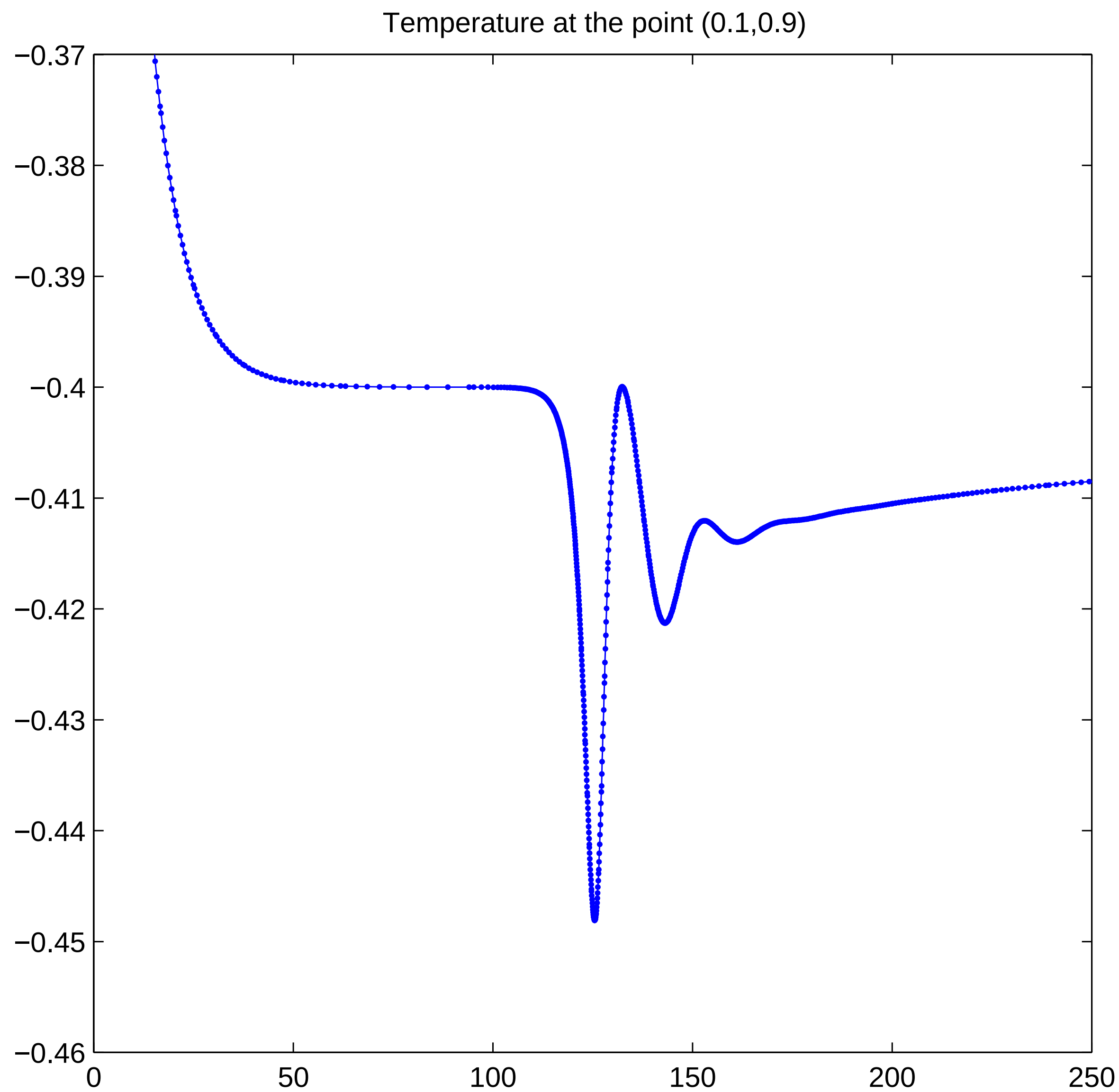
Velocity streamlines : time=300



Problem I: Timestep & Kinetic Energy : $\varepsilon_t = 10^{-6}$



Reference Point Temperature : $\varepsilon_t = 10^{-6}$



“Smart Integrator” (SI) revisited

- Optimal time-stepping
- Black-box implementation
- Algorithm efficiency

- Solver efficiency: the linear solver convergence rate is robust with respect to the mesh size h and the flow problem parameters.

Finite element matrix formulation

Introducing the basis sets

$$\mathbf{X}_h = \text{span}\{\vec{\phi}_i\}_{i=1}^{n_u}, \quad \text{Velocity basis functions;}$$

$$M_h = \text{span}\{\psi_j\}_{j=1}^{n_p}, \quad \text{Pressure basis functions.}$$

$$T_h = \text{span}\{\phi_k\}_{k=1}^{n_T}, \quad \text{Temperature basis functions;}$$

gives the method-of-lines discretized system:

$$\begin{pmatrix} M & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & M \end{pmatrix} \begin{pmatrix} \frac{\partial \vec{u}}{\partial t} \\ \frac{\partial p}{\partial t} \\ \frac{\partial T}{\partial t} \end{pmatrix} + \begin{pmatrix} F & B^T & -\overset{\circ}{\overline{M}} \\ B & 0 & 0 \\ 0 & 0 & F \end{pmatrix} \begin{pmatrix} \vec{u} \\ p \\ T \end{pmatrix} = \begin{pmatrix} \vec{0} \\ 0 \\ g \end{pmatrix}$$

with a (vertical–) mass matrix:

$$(\overset{\circ}{\overline{M}})_{ij} = ([0, \phi_i], \phi_j)$$

Preconditioning strategy

$$\begin{pmatrix} F & B^T & -\overset{\circ}{M} \\ B & 0 & 0 \\ 0 & 0 & F \end{pmatrix} \mathcal{P}^{-1} \quad \mathcal{P} \begin{pmatrix} \alpha^u \\ \alpha^p \\ \alpha^T \end{pmatrix} = \begin{pmatrix} \mathbf{f}^u \\ \mathbf{f}^p \\ \mathbf{f}^T \end{pmatrix}$$

Given $S = BF^{-1}B^T$, a perfect preconditioner is given by

$$\begin{pmatrix} F & B^T & -\overset{\circ}{M} \\ B & 0 & 0 \\ 0 & 0 & F \end{pmatrix} \underbrace{\begin{pmatrix} F^{-1} & F^{-1}B^T S^{-1} & F^{-1}\overset{\circ}{M}F^{-1} \\ 0 & -S^{-1} & 0 \\ 0 & 0 & F^{-1} \end{pmatrix}}_{\mathcal{P}^{-1}}$$

$$= \begin{pmatrix} I & 0 & 0 \\ BF^{-1} & I & BF^{-1}\overset{\circ}{M}F^{-1} \\ 0 & 0 & I \end{pmatrix}$$

For an efficient preconditioner we need to construct a sparse approximation to the “exact” Schur complement

$$S^{-1} = (BF^{-1}B^T)^{-1}$$

For an efficient implementation we must also have an efficient AMG (convection-diffusion) solver ...

