MPE: Numerical Methods

Christmas Lectures

Lawrence Mitchell[1]

Autumn term 2017

[1] lawrence.mitchell@imperial.ac.uk

Sparse linear algebra

Sparse linear algebra: motivation

We wish to solve

$$A\mathbf{x} = \mathbf{b}$$

where A is sparse, normally coming from the discretisation of a PDE.

- Recall, iterative methods for linear systems never need A itself.
- Fixed point iterations and Krylov subspace methods only ever use A in context of matrix-vector product.

Corollaries

- Only need to provide matrix-vector product to solvers.
- If storing A, exploit sparse structure.

Sparse matrix formats

- Rather than storing a dense array (with many zeros), store only the non-zero entries, plus their locations.
- Data size becomes $\mathcal{O}(n_{\text{nz}})$ rather than $\mathcal{O}(n_{\text{row}} n_{\text{col}})$.
- For finite stencils (as from mesh-based discretisations) asymptotically save $\mathcal{O}(n_{\text{col}})$.

Name	Easy insertion	Fast $A\mathbf{x}$	$A + B$
Coordinate (COO)	Yes	No	Easy
CSR	No	Yes	Hard[2]
CSC	No	Yes	Hard[2]
ELLPACK	No	Yes	Hard[2]

Table: Common sparse storage types. Saad 2003, § 3.4 provides a nice discussion of various formats.

[2] unless A and B have matching sparsity

Sparse matrix formats II: a zoo

Many formats

Operations with sparse matrices are bounded by the memory bandwidth of the machine. The proliferation of slight variations to the CSR format all attempt to exploit extra structure in the matrix to increase performance through vectorisation and better cache reuse.

Common interface

Fortunately, you shouldn't have to care. A sparse matrix library should offer a consistent interface to insert values, and perform matrix operations, irrespective of the underlying format.

Sparse matrix implementation: libraries

Maxim

The most important part of programming is knowing when not to write your own code.

There are many full-featured sparse libraries available (serial and parallel). When you need sparse linear algebra, take the time to learn one.

Name	Language	Fortran?	Python?	Parallel	PCs
PETSc[3]	C	Yes	Yes	Yes	Many
`scipy.sparse`[4]	Python	No	Yes	No	Some
EIGEN[5]	C++	No	No	No	Some
Trilinos[6]	C++	No	Yes	Yes	Many

Table: Some sparse libraries

[3] mcs.anl.gov/petsc
[4] docs.scipy.org/doc/scipy/reference/sparse.html
[5] eigen.tuxfamily.org
[6] trilinos.org

Some advice

- We've seen already that iterative methods only need $A\mathbf{x}$.
- But, it is important to be able to precondition the solver.
- Assembled sparse matrix formats give you good performance, and access to a wide suite of preconditioners.

Maxim

Always start by implementing problems with assembled operators. Now you can try lots of things quickly and get your model working. Then, and only then can you start worrying about further performance optimisations.

Preconditioning Krylov methods

Questions upon encountering a matrix

1. What do you want to do with it?
   - Compute $A\mathbf{x}$?
   - Solve linear systems (or eigen-problems)?
2. What does the spectrum look like?
   - Are the eigenvalues all distinct, or clustered?
   - Symmetric positive definite? $\sigma(A) \subset \mathbb{R}^+$
   - Nonsymmetric definite? $\sigma(A) \subset \{z \in \mathbb{C} : \Re(z) > 0\}$
   - Symmetric indefinite? $\sigma(A) \subset \mathbb{R}$
   - Nonsymmetric indefinite? $\sigma(A) \in \mathbb{C}$
3. What is its sparsity?
4. Is there a better way of computing $A\mathbf{x}$ than by starting with A?
5. Is there another matrix whose spectrum is similar, but is "nicer"?
6. How can we precondition A?

Krylov methods are not solvers

Assertion (Krylov solvers are not solvers)

Despite guarantees of convergence in exact arithmetic for CG (and GMRES), in actual practical cases a bare Krylov method is almost useless.

- Krylov methods converge fast if:
  1. there is a low-degree polynomial with $p(0) = 1$ with $p(\lambda_i) = 0 \; \forall \lambda_i$, or
  2. you're lucky and you get a "special" right hand side.
- Convergence to a tolerance requires $p(\lambda_i)$ small. Achievable if eigenvalues are clustered.
- For most operators we will encounter, the eigenvalues are typically not clustered.

Preconditioning to the rescue

Definition (Preconditioner)

A preconditioner $\mathcal{P}$ is a method for constructing a linear operator $P^{-1} = \mathcal{P}(A, A_p)$ using a matrix A and some extra information A_p, such that the spectrum of $P^{-1}A$ (or AP^{-1}) is well-behaved.

- P^{-1} is dense, and P itself is often not available (and not needed).
- Normally, A is not used by $\mathcal{P}$. But often we make the choice $A_p = A$.
- Often $\mathcal{P}$ can be a (matrix-based) "black-box". Things like Jacobi, Gauss-Seidel, (incomplete) factorisations fall into this category.
- If you know something about A, you can often do better than a black-box approach.

If you're writing a simulation

Direct solvers (LU factorisation)

Reasonable for medium-sized problems, robust but not scalable.

- 2D $\mathcal{O}(N_{\text{dof}}^{3/2})$ flops, $\mathcal{O}(N_{\text{dof}} \log N_{\text{dof}})$ memory.
- 3D $\mathcal{O}(N_{\text{dof}}^{2})$ flops, $\mathcal{O}(N_{\text{dof}}^{4/3})$ memory.

If you're writing a simulation

Direct solvers (LU factorisation)

Reasonable for medium-sized problems, robust but not scalable.

- 2D $\mathcal{O}(N_{\text{dof}}^{3/2})$ flops, $\mathcal{O}(N_{\text{dof}} \log N_{\text{dof}})$ memory.
- 3D $\mathcal{O}(N_{\text{dof}}^{2})$ flops, $\mathcal{O}(N_{\text{dof}}^{4/3})$ memory.

1. Develop your problem at small scale, using a (sparse) direct solver. "Get all the maths right".

If you're writing a simulation

Direct solvers (LU factorisation)

Reasonable for medium-sized problems, robust but not scalable.

- 2D $\mathcal{O}(N_{\text{dof}}^{3/2})$ flops, $\mathcal{O}(N_{\text{dof}} \log N_{\text{dof}})$ memory.
- 3D $\mathcal{O}(N_{\text{dof}}^{2})$ flops, $\mathcal{O}(N_{\text{dof}}^{4/3})$ memory.

1. Develop your problem at small scale, using a (sparse) direct solver. "Get all the maths right".
2. Switch to an iterative method, weep quietly as your problem no longer converges.

If you're writing a simulation

Direct solvers (LU factorisation)

Reasonable for medium-sized problems, robust but not scalable.

- 2D $\mathcal{O}(N_{\text{dof}}^{3/2})$ flops, $\mathcal{O}(N_{\text{dof}} \log N_{\text{dof}})$ memory.
- 3D $\mathcal{O}(N_{\text{dof}}^{2})$ flops, $\mathcal{O}(N_{\text{dof}}^{4/3})$ memory.

1. Develop your problem at small scale, using a (sparse) direct solver. "Get all the maths right".
2. Switch to an iterative method, weep quietly as your problem no longer converges.
3. Read the literature to find a robust h-independent preconditioner (iterations constant irrespective of resolution).

If you're writing a simulation

Direct solvers (LU factorisation)

Reasonable for medium-sized problems, robust but not scalable.

- 2D $\mathcal{O}(N_{\text{dof}}^{3/2})$ flops, $\mathcal{O}(N_{\text{dof}} \log N_{\text{dof}})$ memory.
- 3D $\mathcal{O}(N_{\text{dof}}^{2})$ flops, $\mathcal{O}(N_{\text{dof}}^{4/3})$ memory.

1. Develop your problem at small scale, using a (sparse) direct solver. "Get all the maths right".
2. Switch to an iterative method, weep quietly as your problem no longer converges.
3. Read the literature to find a robust h-independent preconditioner (iterations constant irrespective of resolution).
4. ... (implementation).

If you're writing a simulation

Direct solvers (LU factorisation)

Reasonable for medium-sized problems, robust but not scalable.

- 2D $\mathcal{O}(N_{\text{dof}}^{3/2})$ flops, $\mathcal{O}(N_{\text{dof}} \log N_{\text{dof}})$ memory.
- 3D $\mathcal{O}(N_{\text{dof}}^{2})$ flops, $\mathcal{O}(N_{\text{dof}}^{4/3})$ memory.

1. Develop your problem at small scale, using a (sparse) direct solver. "Get all the maths right".
2. Switch to an iterative method, weep quietly as your problem no longer converges.
3. Read the literature to find a robust h-independent preconditioner (iterations constant irrespective of resolution).
4. ... (implementation).
5. Solve at scale (without waiting until next year).

Choosing a preconditioner: connections to PDEs

- We often think of preconditioning in the context of "I have a matrix system I want to solve".
- However, there is a very deep connection between preconditioning and functional analysis (and the theory of PDEs).
- In particular, figuring out what an appropriate preconditioner is.
- For more details, Kirby (2010) and Málek and Strakoš (2014) provide a good introduction.

A sketch for CG

- We can formulate Krylov methods in Hilbert spaces. Let

$$A : V \to V; \quad b \in V.$$

- A Krylov method seeks an "optimal"

$$x_m \in K_m(A, b) = \text{span}\{b, Ab, A^2 b, \ldots, A^{m-1} b\},$$

  where K_m is the Krylov basis.
- CG is appropriate if A is SPD and finds x_m minimising the A-norm of the error:

$$x_m = \underset{y \in K_m}{\arg\min} \langle Ay, y \rangle - 2\langle b, y \rangle$$

- Note that this construction requires that $A : V \to V$.

Where's the problem?

- For a discretisation of a PDE, we typically have

$$A : V \to V^*.$$

- Consider an H^1 discretisation of the Laplacian. This maps from H^1 (the space of piecewise smooth functions) to its dual H^{-1}. But

$$H^1 \subset L^2 \subset H^{-1}$$

- So now $V^* \neq V$. But CG requires that $b, Ab, \ldots, \in V$.
- We can think of preconditioning as fixing this "type-error" by choosing $B : V^* \to V$ and then solving the preconditioned problem

$$BA : V \to V^* \to V.$$

- Analysis of the PDE tells you an appropriate choice of B.

An exemplar problem

A concrete example

Model problem

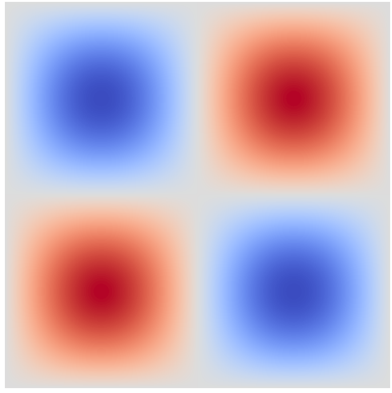
$$
\begin{aligned}
-\nabla^2 u(x, y) &= f(x, y), && \text{in } \Omega = [-1, 1]^2 \\
u(x, y) &= 0. && \text{on } \partial\Omega
\end{aligned}
$$

Discretised with 5-point stencil on regular grid (expect $\mathcal{O}(h^2)$ convergence of error).

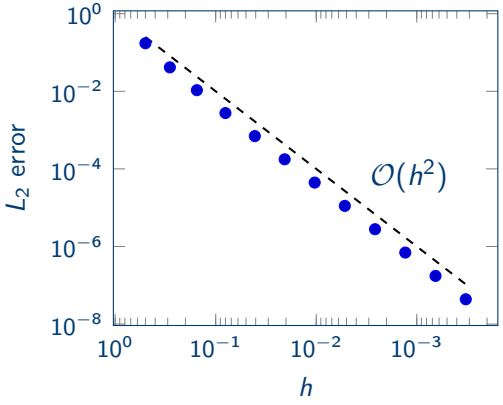
Is my code correct?

First, I need to check that I have implemented things correctly

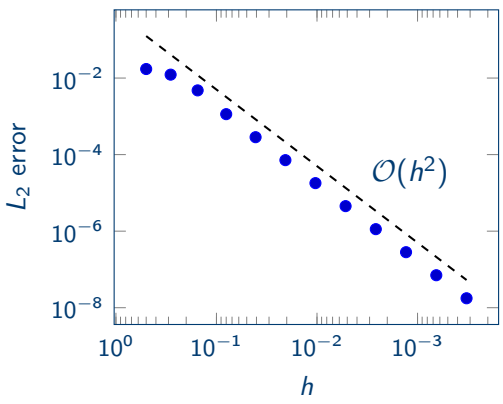
Two exact solutions



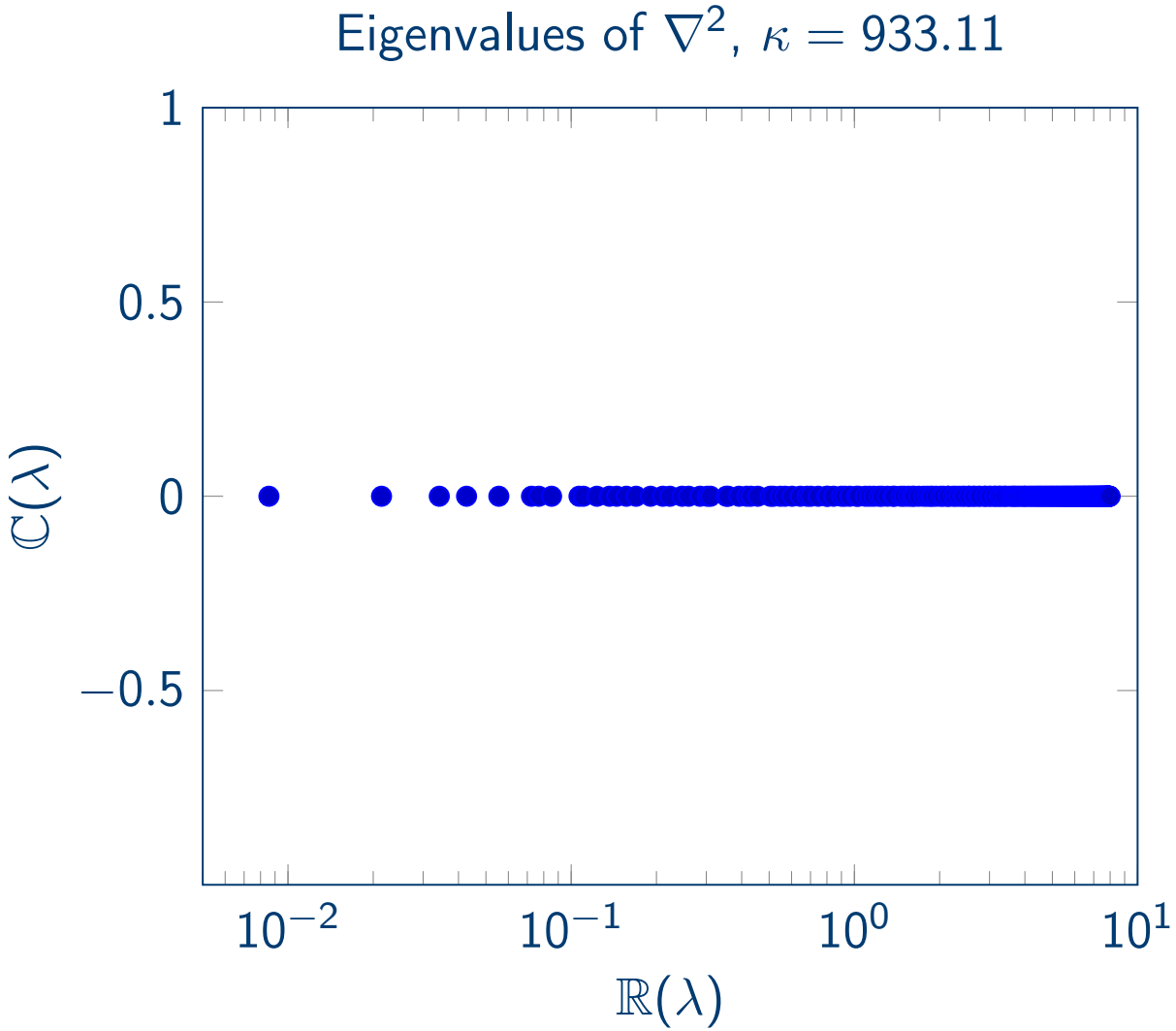
$u(x, y) = \sin(\pi x) \sin(\pi y)$



$u(x, y) = \sin(\pi x) \sin(\pi y) \exp(-10(x^2 + y^2))$



Spectrum


Eigenvalues of ∇^2, $\kappa = 933.11$
1
0.5
0
−0.5
$\mathbb{C}(\lambda)$
10^{-2}
10^{-1}
10^{0}
10^{1}
$\mathbb{R}(\lambda)$

Expected convergence

Recall that the A-norm of the error at the k^{th} iteration is bounded above by

$$||u_* - u_k||_A = ||e_k||_A \leq 2||e_0||_A \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k .$$

Where $\kappa = |\lambda_{\text{max}}/\lambda_{\text{min}}|$ is the condition number of A (or the preconditioned A as appropriate).

