Two Iterative Solvers for a Nonlinear Elliptic PDE

Jerry Lingjie Mei*
Nolan Reilly†

May 20, 2018

Abstract

We solve nonlinear PDE $\nabla \cdot [(1+\epsilon u)\nabla u] = f$ with periodic boundary conditions by means of two separate fixed-point iterations, $\nabla^2 u_n = \frac{f-\epsilon\nabla u_n \cdot \nabla u_n}{1+\epsilon u_n}$ and $\nabla \cdot [(1+\epsilon u_n)\nabla u_{n+1}] = f$. The periodic structure of the problem allows us to very efficiently solve the Poisson equations of the first iteration, making this iteration converge multiple times faster than its competitor, which requires a linear system inversion at each step. We implement two acceleration strategies to improve the convergence rates of these two iterations: geometric multigrid method and Anderson acceleration. We discover both of these to give significant, but not overwhelming, speed gains on this problem.

1 Introduction

The Poisson equation is by far the best-studied elliptic PDE for numerical solution, as it is simple and linear. However, many physical processes are nonlinear in character, from stress distribution in solids to gravitation in General Relativity, and these give rise in their steady states to *nonlinear* elliptic equations. We take it upon ourselves to explore iterative schemes to solve such a nonlinear elliptic PDE.

We will consider the following nonlinear elliptic PDE, which displays tunable nonlinearity while still remaining relatively simple:

$$\begin{aligned} \nabla \cdot [(1+\varepsilon u)\nabla u] &= f \\ u &: [0,1]^2 \to \mathbb{R} \\ u &\text{ periodic} \end{aligned}$$

This nonlinear PDE can be interpreted as adding a non-linear term onto the Poisson equation, and we are going to assume it is real-valued and stays on a periodic domain.

2 Basic Tools

2.1 Iterative Schemes

As this kind of PDE is impossible except in extraordinary cases to solve analytically, we have to use some iterative solver to calculate the function f until it is convergent. We will implement two fixed-point solvers:

*Class of 2020, Course 18 and 6-3, Massachusetts Institute of Technology
†Class of 2021, Course 18, Massachusetts Institute of Technology

- Using the vector identity

$$\nabla \cdot (\varphi A) = \varphi \nabla \cdot A + A \cdot \nabla \varphi$$

The condition can be translated into

$$(1 + \varepsilon u)\nabla^2 u + \varepsilon(\nabla u \cdot \nabla u) = f$$

This suggests the following iteration:

$$\nabla^2 u_{n+1} = \frac{f - \varepsilon(\nabla u_n \cdot \nabla u_n)}{1 + \varepsilon u_n}$$

- The second iteration arises directly from the initial problem formulation:

$$\nabla \cdot [(1 + \varepsilon u_n)\nabla u_{n+1}] = f$$

2.2 Discretization

We will solve the PDE on a $N \times N$ grid, where N is a power of 2. To calculate the gradients and Laplacian on a grid, we employ spectral differentiation using the FFT. Let $\mathcal{F}$ be a 2-dimensional DFT on a $N \times N$ grid, and $u \xrightarrow{\mathcal{F}} \hat{u}$, then

$$u_x \xrightarrow{\mathcal{F}} 2\pi i k \hat{u} \xrightarrow{\mathcal{F}^{-1}} u_x.$$

Similarly,

$$\nabla^2 u \xrightarrow{\mathcal{F}} (2\pi i)^2 (k^2 + l^2)\hat{u}$$

$$\hat{f}/(2\pi i)^2 (k^2 + l^2) \xrightarrow{\mathcal{F}^{-1}} u$$

Both of these procedures take $O(N^2 \log N)$ time to compute.

3 The First Iteration

3.1 Naive Execution

The naive execution is just combining discretization and the first iteration:

Algorithm 1: Naive First Iteration

```
n = 0, u_0 = 0;
while ||u_n − u_{n+1}||_F ≤ τ||f||_F do
    Calculate the gradient of u_n using DFT;
    Calculate the right hand side of the iteration: R_n = (f − ε(∇u_n · ∇u_n)) / (1 + εu_n);
    Solve the Poisson equation ∇²u_{n+1} = R_n using DFT;
```

3.2 Multigrid Method

The Geometric multigrid method[3][6] can be designed to accelerate the convergence of iterative PDE solvers, including our first iteration. There are two essential flavors of multigrid: Krylov-subspace multigrid and smoothing multigrid. Krylov subspace multigrid uses the coarser solutions from coarser grids to precondition finer-grid

solutions, using the much cheaper inversion on a smaller grid as an approximate inverse. Smoothing multigrid was the algorithm historically first developed; it observes that most fixed-point-based iterative methods – like Jacobi iteration or Gauss-Seidel – reduce error components by a linear factor every iteration which is much smaller for high-frequency error than for low-frequency error, and exploits this by iterating them on multiple scales to decrease all frequencies of error at the highest rate [5]. We implemented smoothing multigrid applied directly to our first fixed-point iteration.