HSL — HSL_MI20

PACKAGE SPECIFICATION — HSL 2007

1 SUMMARY

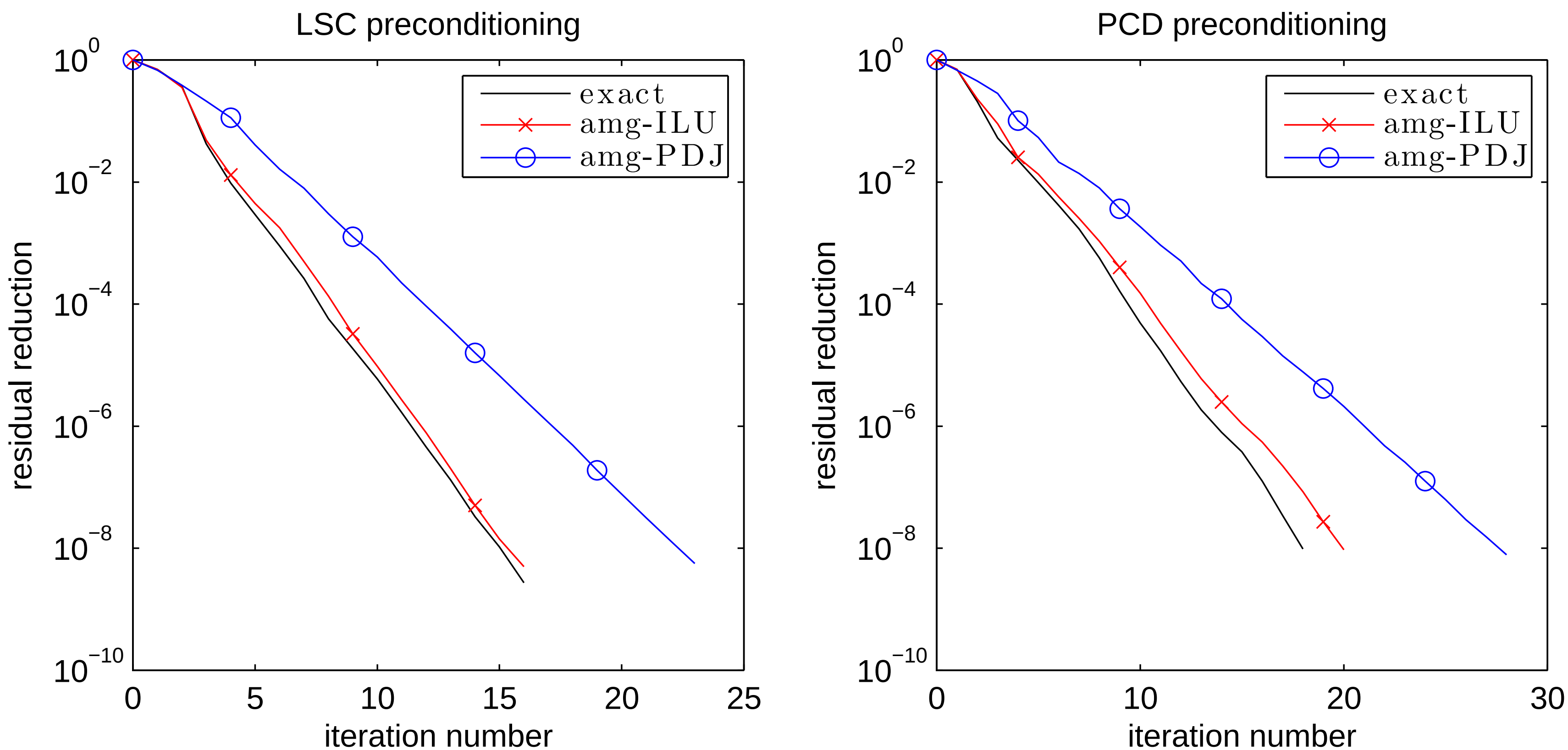
Given an $n \times n$ sparse matrix $\mathbf{A}$ and an $n-$vector $\mathbf{z}$, `HSL_MI20` computes the vector $\mathbf{x} = \mathbf{Mz}$, where $\mathbf{M}$ is an algebraic multigrid (AMG) v-cycle preconditioner for $\mathbf{A}$. A classical AMG method is used, as described in [1] (see also Section 5 below for a brief description of the algorithm). The matrix $\mathbf{A}$ must have positive diagonal entries and (most of) the off-diagonal entries must be negative (the diagonal should be large compared to the sum of the off-diagonals). During the multigrid coarsening process, positive off-diagonal entries are ignored and, when calculating the interpolation weights, positive off-diagonal entries are added to the diagonal.

Reference

[1] K. Stüben. *An Introduction to Algebraic Multigrid*. In U. Trottenberg, C. Oosterlee, A. Schüller, eds, 'Multigrid', Academic Press, 2001, pp 413-532.

ATTRIBUTES — Version: 1.1.0 **Types:** Real (single, double). **Uses:** `HSL_MA48`, `HSL_MC65`, `HSL_ZD11`, and the `LAPACK` routines `_GETRF` and `_GETRS`. **Date:** September 2006. **Origin:** J. W. Boyle, University of Manchester and J. A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable dummy arguments and allocatable components of derived types. **Remark:** The development of `HSL_MI20` was funded by EPSRC grants EP/C000528/1 and GR/S42170.