Poisson convergence

The Laplacian has an h-dependent condition number:

$$\lim_{h \to 0} \kappa \sim \mathcal{O}(h^{-2})$$

and so we expect CG to converge in $\mathcal{O}(h^{-1})$ iterations.

Stopping criteria (a reminder)

CG minimises the A-norm of the error, but we don't have access to that while iterating (we don't know the solution!). However, we can bound the 2-norm of the error.

Stopping criteria (a reminder)

CG minimises the A-norm of the error, but we don't have access to that while iterating (we don't know the solution!). However, we can bound the 2-norm of the error.

Theorem

If we require $||r_k||_2 < \lambda_{min}^{-1}\delta$ *then we guarantee* $||u_* - u_k||_2 < \delta$.

Proof.

$$\begin{aligned} ||u_* - u_k||_2 &= ||A^{-1}A(u_* - u_k)||_2 \leq ||A^{-1}||_2||(b - Au_k)||_2 \\ &= \lambda_{\text{min}}^{-1}||r_k||_2. \end{aligned}$$

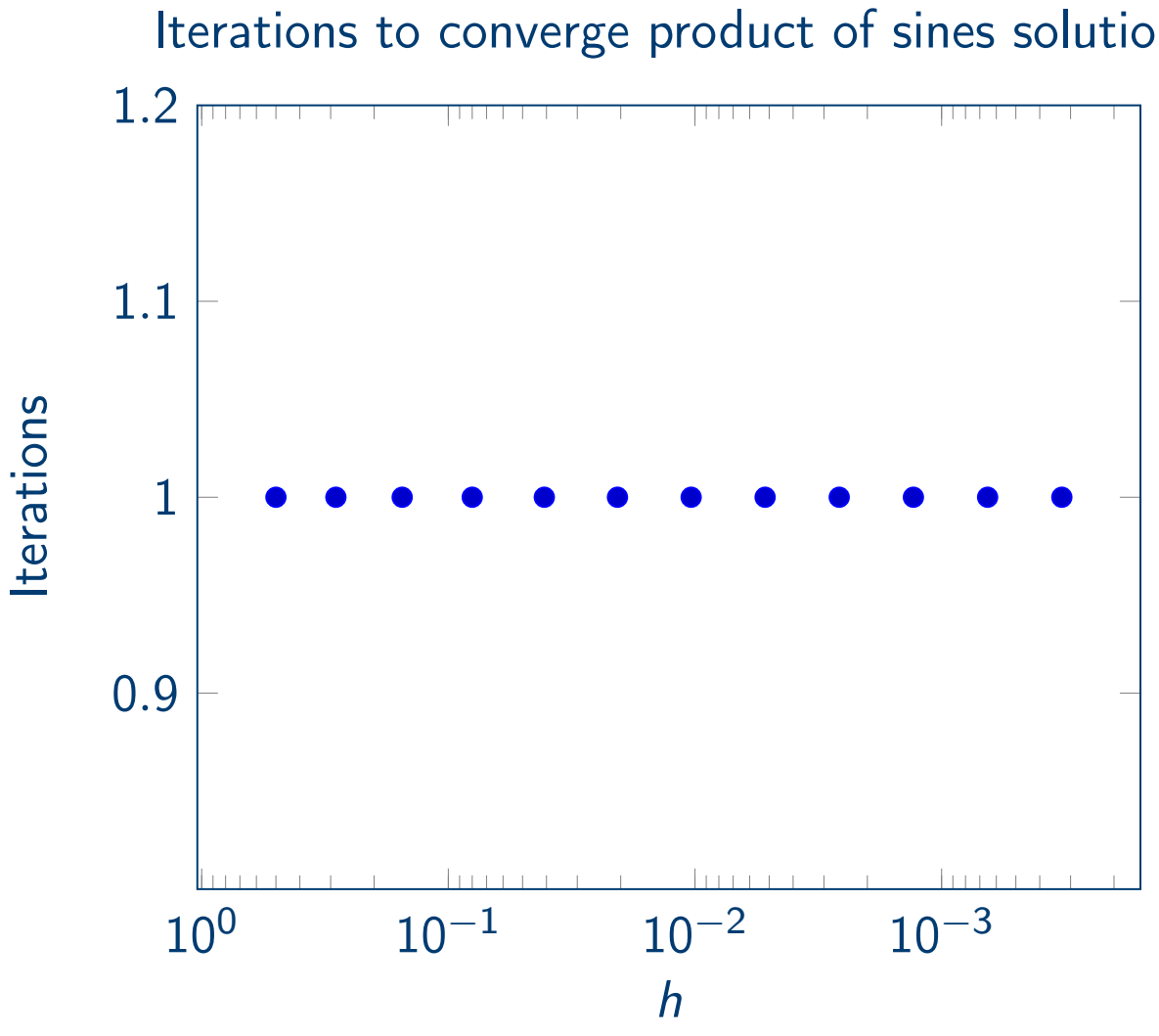
□

Back to the model problem

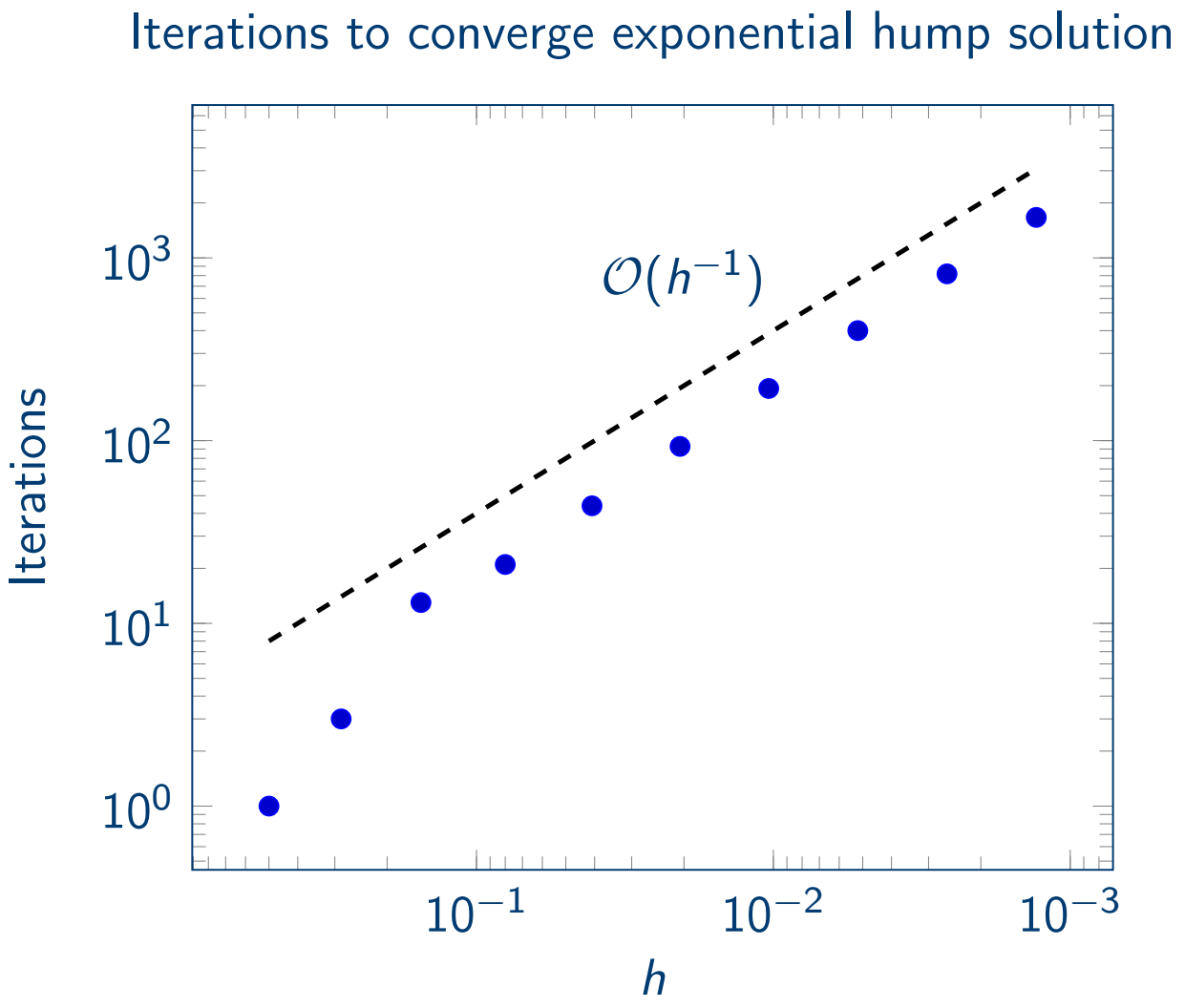
We've seen that the unpreconditioned operator has a bad spectrum for iterative solvers. Let's try when $u(x, y) = \sin(\pi x)\sin(\pi y)$

Back to the model problem

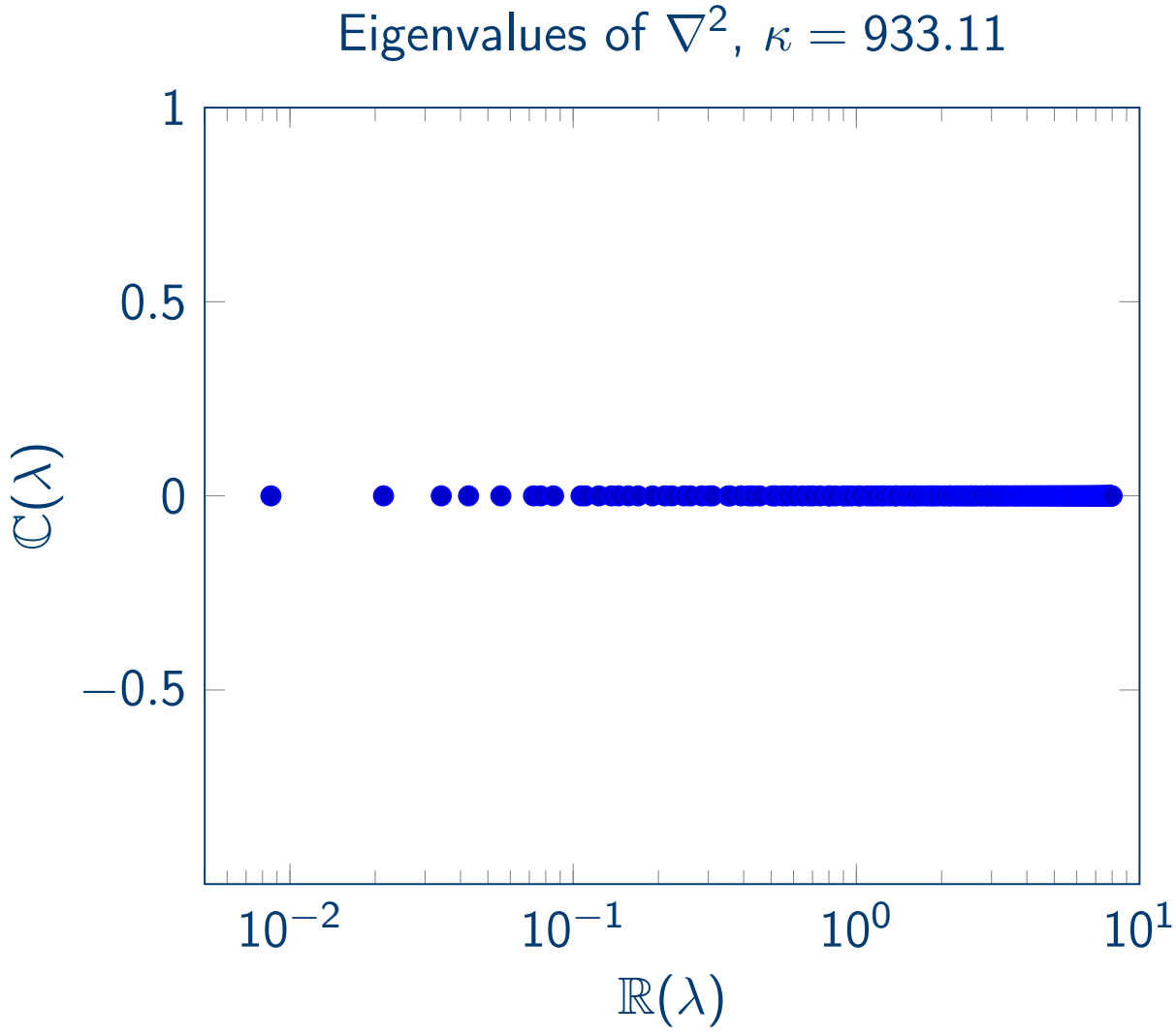
We've seen that the unpreconditioned operator has a bad spectrum for iterative solvers. Let's try when $u(x, y) = \sin(\pi x)\sin(\pi y)$