To transfer the data between grids, two operators are introduced: restriction (to coarsen a solution) and interpolation (to refine it). The restriction operator is a weighted average of its nearby nodes in a finer grid. In the 2D case, it can be simplified as follows:

Algorithm 2: Restriction from a finer grid f to a coarser grid c

for $i = 1 : N$ **do**
  for $j = 1 : N$ **do**
    $c_{i,j} = \frac{1}{16}c_{2i-1,2j-1} + \frac{1}{8}c_{2i-1,2j} + \frac{1}{16}c_{2i-1,2j+1} + \frac{1}{8}c_{2i,2j-1} + \frac{1}{4}c_{2i,2j} + \frac{1}{8}c_{2i,2j+1} + \frac{1}{16}c_{2i+1,2j-1} + \frac{1}{8}c_{2i+1,2j} + \frac{1}{16}c_{2i+1,2j+1}$;

The interpolation operator is a simple linear interpolation function. It is worth noting that the interpolation and restriction operators, when viewed as matrices, are in fact scaled transposes of one another.

Algorithm 3: Interpolation from a coarser grid c to a finer grid f

for $i = 1 : N$ **do**
  for $j = 1 : N$ **do**
    $f_{2i,2j} = c_{i,j}$;
    $f_{2i,2j+1} = \frac{1}{2}c_{i,j} + \frac{1}{2}c_{i,j+1}$;
    $f_{2i+1,2j} = \frac{1}{2}c_{i,j} + \frac{1}{2}c_{i+1,j}$;
    $f_{2i+1,2j+1} = \frac{1}{4}c_{i,j} + \frac{1}{4}c_{i+1,j} + \frac{1}{4}c_{i,j+1} + \frac{1}{4}c_{i+1,j+1}$;

To put iterative solvers into the multigrid system, cycles are introduced to represent the order to iterate through different grids. The iterative solvers are only called a fixed number of times unrelated to the size of the grid before visiting another grid. Since the computational cost on a coarser grid is smaller than a finer grid, all different types of cycles try to visit and iterate on the coarsest grid the most number of times. These are the three most typical types of cycles: the V cycle, the W cycle and the so-called Full Multigrid cycle, illustrated in the following graph (note that the nodes on the top represents the finest grid):

Figure 1: V cycle, W cycle and full cycle

In our implementation, we choose to use full multigrid, as it has the least time complexity for the same number of iterations. In formal definition, the full cycle can written like the following:

Algorithm 4: Full cycle

Iterate on Grid 4 for *maxiter* times and interpolate to Grid 3;
Iterate on Grid 3 for *maxiter* times and restrict to Grid 4;
Iterate on Grid 4 for *maxiter* times and interpolate to Grid 3;
Iterate on Grid 3 for *maxiter* times and interpolate to Grid 2;
Iterate on Grid 2 for *maxiter* times and restrict to Grid 3;
Iterate on Grid 3 for *maxiter* times and restrict to Grid 4;
Iterate on Grid 4 for *maxiter* times and interpolate to Grid 3;
Iterate on Grid 3 for *maxiter* times and interpolate to Grid 2;
Iterate on Grid 2 for *maxiter* times and interpolate to Grid 1;
while *Grid 1 can still be improved* **do**
    Iterate on Grid 1 for *maxiter* times and restrict to Grid 2;
    Iterate on Grid 2 for *maxiter* times and restrict to Grid 3;
    Iterate on Grid 3 for *maxiter* times and restrict to Grid 4;
    Iterate on Grid 4 for *maxiter* times and interpolate to Grid 3;
    Iterate on Grid 3 for *maxiter* times and interpolate to Grid 2;
    Iterate on Grid 2 for *maxiter* times and interpolate to Grid 1;

Despite the fact that multigrid converges in constant iterations, and have the same time complexity as the iterative solver itself, multigrid produces an error related to its mesh size. Taking that into notice, we only use multigrid as an initial guess for the first iteration, and it still works really well.