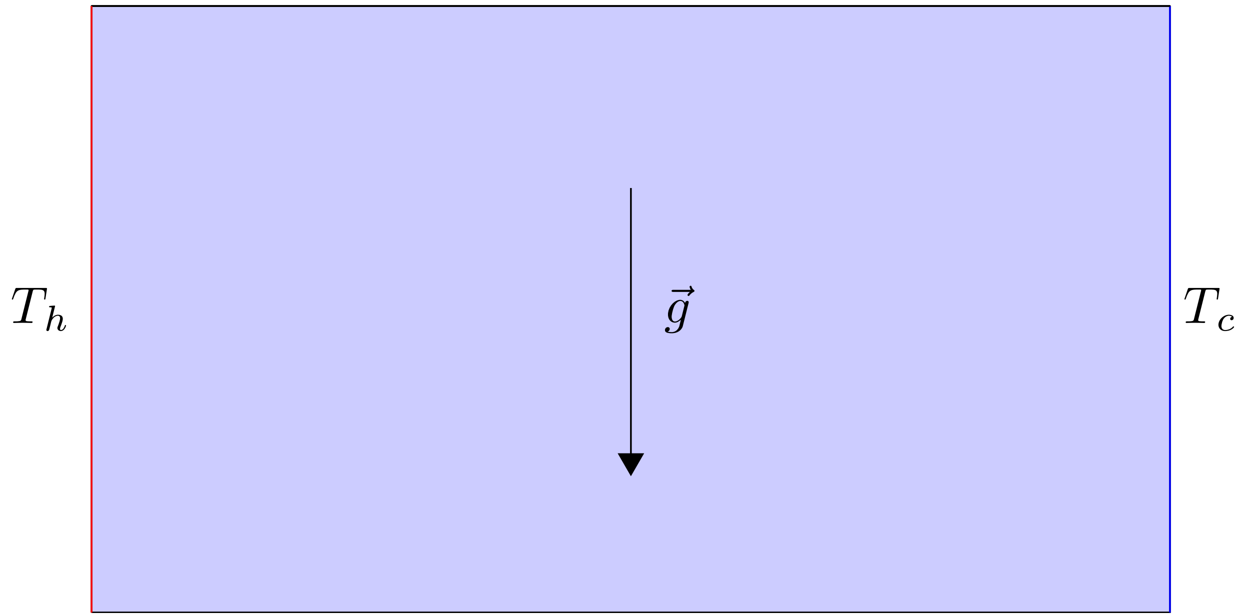
Solver performance



GMRES convergence close to steady state with $k_n \sim 4$.
Note that $\nu = 0.0218$ and $\nu = 0.00306$.