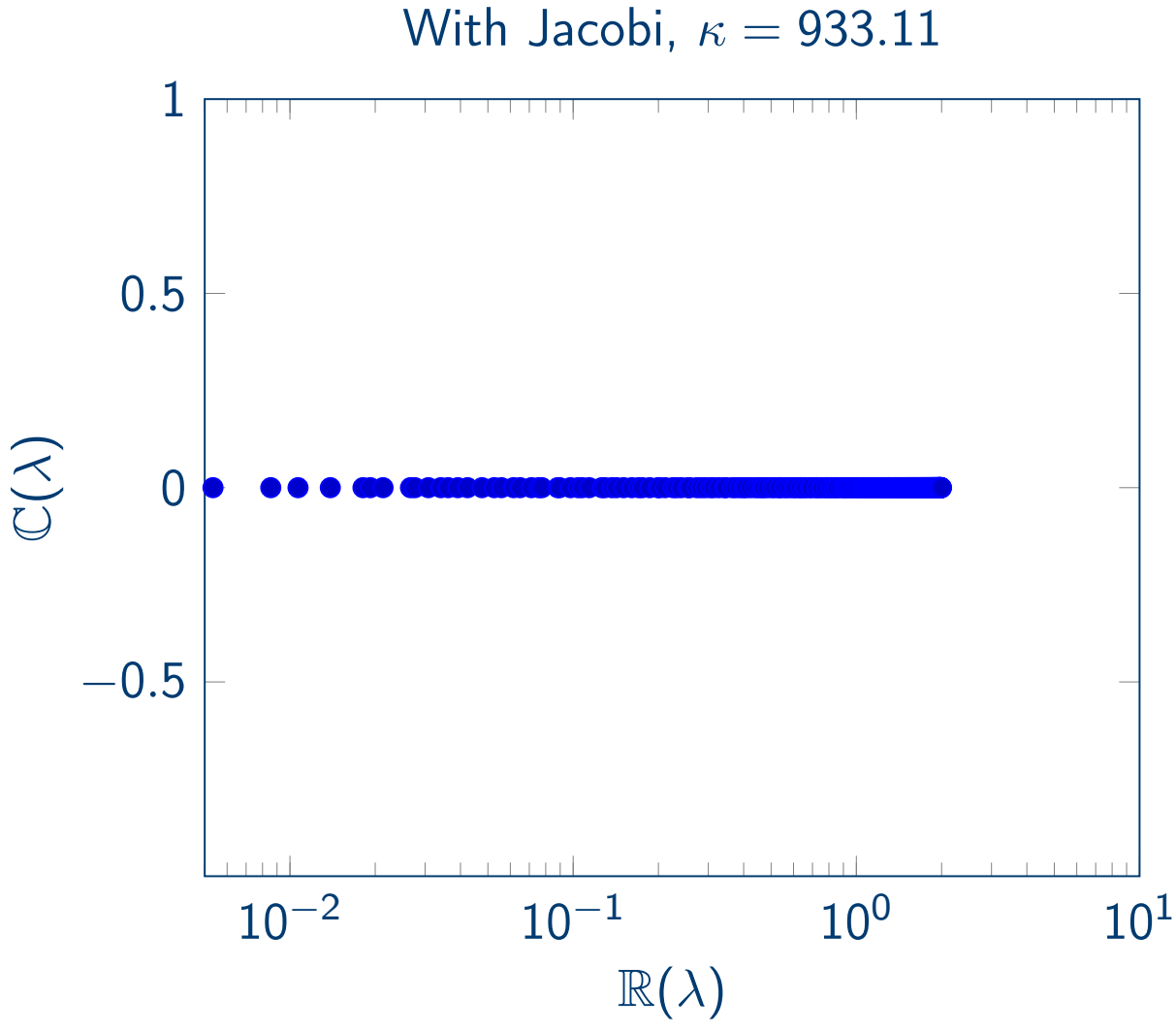
We had a special right hand side



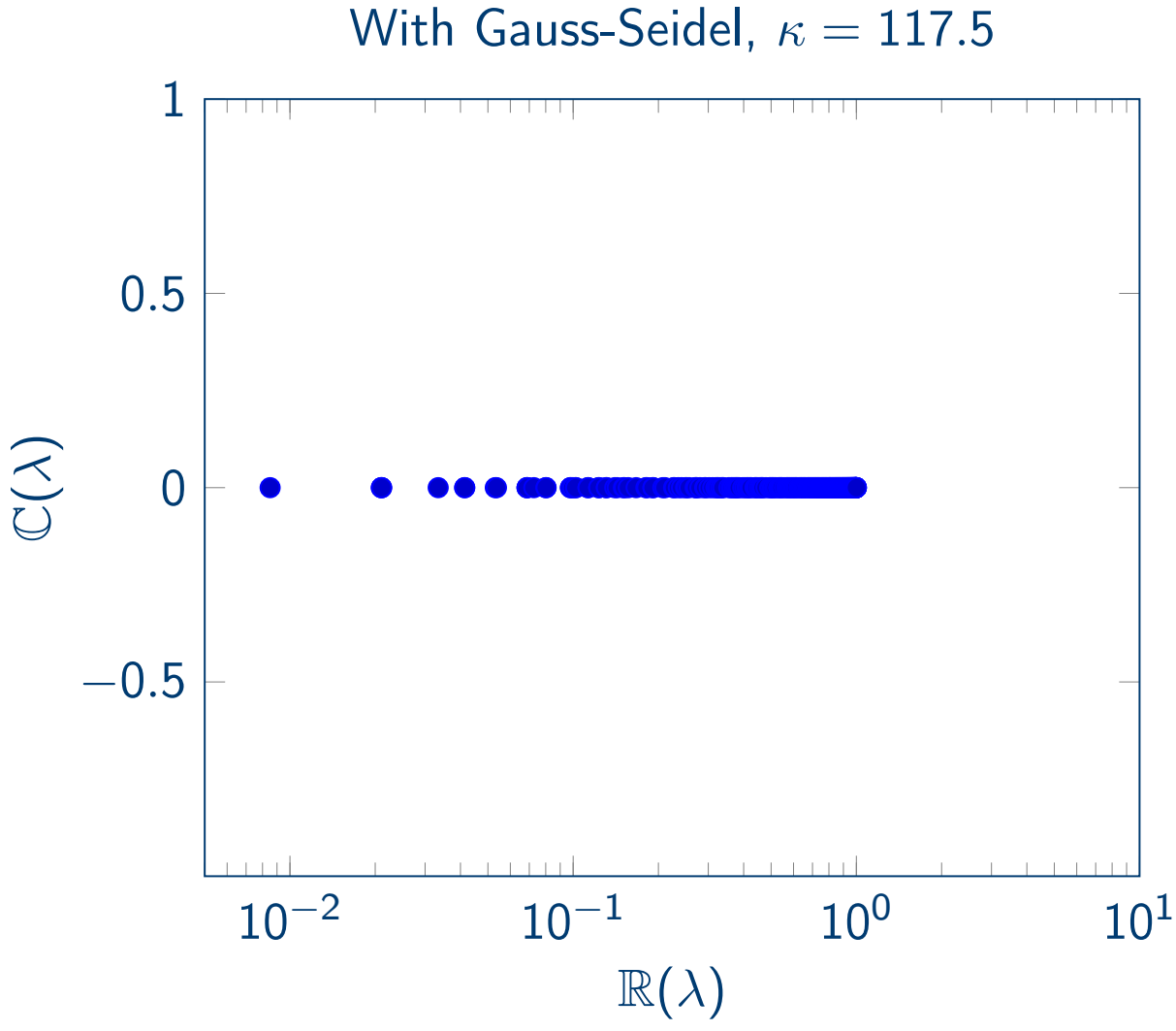
Some preconditioned spectra



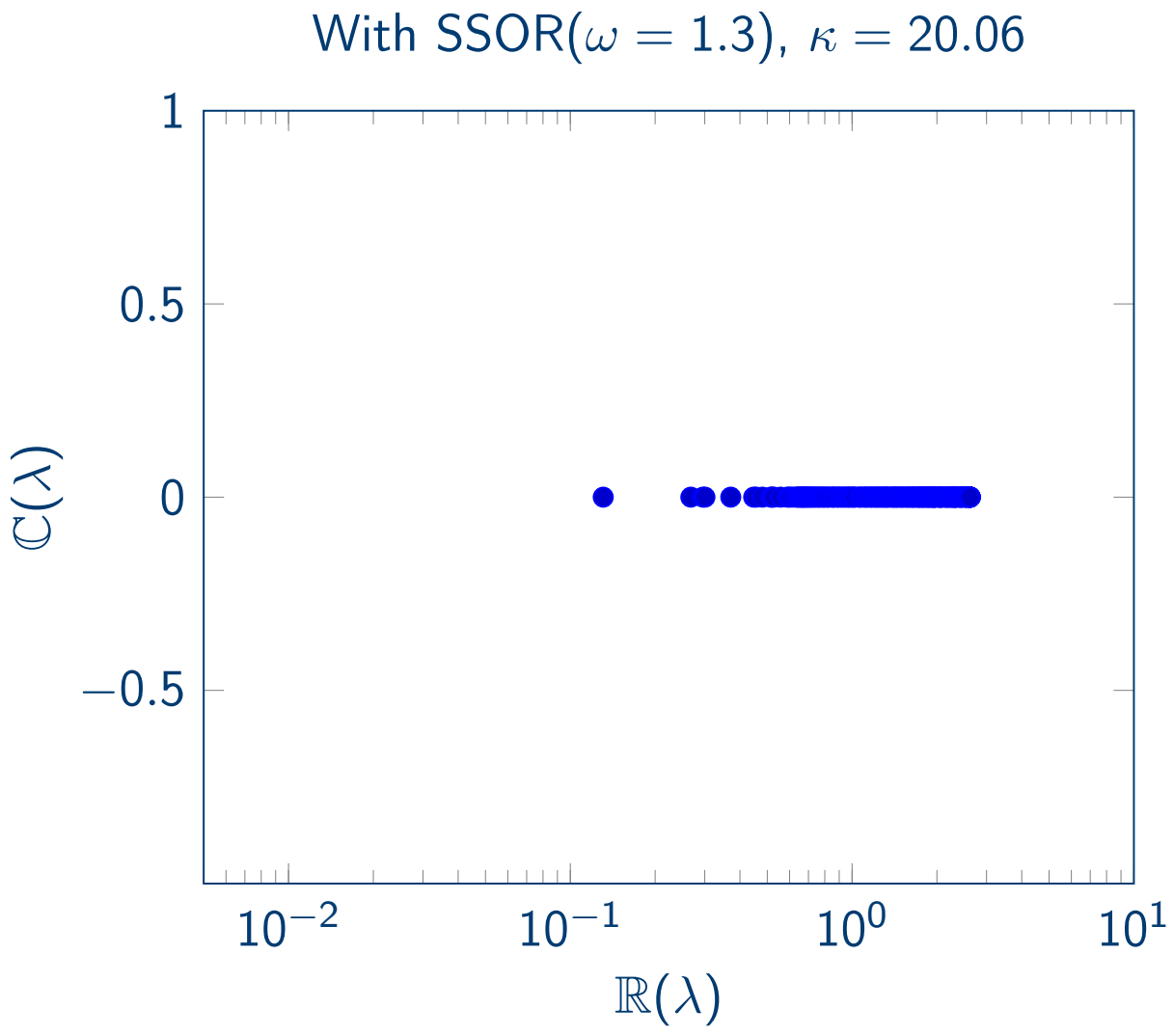
Some preconditioned spectra



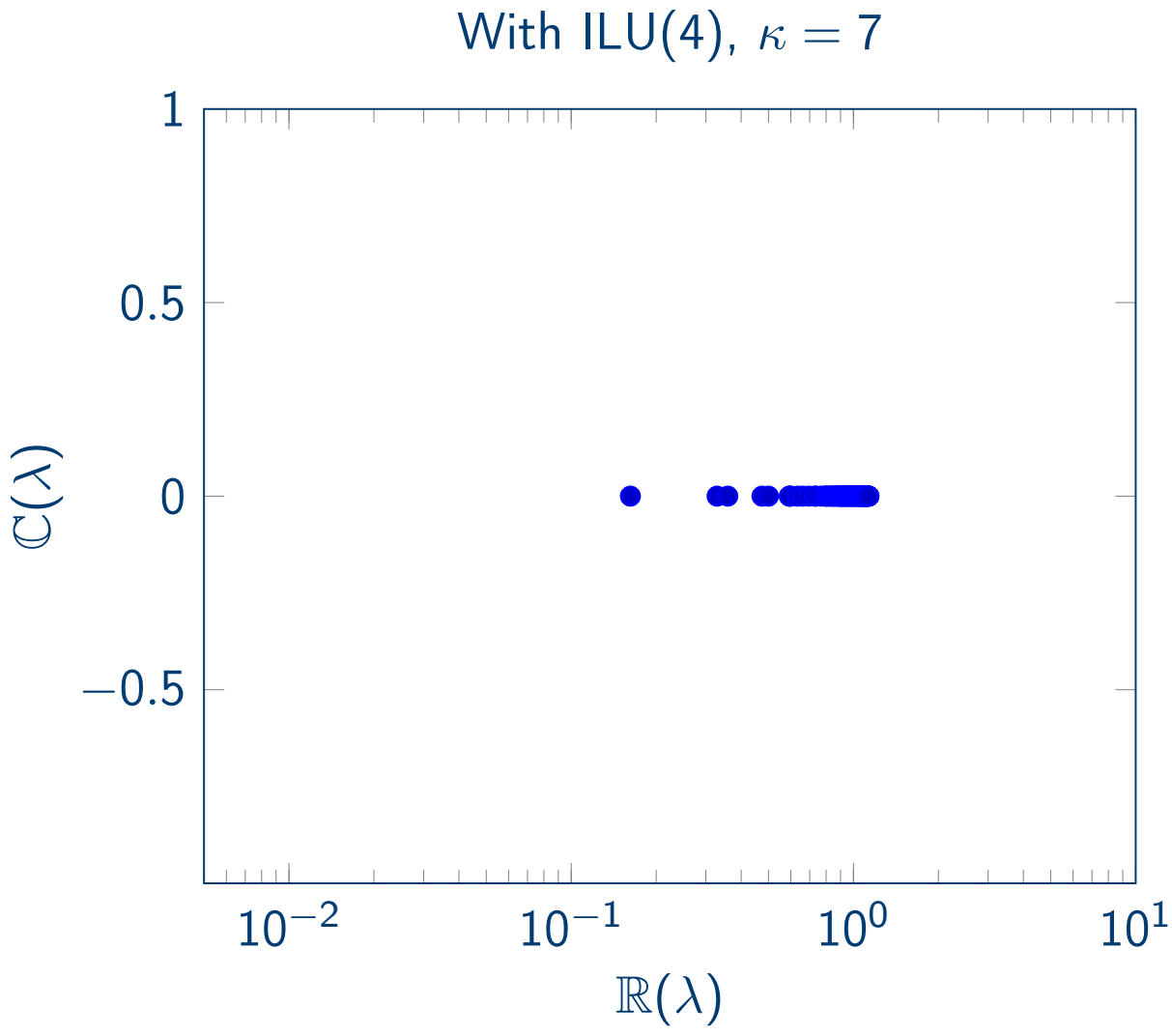
Some preconditioned spectra



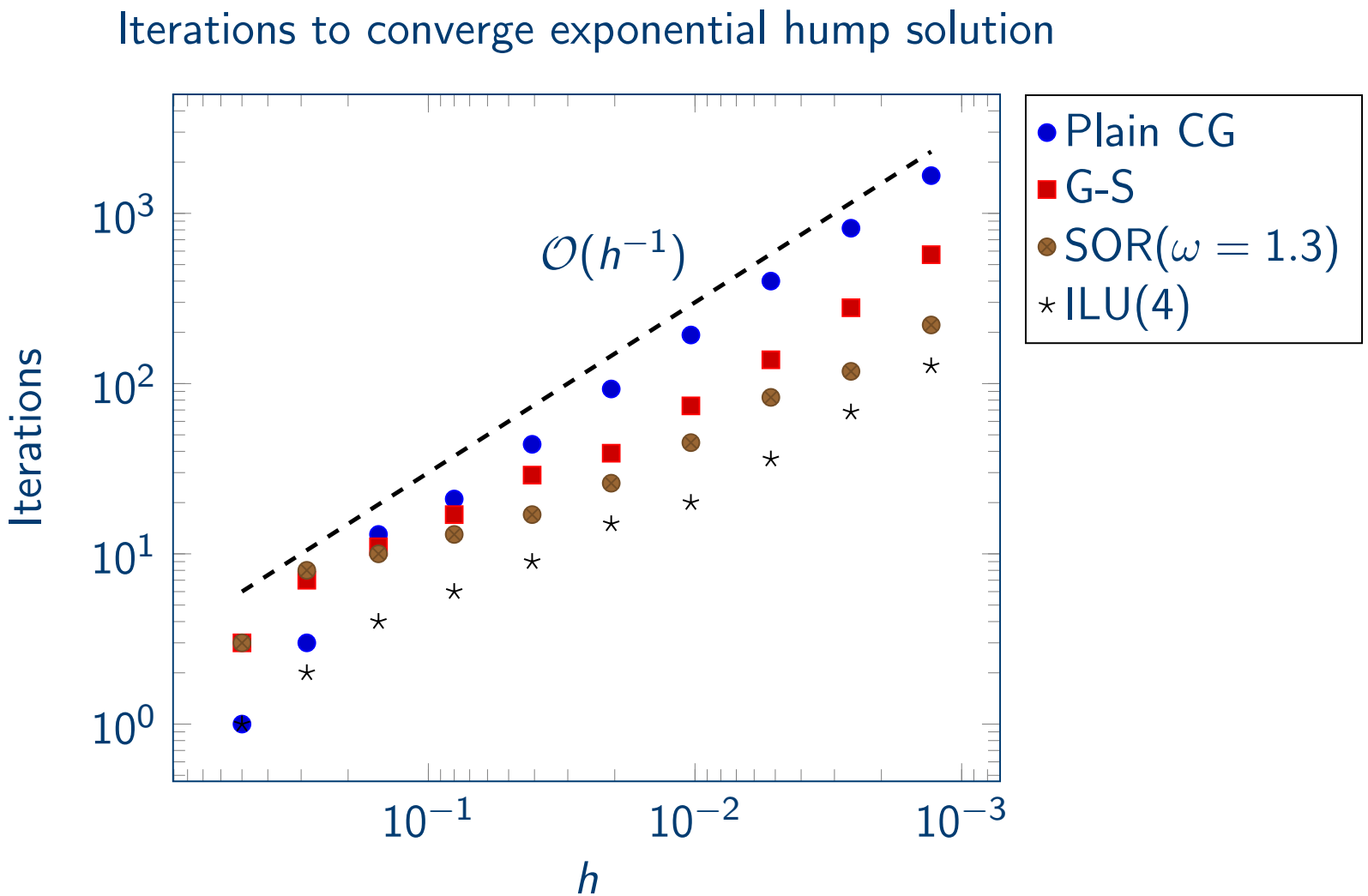
Some preconditioned spectra



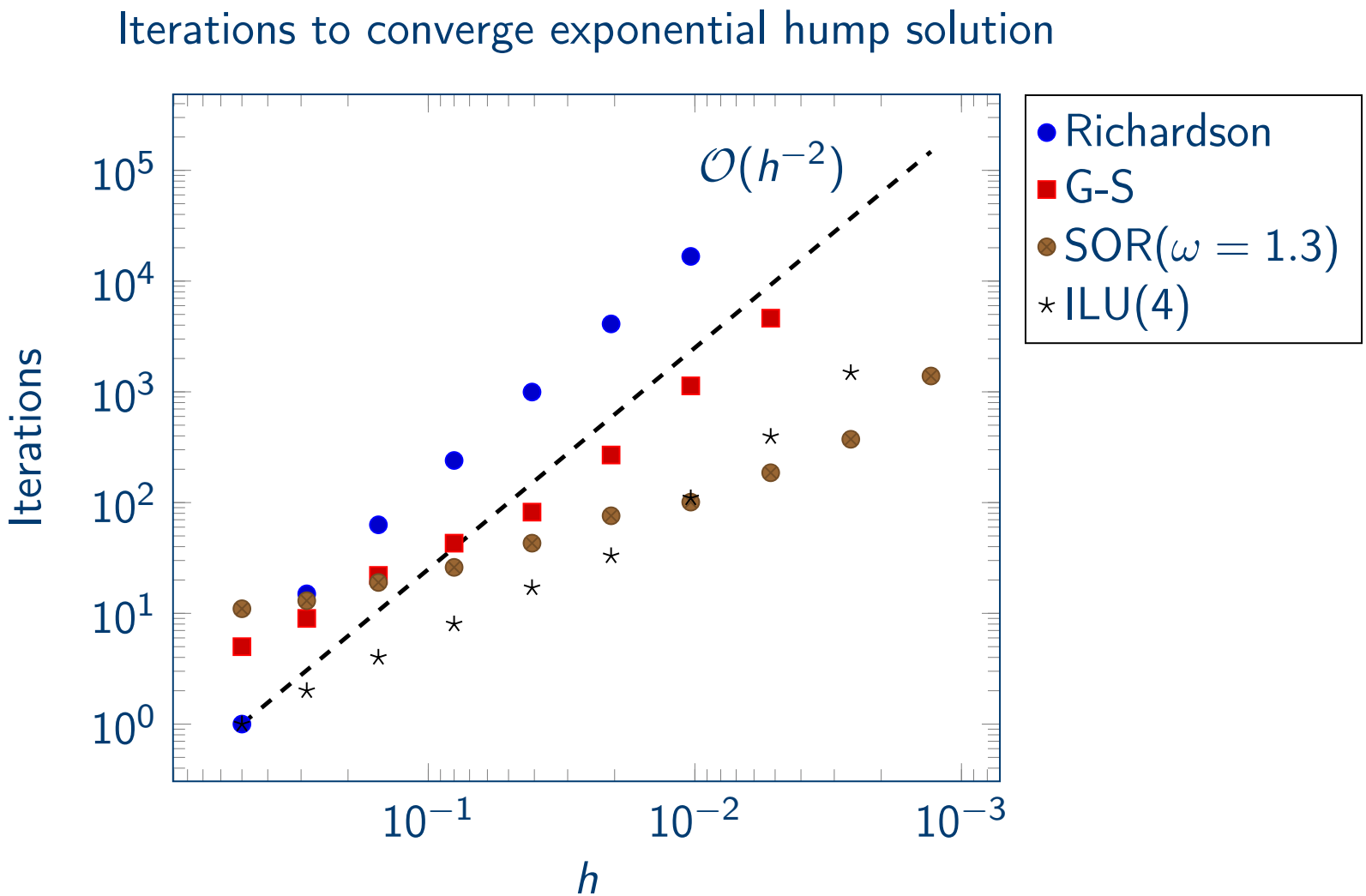