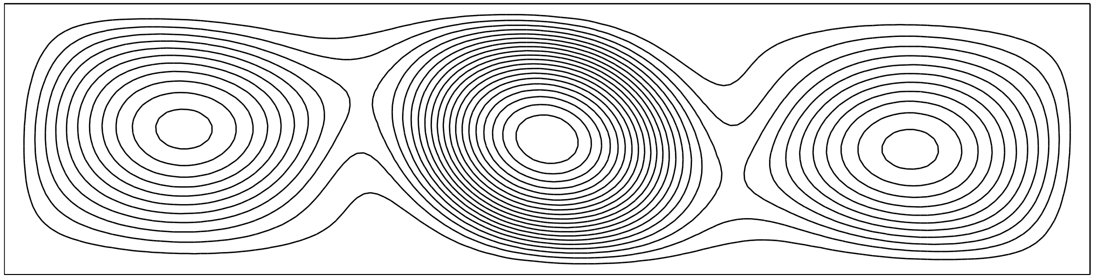
Problem II: 1:4 cavity domain

Lateral heating: Hopf Bifurcation

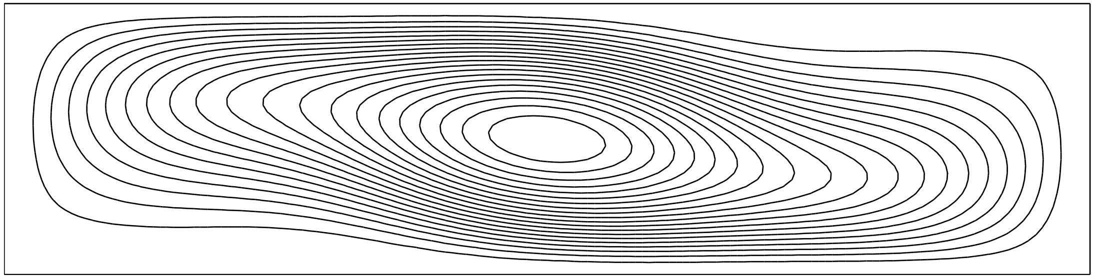


Problem II: Gallium Arsenide

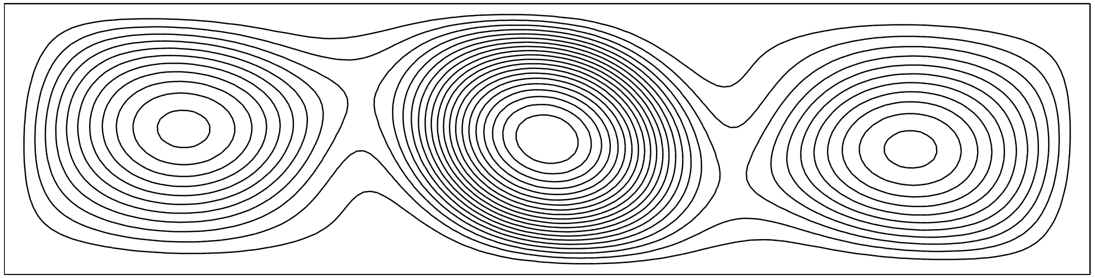