Some preconditioned spectra



Back to convergence

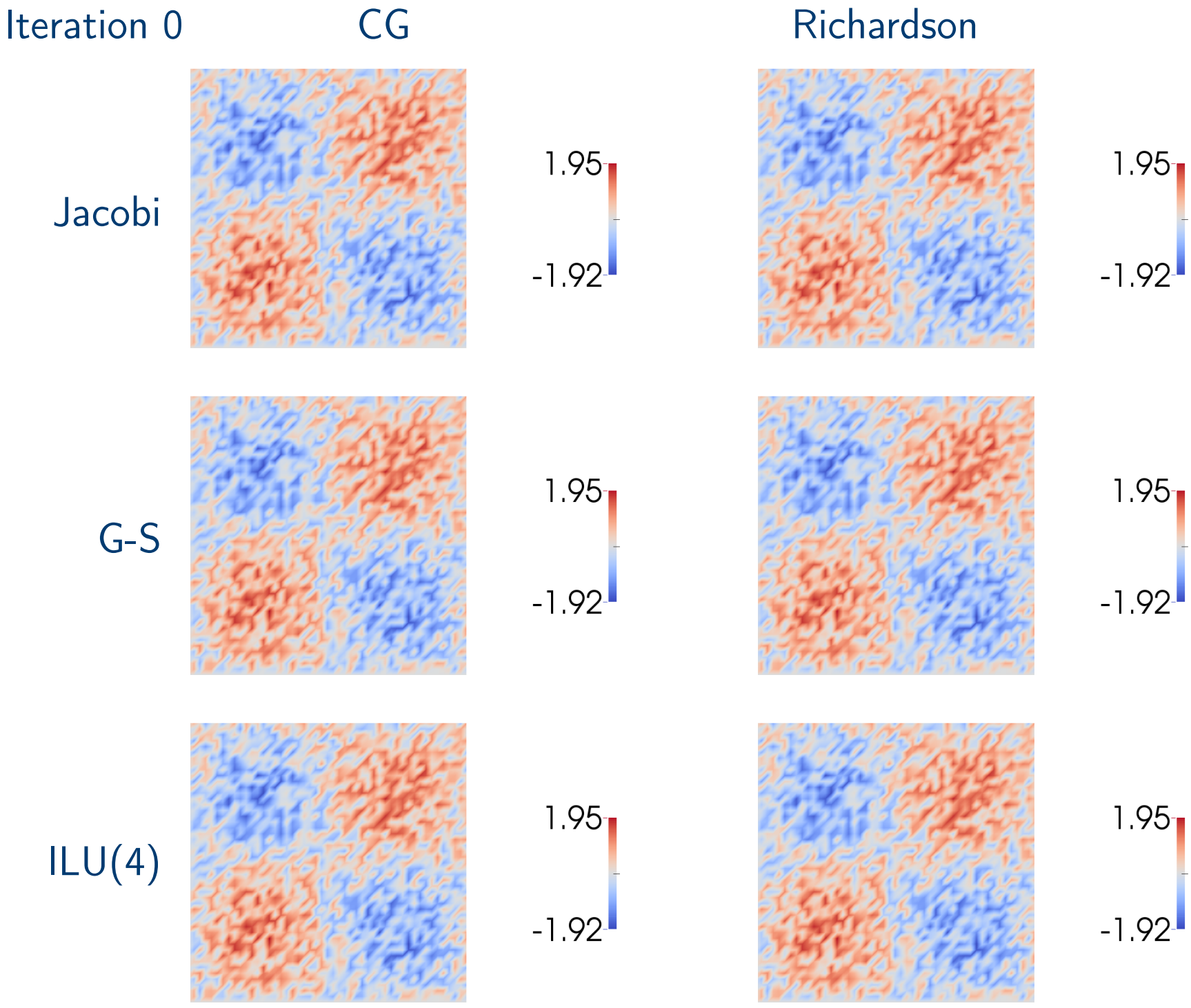


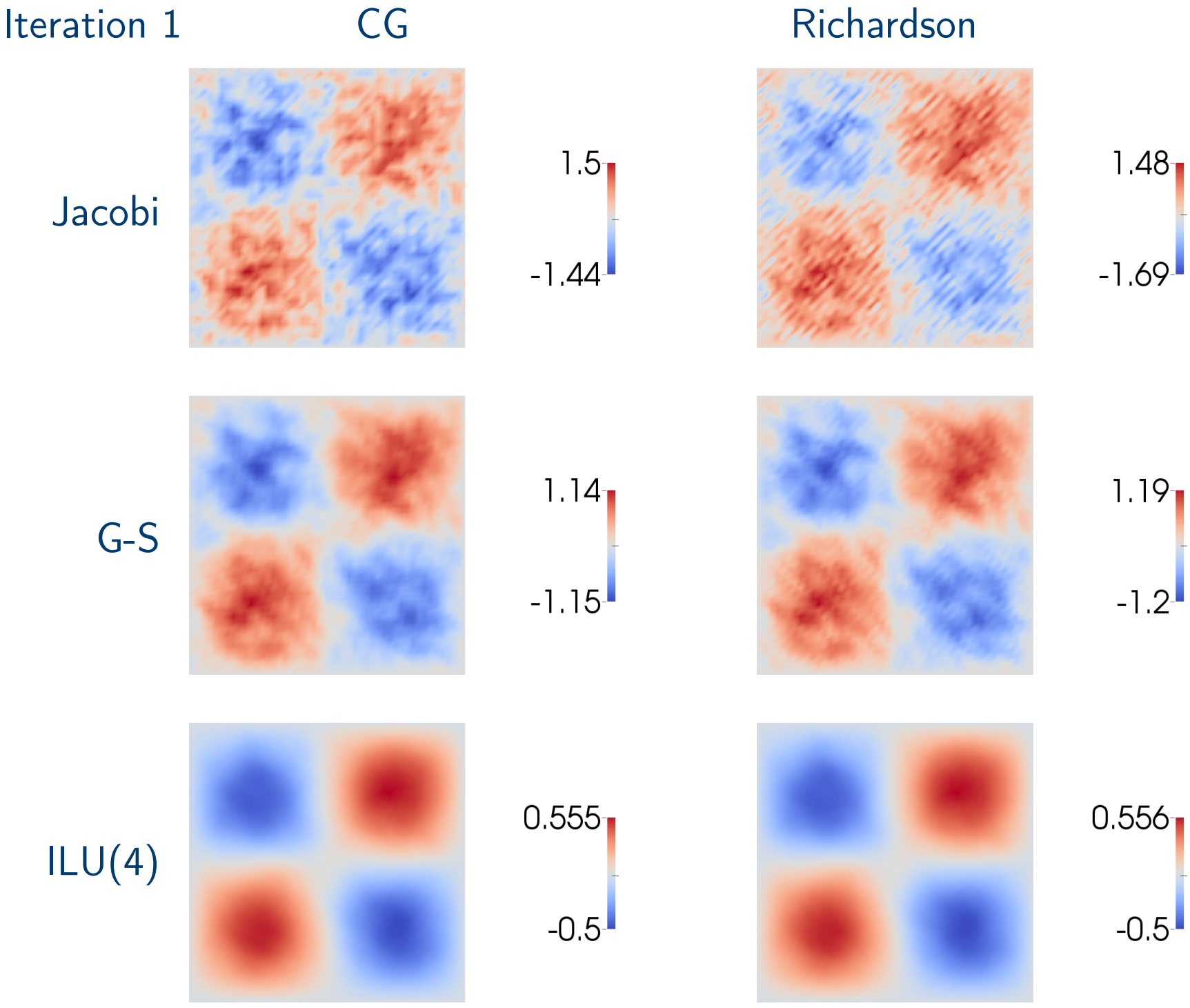
Compare (damped) Richardson

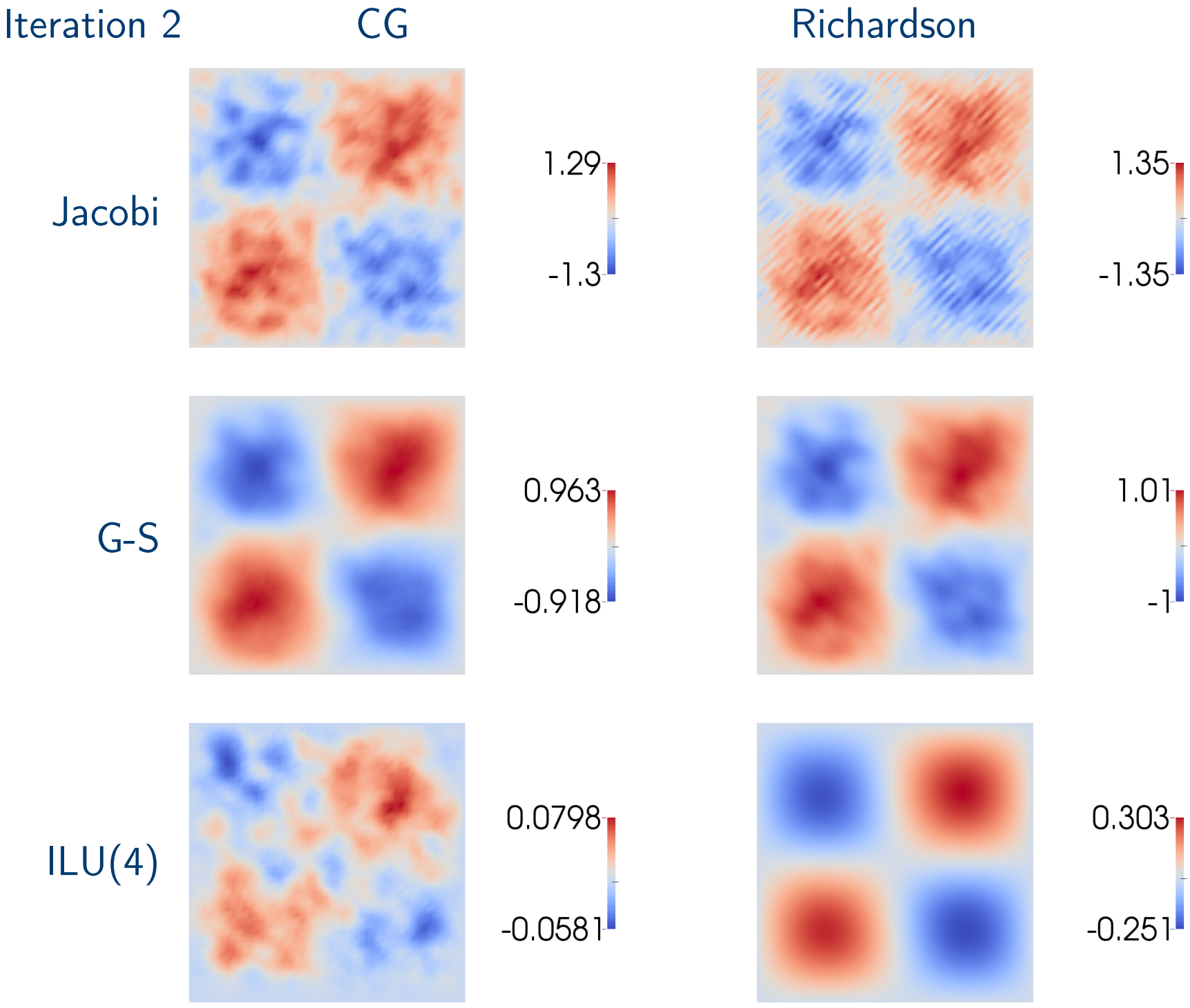


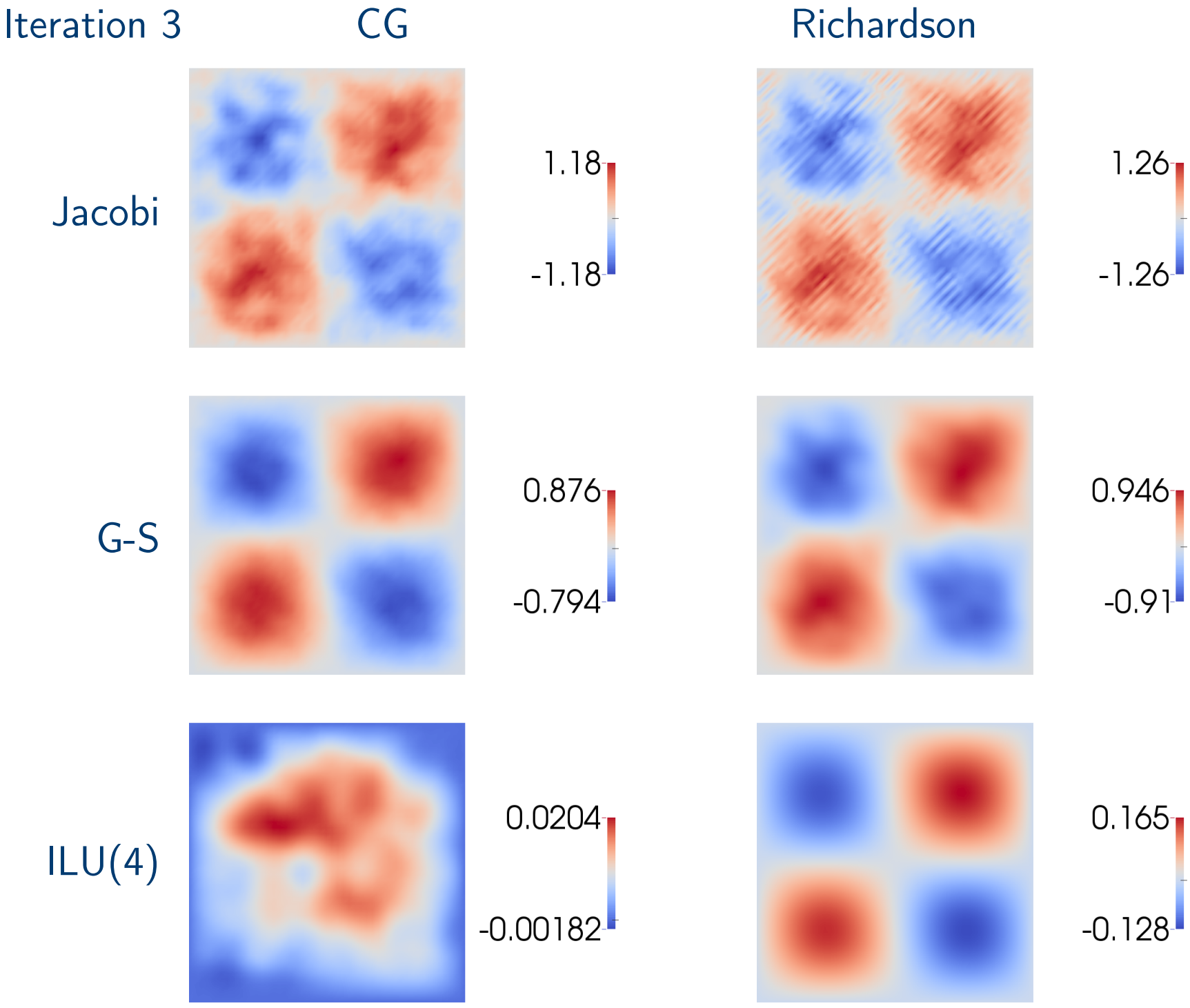
What is going wrong?

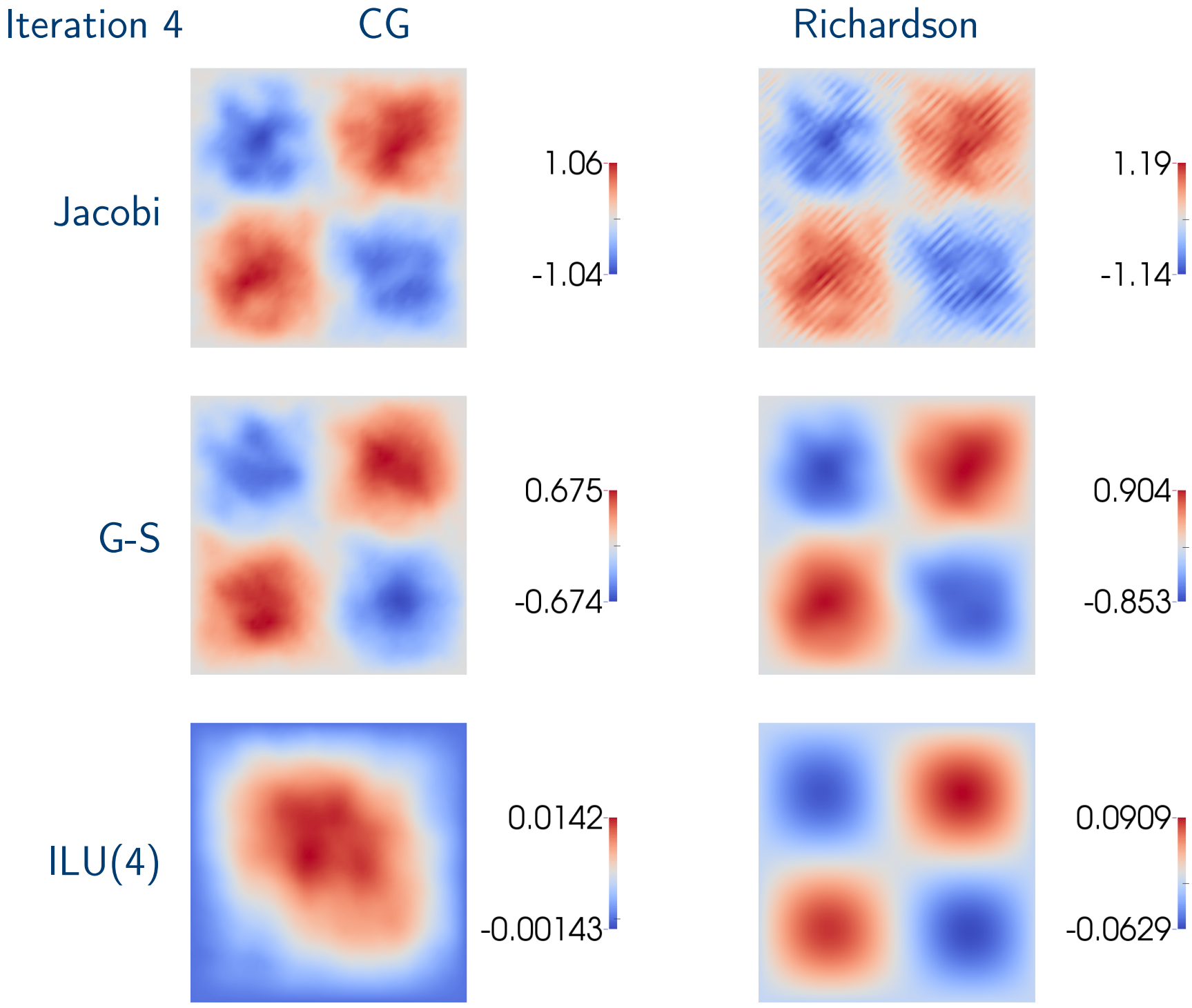
- The asymptotic convergence rate is as expected, can we gain an intuition for why we get this behaviour?