Velocity streamlines : time= 223.90



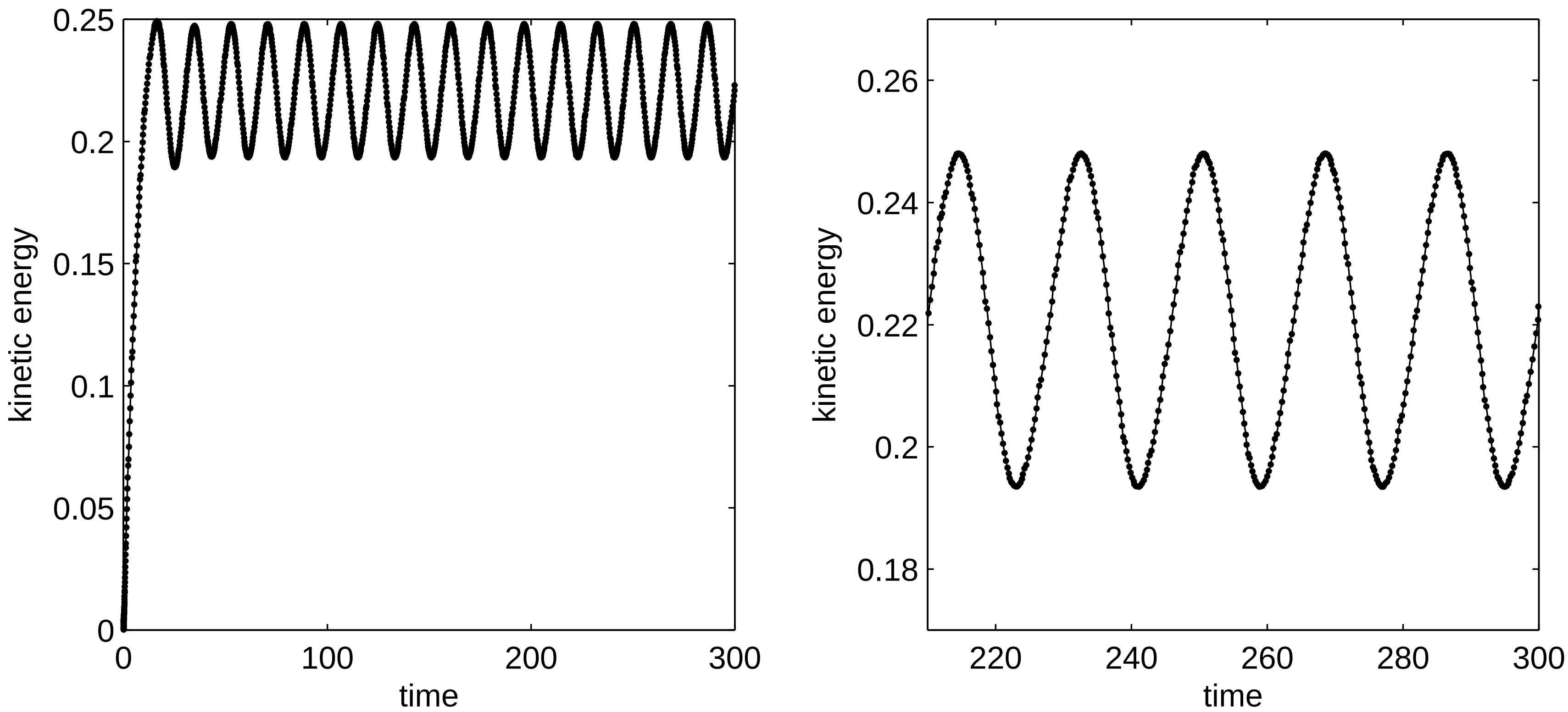
Velocity streamlines : time= 232.95



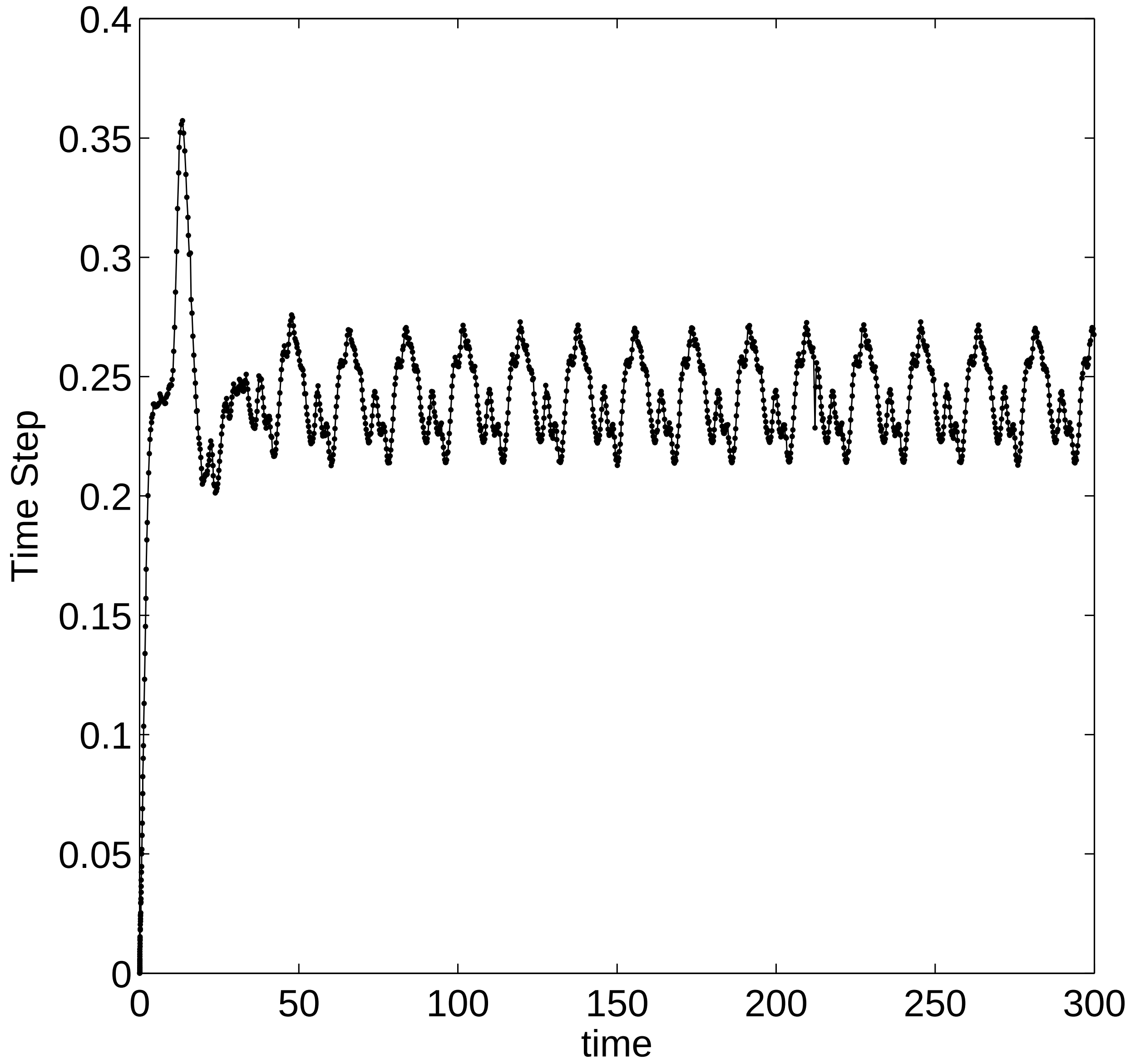
Velocity streamlines : time= 241.84