- It's instructive to look at what happens to the error.
- Let's choose the forcing such that $u_* = \sin(\pi x)\sin(\pi y)$
- Initial guess, randomly choose $u_0(x, y) \in [-1, 1]$ satisfying the zero Dirichlet conditions.

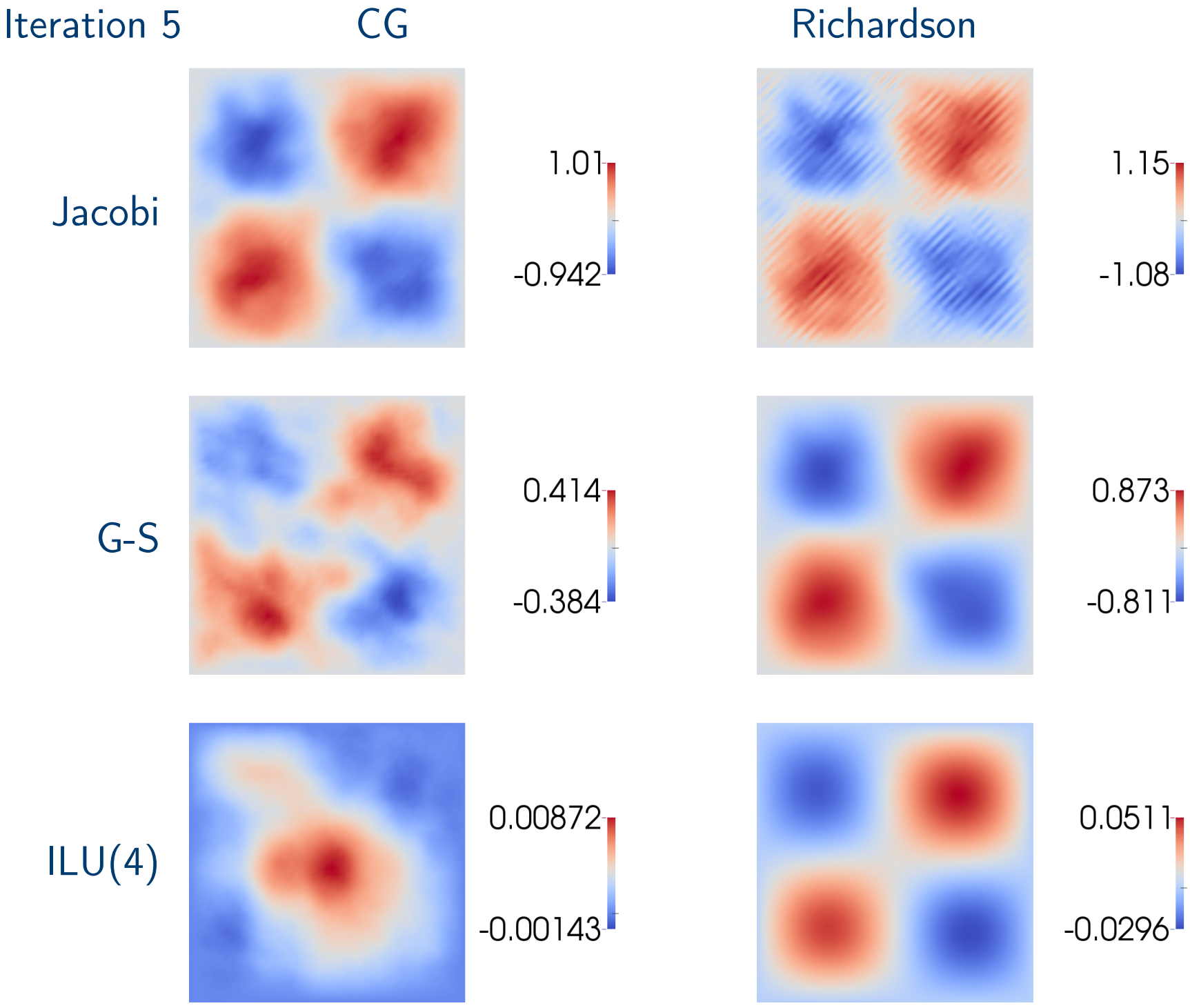

Iteration 0
CG
Richardson
Jacobi
G-S
ILU(4)
1.95
-1.92
1.95
-1.92
1.95
-1.92
1.95
-1.92
1.95
-1.92
1.95
-1.92

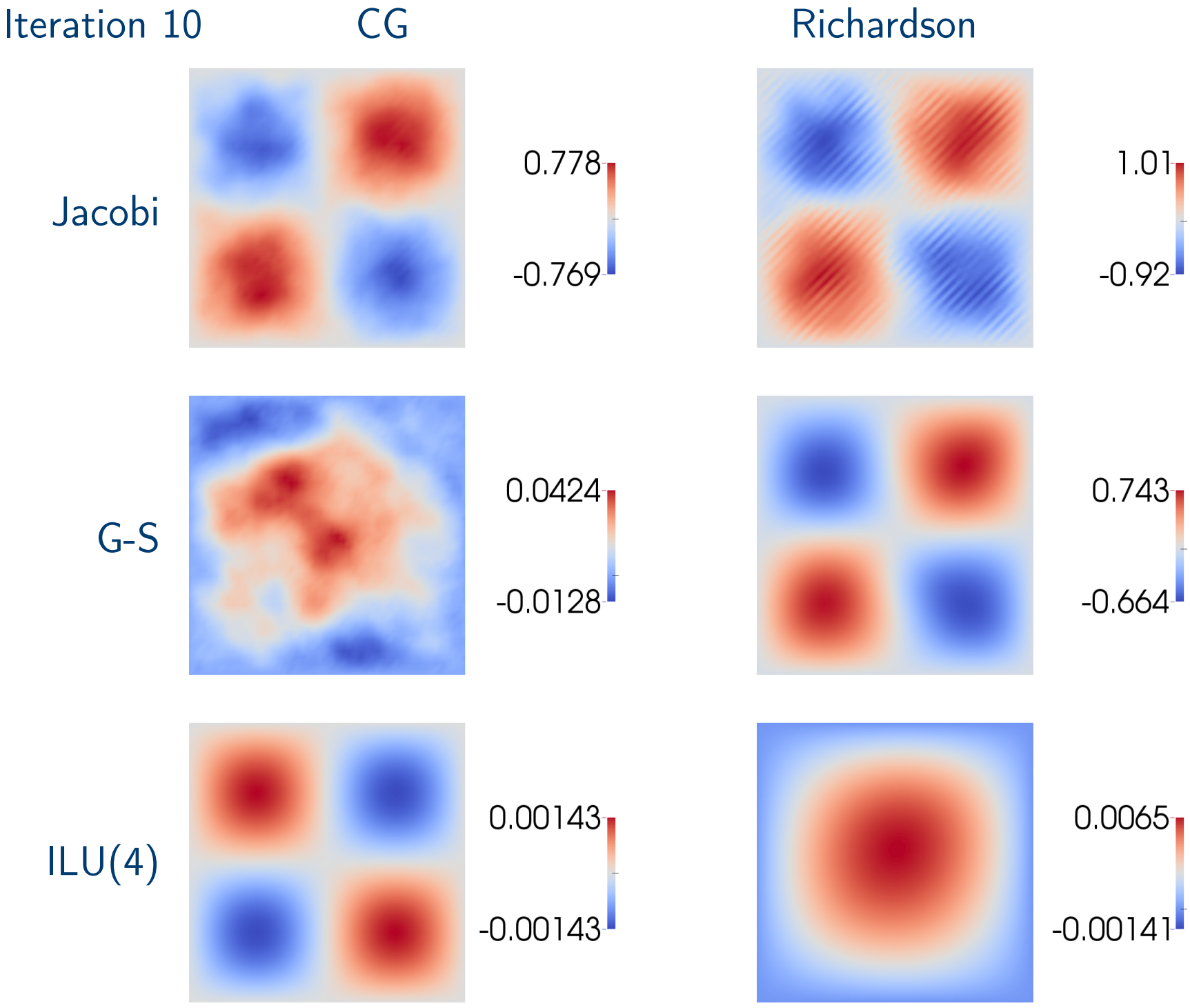

Iteration 1
CG
Richardson
Jacobi
1.5
-1.44
1.48
-1.69
G-S
1.14
-1.15
1.19
-1.2
ILU(4)
0.555
-0.5
0.556
-0.5

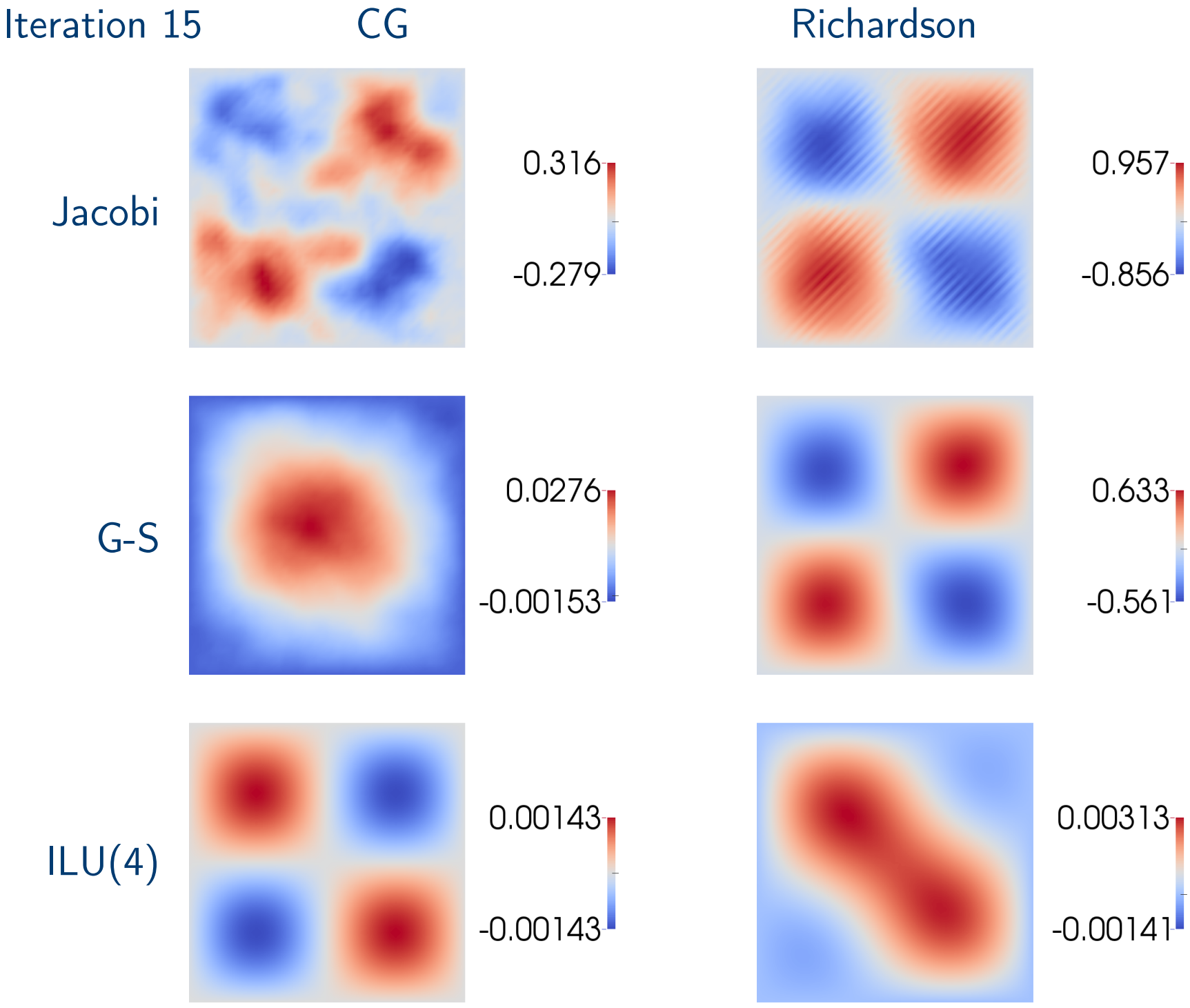

Iteration 2
CG
Richardson
Jacobi
1.29
-1.3
1.35
-1.35
G-S
0.963
-0.918
1.01
-1
ILU(4)
0.0798
-0.0581
0.303
-0.251

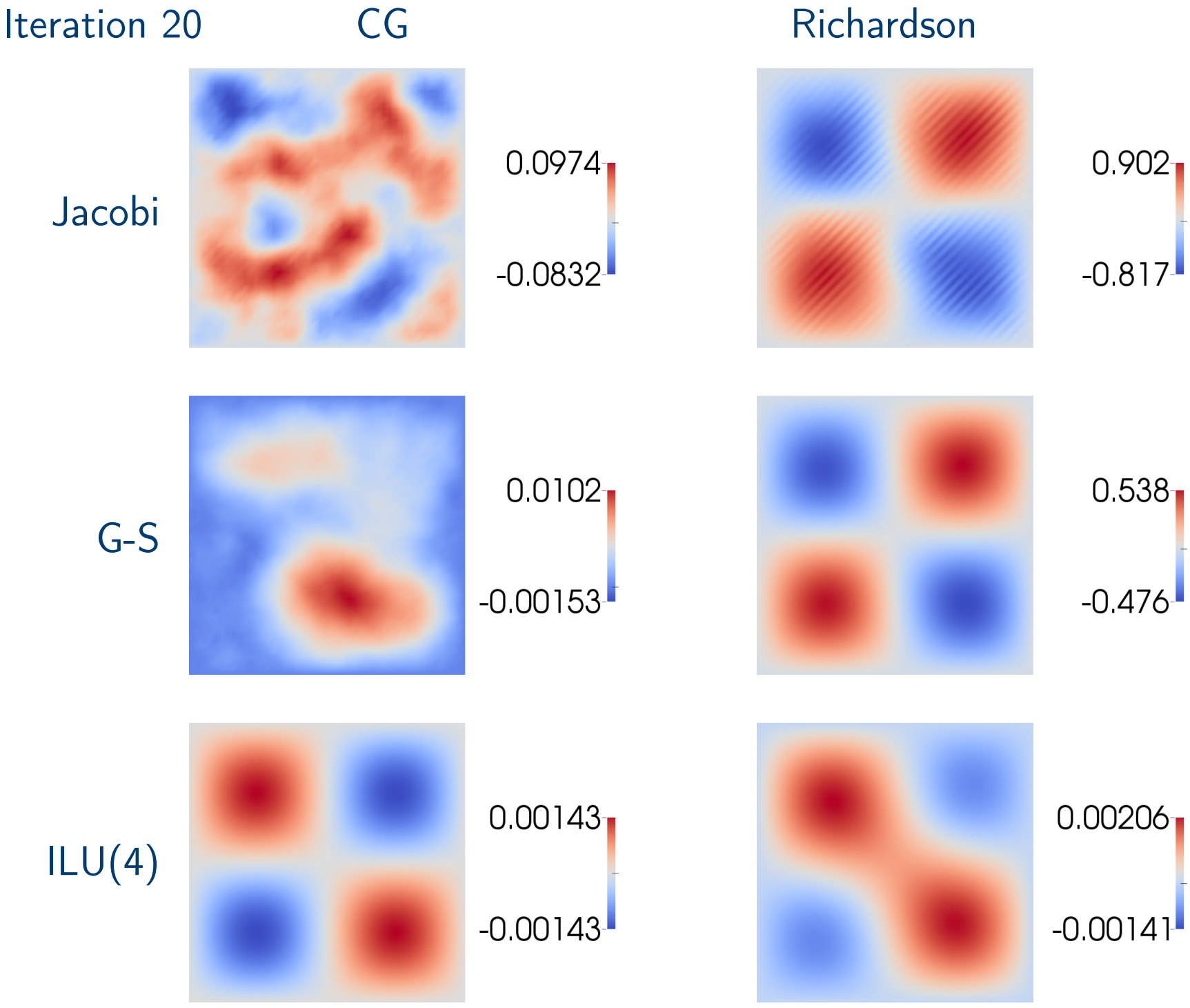

Iteration 3
CG
Richardson
Jacobi
1.18
-1.18
1.26
-1.26
G-S
0.876
-0.794
0.946
-0.91
ILU(4)
0.0204
-0.00182
0.165
-0.128