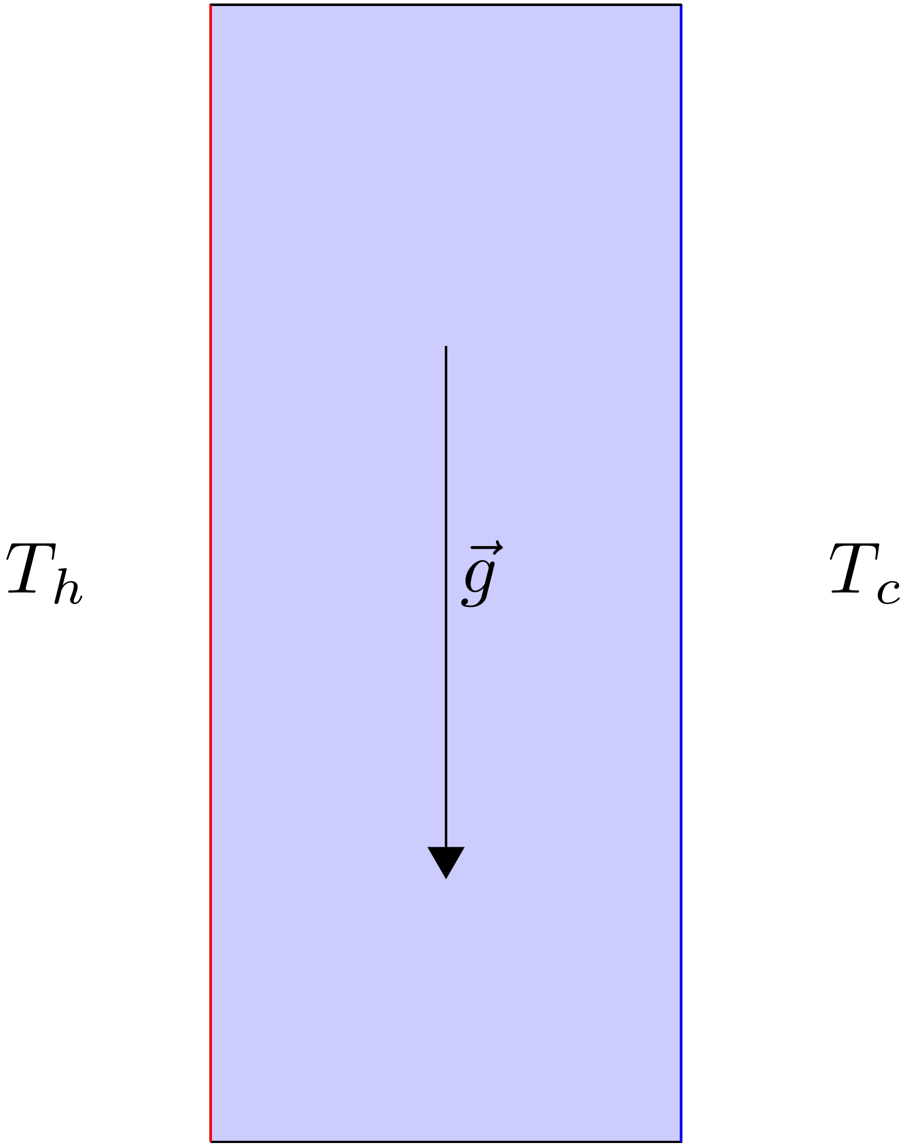
Problem II: Kinetic Energy : $\varepsilon_t = 3 \times 10^{-5}$



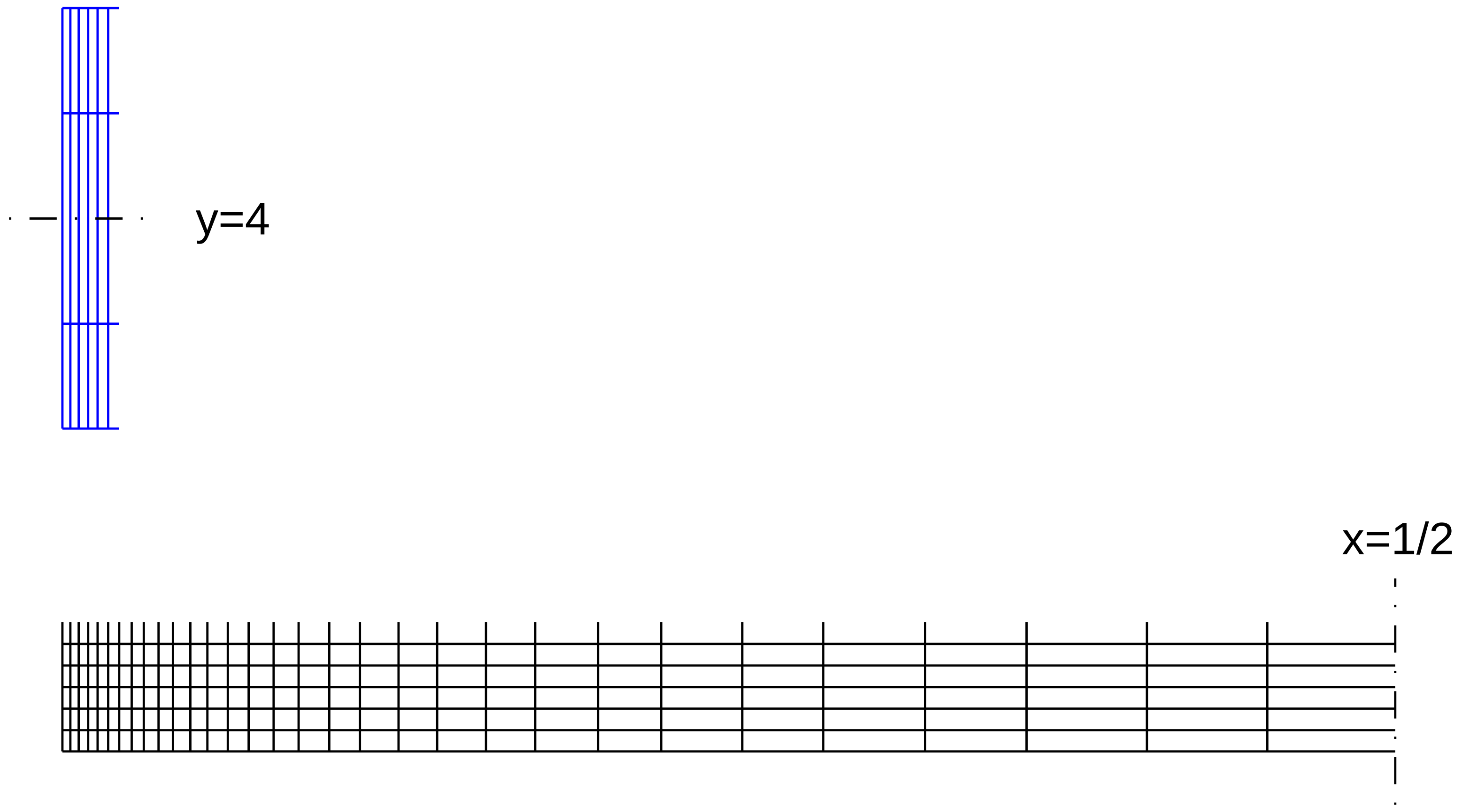
Problem II: Time step history : $\varepsilon_t = 3 \times 10^{-5}$



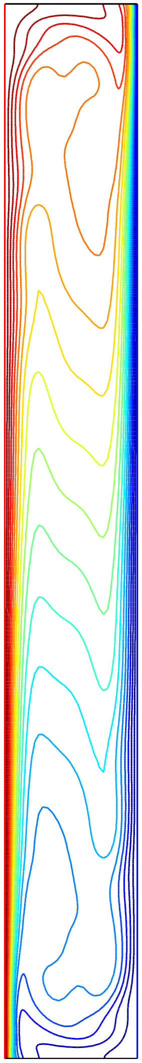
Problem XXX: 8:1 cavity domain



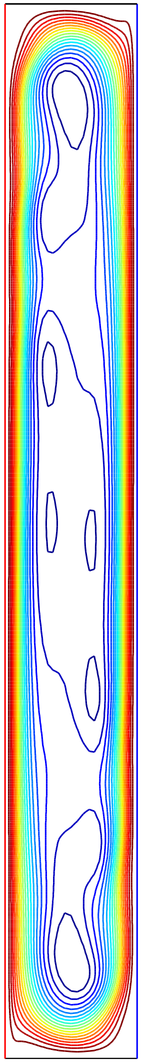
Problem XXX: 31×248 **stretched grid**



Problem XXX: Snapshot Solution

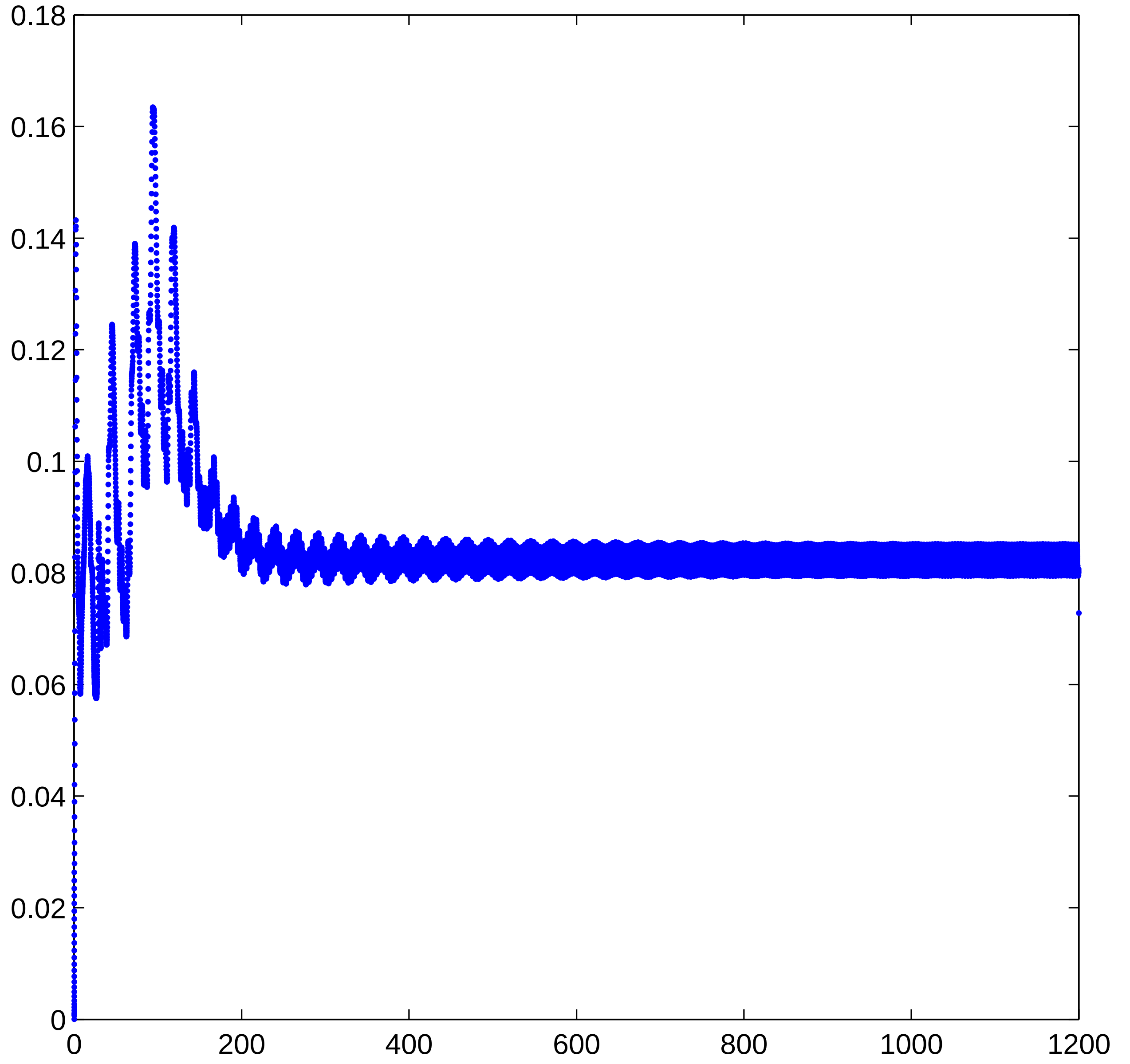


Isotherms : t=1200

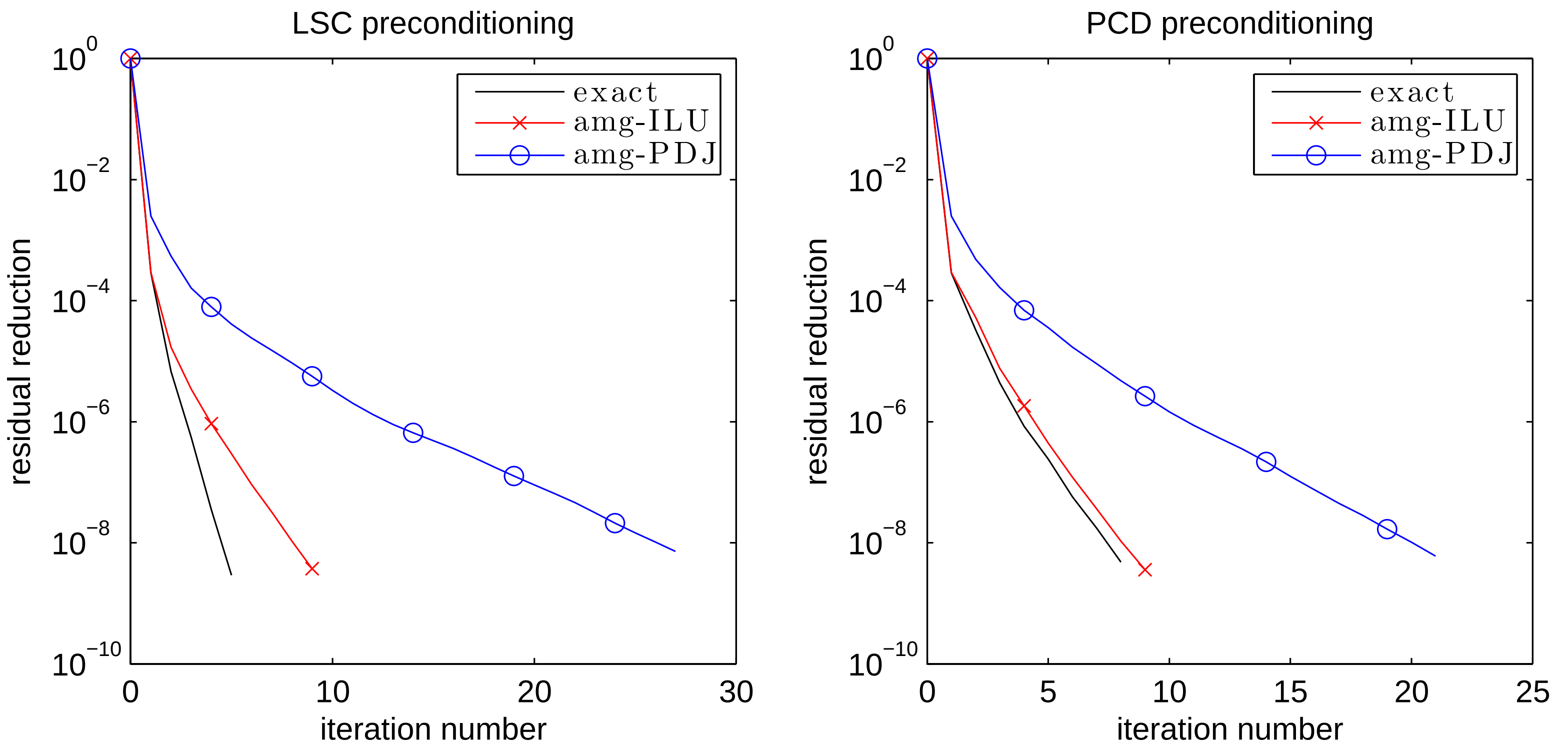


Streamlines : t=1200

Problem XXX: Time step history : $\varepsilon_t = 3 \times 10^{-5}$



Problem XXX: Solver performance

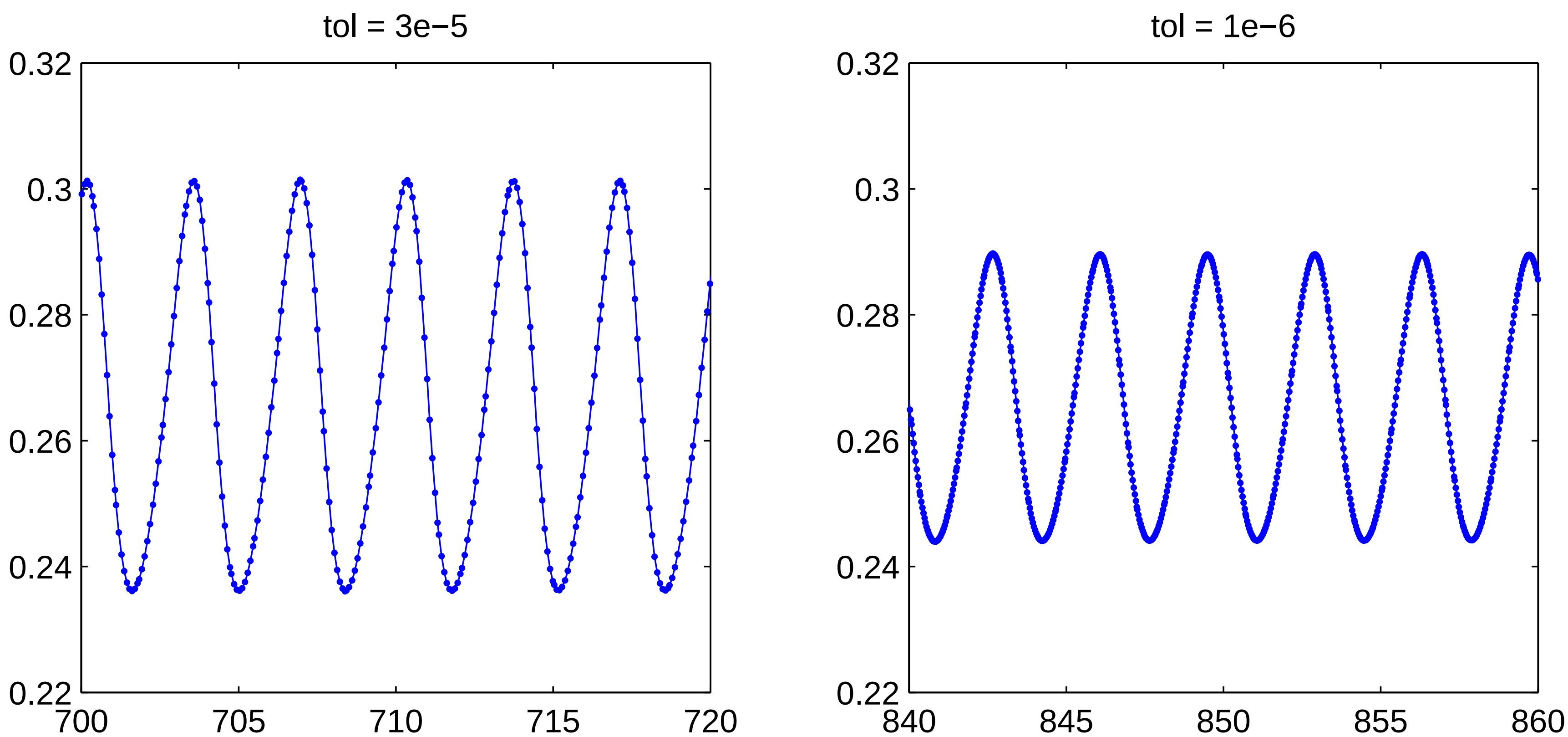


GMRES convergence for snapshot solution with $k_n \sim 0.082$.
Note that $\nu = 0.00145$ and $\nu = 0.00203$.

Problem XXX: Reference statistics

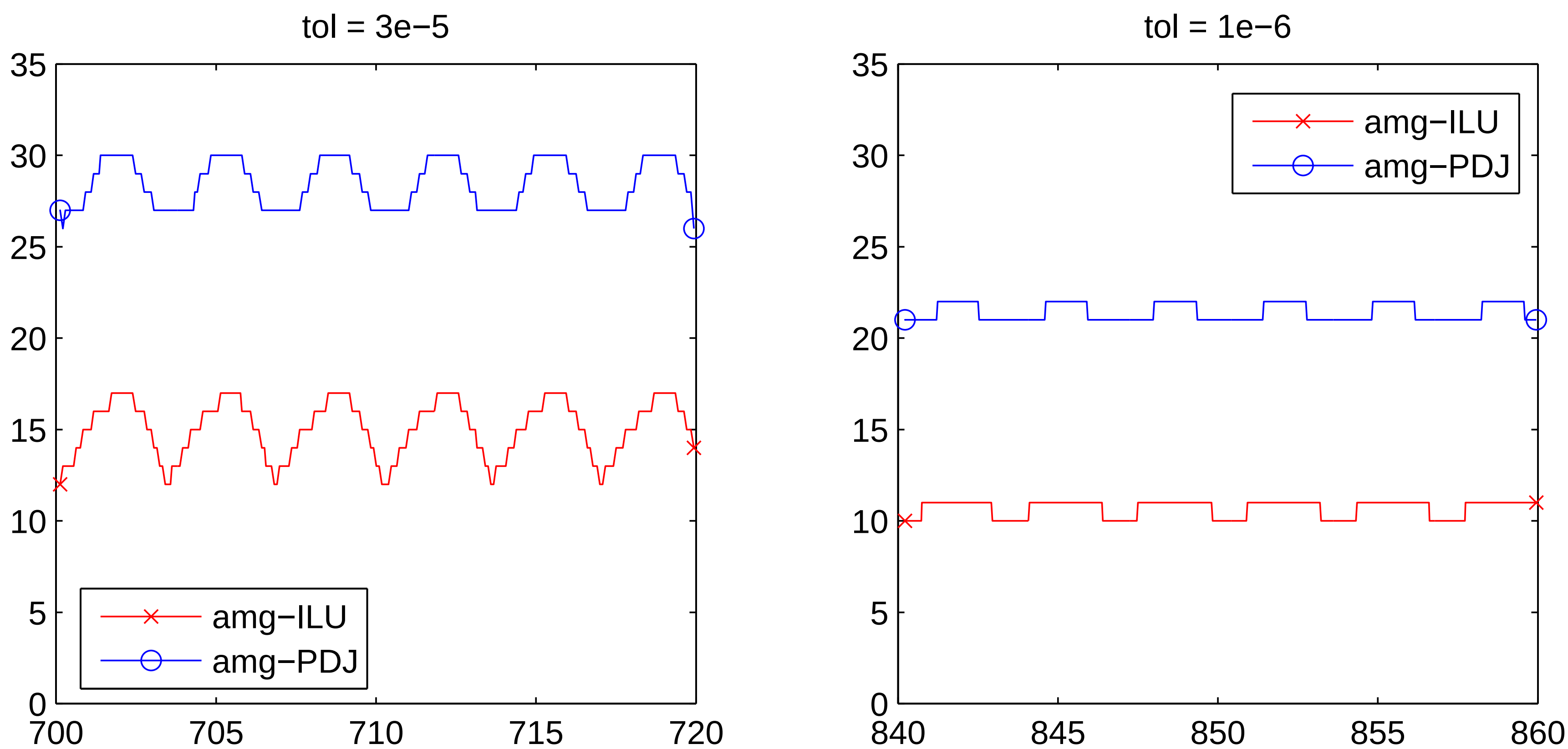
	MIT Benchmark	$\varepsilon_t = 3 \cdot 10^{-5}$	$\varepsilon_t = 1 \cdot 10^{-6}$
$(\Delta p)_{\min}$	-0.0125	-0.0178	-0.0135
$(\Delta p)_{\max}$	0.0074	0.0116	0.0082
$\Delta(\Delta p)$	0.0198	0.0294	0.0218
$\overline{\Delta p}$	-0.0026	-0.0031	-0.0027
$T_{\min}$	0.2461	0.2362	0.2442
$T_{\max}$	0.2872	0.3012	0.2896
ΔT	0.0411	0.0650	0.0454
$\overline{T}$	0.2666	0.2687	0.2669
Period	3.4135	3.382	3.412

Problem XXX: Tolerance comparison



Temperature evolution at the MIT reference point.

Problem XXX: Tolerance comparison



Iteration counts using inexact PCD preconditioning.

What have we achieved?

- Black-box implementation: few parameters that have to be estimated a priori.
- Optimal complexity: essentially $O(n)$ flops per iteration, where n is dimension of the discrete system.
- Efficient linear algebra: convergence rate is (essentially) independent of h. Given an appropriate time accuracy tolerance convergence is also robust with respect to ν