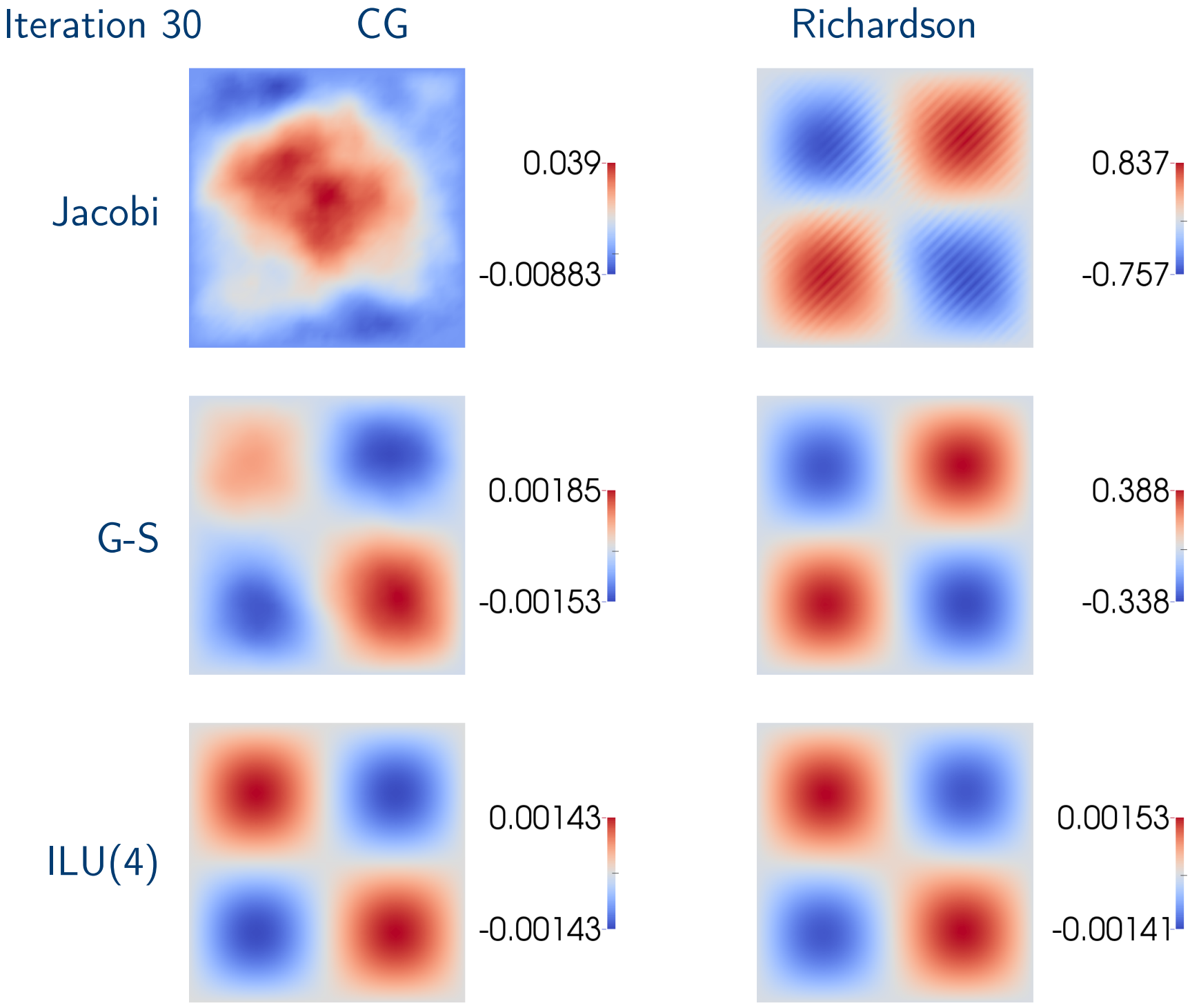

Iteration 4
CG
Richardson
Jacobi
1.06
-1.04
1.19
-1.14
G-S
0.675
-0.674
0.904
-0.853
ILU(4)
0.0142
-0.00143
0.0909
-0.0629

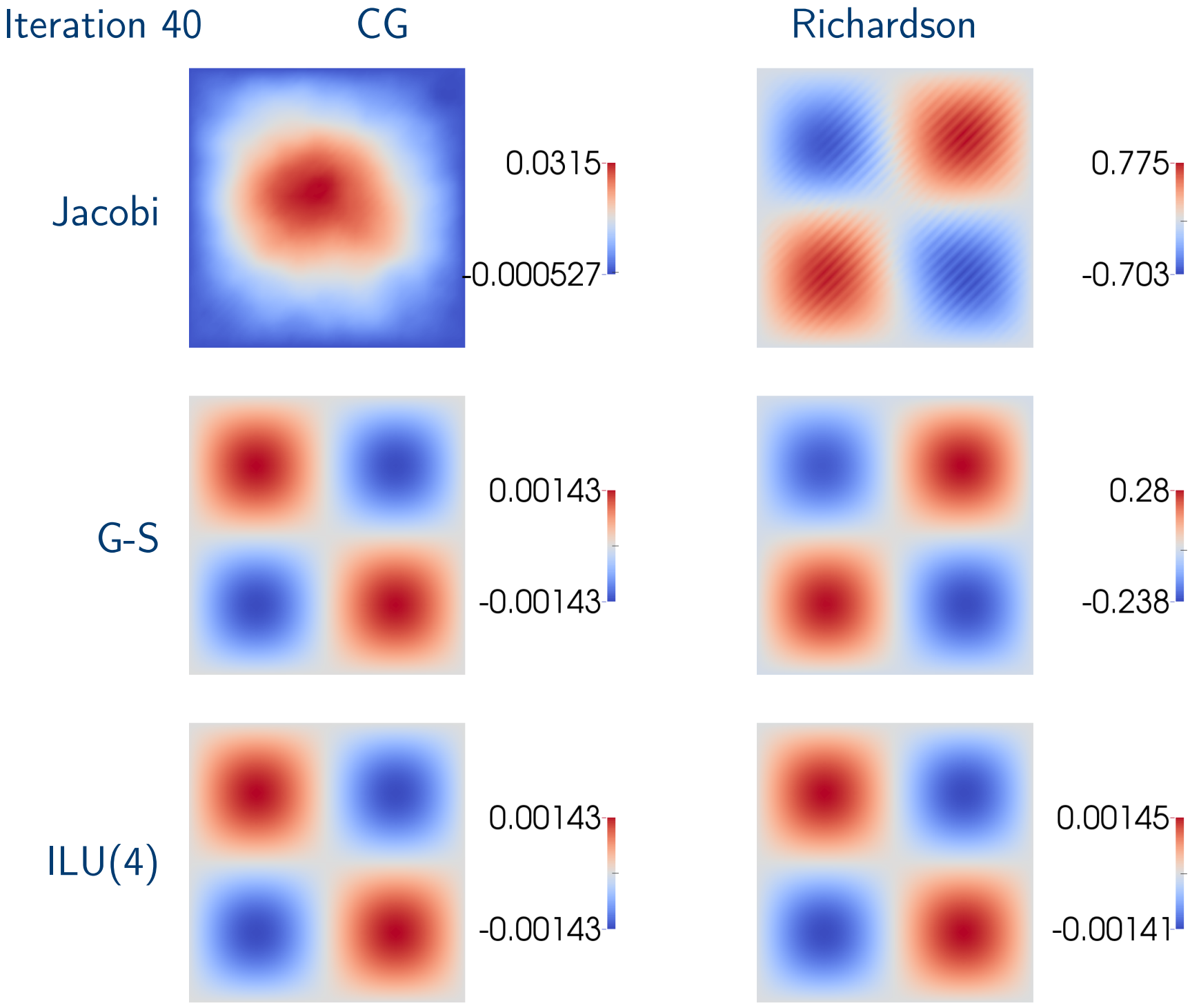

Iteration 5
CG
Richardson
Jacobi
1.01
-0.942
1.15
-1.08
G-S
0.414
-0.384
0.873
-0.811
ILU(4)
0.00872
-0.00143
0.0511
-0.0296


Iteration 10
CG
Richardson
Jacobi
0.778
-0.769
1.01
-0.92
G-S
0.0424
-0.0128
0.743
-0.664
ILU(4)
0.00143
-0.00143
0.0065
-0.00141


Iteration 15
CG
Richardson
Jacobi
0.316
-0.279
0.957
-0.856
G-S
0.0276
-0.00153
0.633
-0.561
ILU(4)
0.00143
-0.00143
0.00313
-0.00141


Iteration 20
CG
Richardson
Jacobi
0.0974
-0.0832
0.902
-0.817
G-S
0.0102
-0.00153
0.538
-0.476
ILU(4)
0.00143
-0.00143
0.00206
-0.00141


Iteration 30
CG
Richardson
Jacobi
0.039
-0.00883
0.837
-0.757
G-S
0.00185
-0.00153
0.388
-0.338
ILU(4)
0.00143
-0.00143
0.00153
-0.00141


Iteration 40
CG
Richardson
Jacobi
0.0315
-0.000527
0.775
-0.703
G-S
0.00143
-0.00143
0.28
-0.238
ILU(4)
0.00143
-0.00143
0.00145
-0.00141

Intuition: what's going on

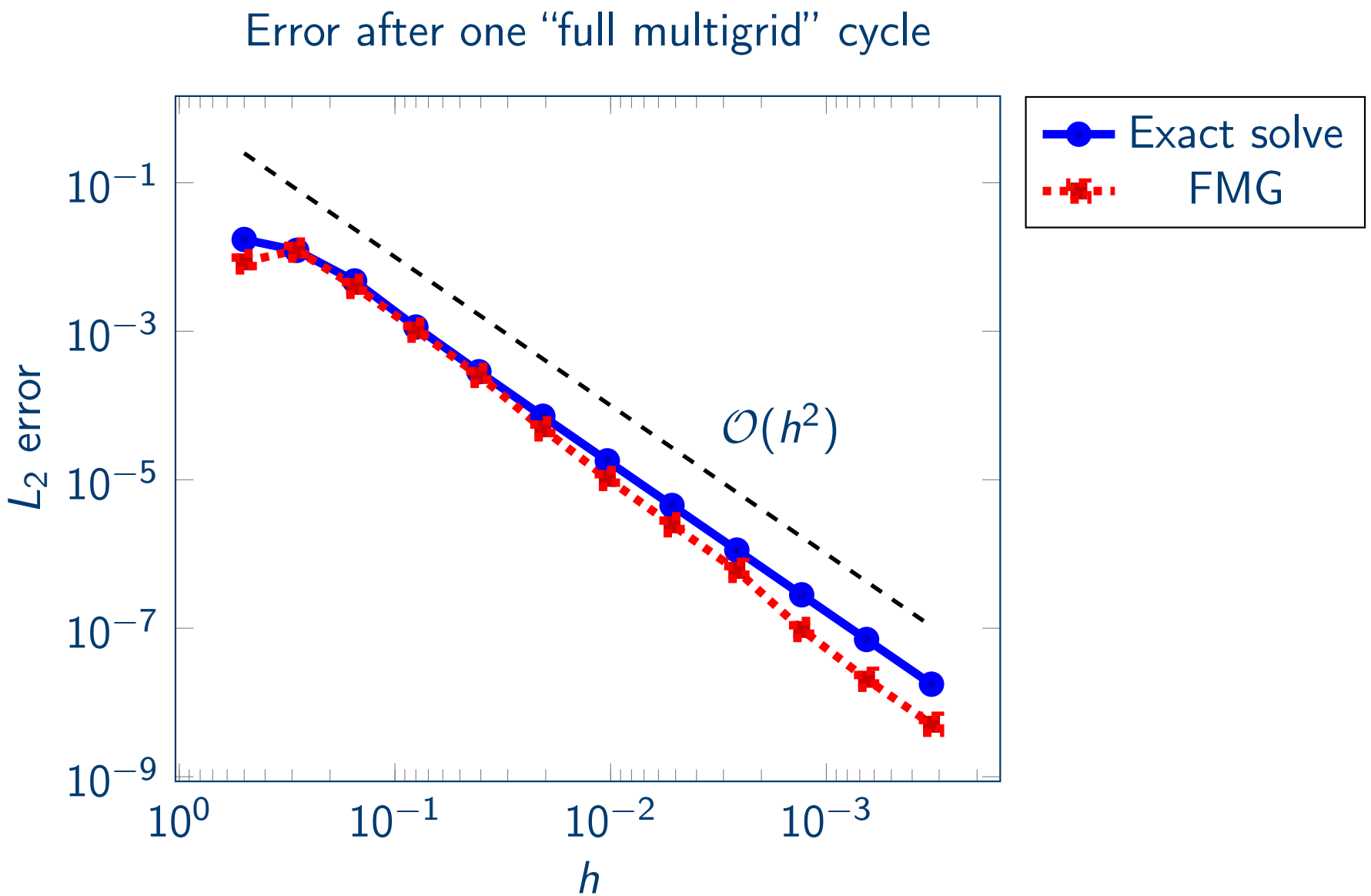
- Poisson problem is globally coupled
- But splitting-based solvers only propagate information locally
- So as we increase the resolution more and more, everything takes longer
- Stationary iterations like Jacobi, G-S, SOR are often called smoothers
- They remove high frequency error very well, but take a long time to damp the low frequency error.

Multilevel methods

- Use a hierarchy of scales
- Use cheap smoothers to get a smooth error
- Move to a coarser grid (where the error looks rough again)
- Rinse and repeat

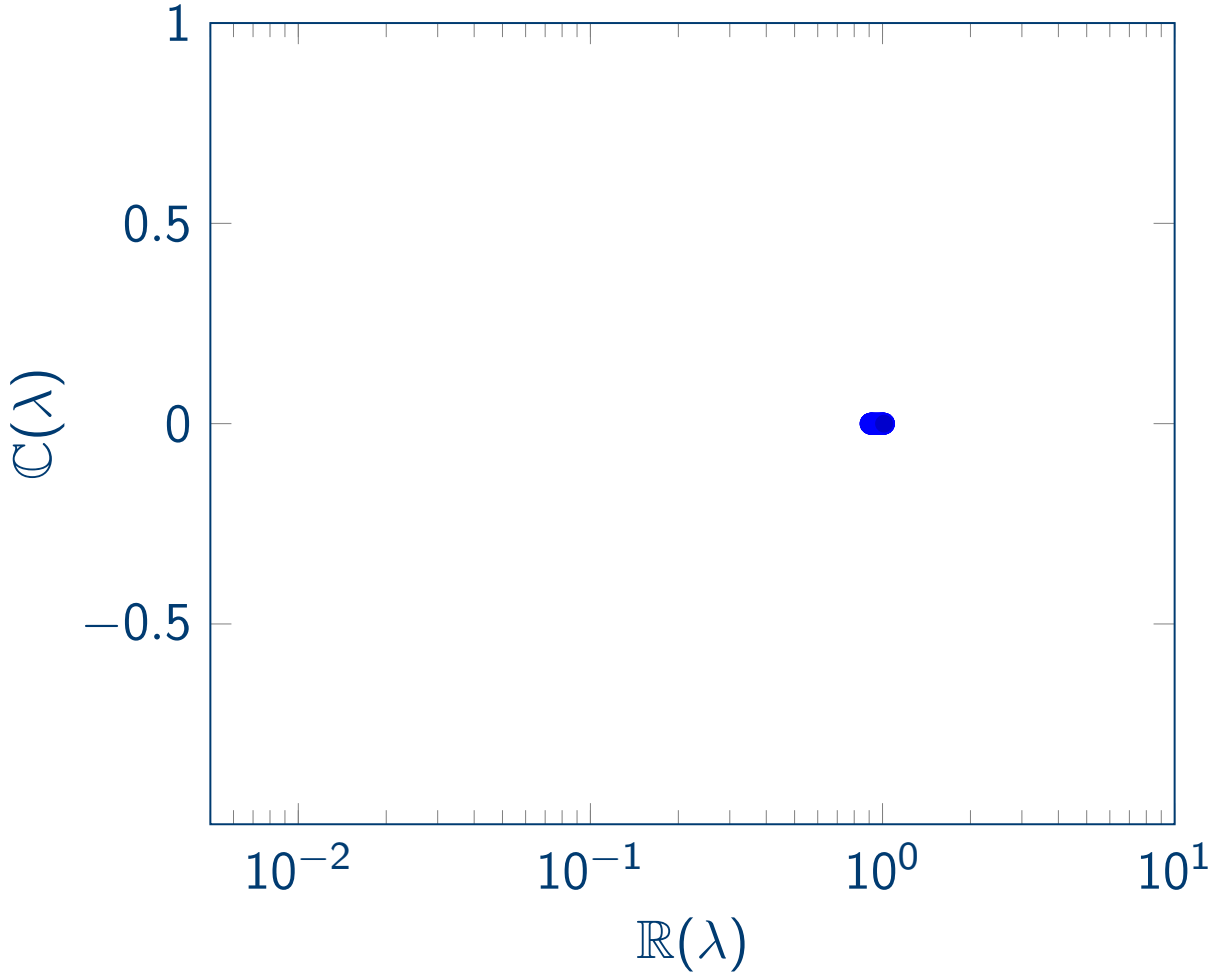
Optimality

Multigrid methods can be algorithmically optimal. Requiring $\mathcal{O}(N_{\text{dof}})$ work to reduce the error to within discretisation error.
If you're interested in this, the classic text on multigrid is Brandt 1977, but there is a huge literature on this.

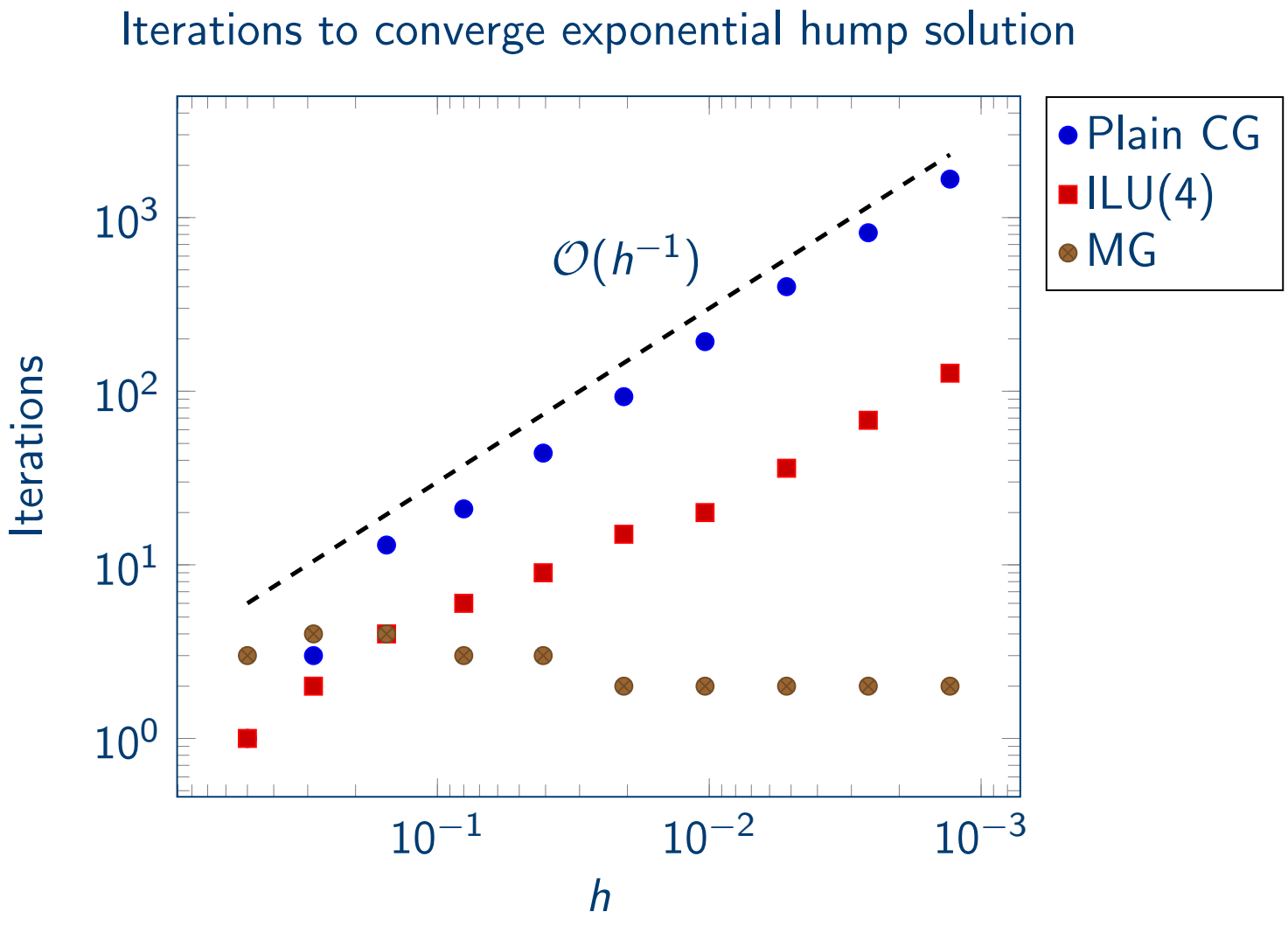

Error after one "full multigrid" cycle
Exact solve
FMG
$\mathcal{O}(h^2)$
L_2 error
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^{0}
10^{-1}
10^{-2}
10^{-3}
h

Our favourite spectrum

Eigenvalues of ∇^2 preconditioned by multigrid V-cycle, $\kappa = 1.11$



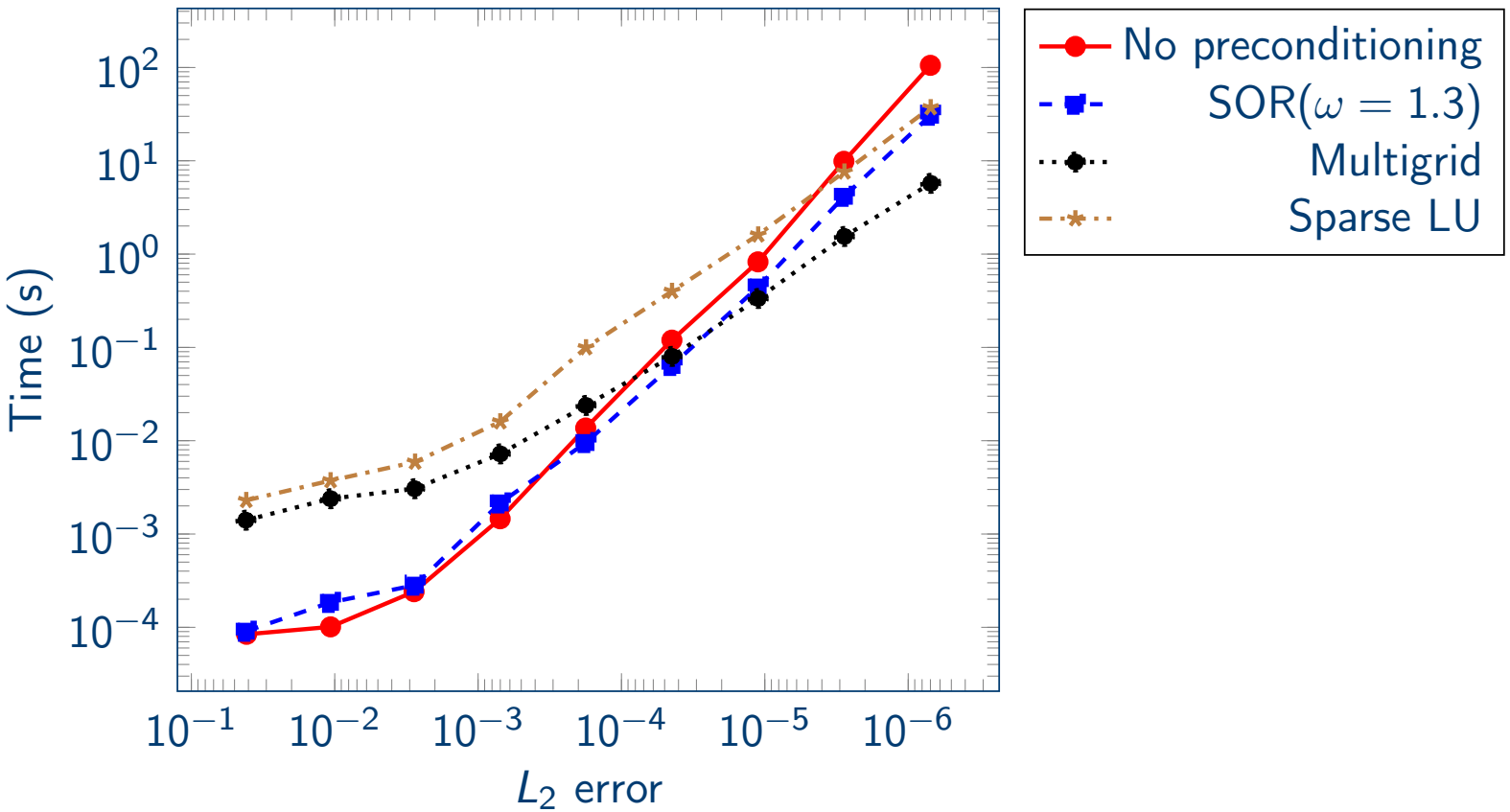
And convergence



What is best for me?

Work precision diagrams

To judge which method to use for your problem, it is important to consider the regime you're interested in. Work precision diagrams are useful for this.



Use libraries in your software

Maxim

The most important part of programming is knowing when not to write your own code.

- You should not, except maybe for interest, implement all these iterative methods (and preconditioners) yourself!
- There are many high-quality libraries available. Pick one, and use it.
- I used PETSc to develop the example that produced the results in these slides.

My operator isn't SPD

What to do

- CG (Hestenes and Stiefel 1952) minimises the $A-$norm of the error. If A is not SPD, it doesn't define a norm, so we can't do that
- Instead, minimise $2-$norm of residual: $||b - Ax||_2$.
- If A is symmetric (but indefinite), use MINRES or SYMMLQ (Paige and Saunders 1975).
- If A is not symmetric, probably use GMRES (Saad and Schultz 1986).

Asymmetry makes everything worse

- MINRES uses short recurrences and, like CG, uses bounded memory
- GMRES needs to reorthogonalise the current subspace at every step, therefore memory use grows with iteration count.
- Other non-symmetric methods (BICGSTAB, CGS, ...) have worse convergence properties (or no guarantees).

GMRES issues

- Convergence very operator-dependent, see, for example Nachtigal, Reddy, and Trefethen (1992) and Greenbaum, Pták, and Strakoš (1996).
- Restarted GMRES makes things more complex, see Embree (2003) for a nice review.
- Much harder to find good preconditioners for non-symmetric systems.

What about multiple variables?

- Often, we need to solve a problem of more than one variable. Stokes, Navier-Stokes, Cahn-Hilliard, MHD, combinations thereof.
- "black-box" preconditioning is even less likely to work than for single-variable systems.
- Many of the state-of-the art preconditioners for such problems rely on block factorisations of the operator.

Block systems I

Theorem

If a block matrix

$$\mathcal{A} = \begin{pmatrix} A & B^T \\ C & 0 \end{pmatrix}$$

is preconditioned by

$$\mathcal{P} = \begin{pmatrix} A & 0 \\ 0 & CA^{-1}B^T \end{pmatrix}$$

then the preconditioned matrix $\mathcal{P}^{-1}\mathcal{A}$ has at most four distinct eigenvalues.

Block systems II

Murphy, Golub, and A. J. Wathen (2000).

Writing

$$\mathcal{T} = \begin{pmatrix} I & A^{-1}B^T \\ (CA^{-1}B^T)^{-1}C & 0 \end{pmatrix},$$

then

$$(\mathcal{T} - I/2)^2 = \begin{pmatrix} I/4 + A^{-1}B^T(CA^{-1}B^T)^{-1}C & 0 \\ 0 & I/4 \end{pmatrix}.$$

Since $A^{-1}B^T(CA^{-1}B^T)^{-1}C$ is a projection

$$\left[(\mathcal{T} - I/2)^2 - I/4\right]^2 = (\mathcal{T} - I/2)^2 - I/4$$

and so

$$\mathcal{T}(\mathcal{T} - I)(\mathcal{T}^2 - T - I) = 0.$$

□

Some pointers

- Saad (2003) is good on stationary and Krylov iterations.
- Benzi, Golub, and Liesen (2005) is quite exhaustive on saddle point systems $\begin{pmatrix} A & B_1^T \\ B_2 & -C \end{pmatrix}$.
- A. J. Wathen (2015) is a recent (gentle) review article.
- Elman, Silvester, and A. Wathen (2014) covers saddle point solvers in the context of fluid dynamics.
- Kirby (2010) and Mardal and Winther (2011) present an interesting approach to designing preconditioners based on ideas from functional analysis.

Questions?

References I

- Benzi, M., G. H. Golub, and J. Liesen (2005). "Numerical solution of saddle point problems". In: Acta Numerica 14, pp. 1–137. DOI: 10.1017/S0962492904000212.
- Brandt, A. (1977). "Multi-level adaptive solutions to boundary-value problems". In: Mathematics of Computation 31.138, pp. 333–390.
- Elman, H., D. Silvester, and A. Wathen (2014). Finite elements and fast iterative solvers. Second edition. Oxford University Press.
- Embree, M. (2003). "The Tortoise and the Hare: Restart GMRES". In: SIAM Review 45.2, pp. 259–266. DOI: 10.1137/S00361445013961.
- Greenbaum, A., V. Pták, and Z. Strakoš (1996). "Any Nonincreasing Convergence Curve is Possible for GMRES". In: SIAM Journal on Matrix Analysis and Applications 17.3, pp. 465–469. DOI: 10.1137/S0895479894275030.

References II

- Hestenes, M. R. and E. Stiefel (1952). "Methods of conjugate gradients for solving linear systems". In: Journal of Research of the National Bureau of Standards 49.6, pp. 409–436.
- Kirby, R. C. (2010). "From Functional Analysis to Iterative Methods". In: SIAM Review 52.2, pp. 269–293. DOI: 10.1137/070706914.
- Málek, J. and Z. Strakoš (2014). Preconditioning and the Conjugate Gradient Method in the Context of Solving PDEs. Philadelphia, PA: Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9781611973846.
- Mardal, K.-A. and R. Winther (2011). "Preconditioning discretizations of systems of partial differential equations". In: Numerical Linear Algebra with Applications 18.1, pp. 1–40. DOI: 10.1002/nla.716.

References III

- Murphy, M. F., G. H. Golub, and A. J. Wathen (2000). "A Note on Preconditioning for Indefinite Linear Systems". In: SIAM Journal on Scientific Computing 21.6, pp. 1969–1972. DOI: 10.1137/S1064827599355153.
- Nachtigal, N., S. Reddy, and L. Trefethen (1992). "How fast are nonsymmetric matrix iterations?" In: SIAM Journal on Matrix Analysis and ... 13.3, pp. 778–795. DOI: 10.1137/0613049.
- Paige, C. C. and M. A. Saunders (1975). "Solution of sparse indefinite systems of linear equations". In: SIAM Journal on Numerical Analysis 12.4, pp. 617–629. DOI: 10.1137/0712047.
- Saad, Y. (2003). Iterative Methods for Sparse Linear Systems. Second edition. Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9780898718003.

References IV

- Saad, Y. and M. H. Schultz (1986). "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems". In: SIAM Journal on Scientific and Statistical Computing 7.3, pp. 856–869. DOI: 10.1137/0907058.
- Wathen, A. J. (2015). "Preconditioning". In: Acta Numerica 24, pp. 329–376. DOI: 10.1017/S0962492915000021.