Algorithm 5: First iteration with multigrid full cycle

$n = 0, u_0 = 0$;
Prepare four grids for Full Cycle;
Run the full cycle, running the first iteration at most four times each level;
while $\|u_n - u_{n+1}\|_F \leq \tau \|f\|_F$ **do**
    Calculate the gradient of u_n using DFT;
    Calculate the right hand side of the iteration: $R_n = \frac{f - \varepsilon(\nabla u_n \cdot \nabla u_n)}{1 + \varepsilon u_n}$;
    Solve the Poisson equation $\nabla^2 u_{n+1} = R_n$ using DFT;

3.3 Numerical simulations

We simulate the first iteration on $N = 256, 512, 1024, 2048, 4096$ problems, as well as trying to accelerate it using multigrid. The toy problem we consider is

$$0.6\sin(2\pi x) + 0.2\sin(4\pi y) + 0.1\sin(10\pi x + 8\pi y)$$

with $\varepsilon = 1$, and both iterations stop when the relative error is $\leq 1 \times 10^{-6}$. At most 4 iterations are used in one visit to a single grid.

Problem	Iterations on $N \times N$ grid	Running time (s)
$N = 256$	29	0.214
$N = 512$	29	1.14
$N = 1024$	29	6.29
$N = 2048$	29	38.5
$N = 4096$	29	187
$N = 256$ with multigrid	24	0.189
$N = 512$ with multigrid	20	0.888
$N = 1024$ with multigrid	20	4.87
$N = 2048$ with multigrid	16	23.1
$N = 4096$ with multigrid	16	120

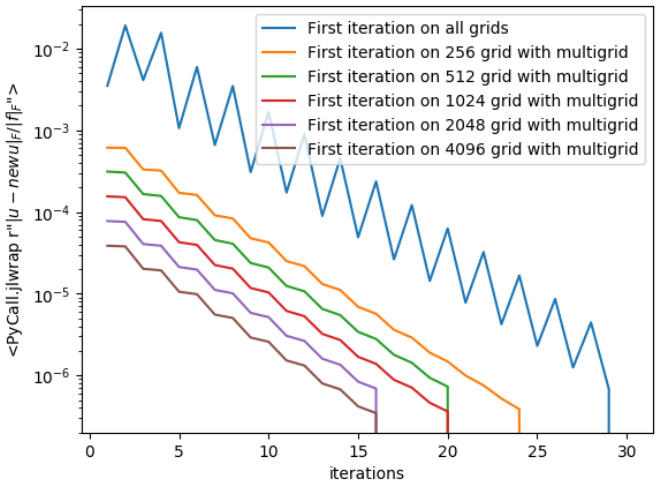

Figure 2: The decaying of error w.r.t. iterations on the coarsest grid

The simulation results shows that both algorithm converges, and the iterative solver equipped with multigrid may save up to 40% time and top-level iterations, which states the effectiveness of using multigrid as an initial guess. We may also find that with the larger N is, the better the initial guess is, this is because multigrid is much better at converging at low frequencies than the naive iteration[2].

4 The Second Iteration

4.1 Naive execution

At its core, the second iteration is extremely simple:

Algorithm 6: Second Iteration

while $\|\nabla \cdot [(1 + \epsilon u_n)\nabla u_n] - f\| > \tau$ **do**
  Solve $\nabla \cdot [(1 + \epsilon u_n)\nabla]\, u_{n+1} = f$

However, despite being in appearance simpler than the first, this iteration is surprisingly nontrivial, and in fact is substantially slower than its competitor. The reason for this is that, unlike for the first iteration, there is no $O(N^2 \log N)$ closed-form solution for the linear system; it must be solved by an iterative process itself, as, when using the FFT, differentiation leads to a dense matrix far too large to be inverted. As the

linear operator is not Hermitian Positive Definite, either, we cannot use Conjugate Gradient to solve it. For stability purposes, then, we employ GMRES. Unfortunately, the Laplacian has more or less evenly spaced eigenvalues over the entire negative real line, so the convergence of GMRES on it is fairly abysmal for a system this size:

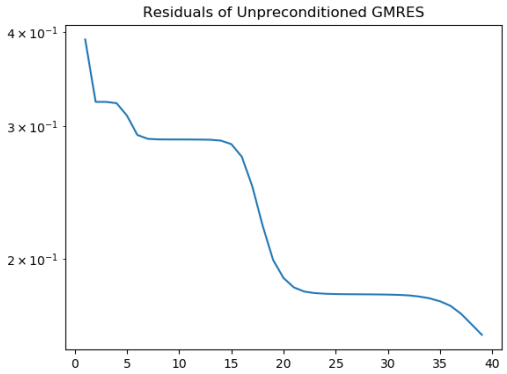

Figure 3: Convergence of unpreconditioned GMRES on $\nabla \cdot [(1 + \epsilon u_n)\nabla]$

4.2 Jacobian preconditioning

To solve the problem, we exploit the fact that u has only relatively low frequencies, such that in the Fourier basis, $\nabla \cdot [(1 + \epsilon u_n)\nabla]$ is very diagonally dominant, captured predominantly by the $4\pi^2(k^2 + \ell^2)$ from the Laplacian. Thus, we apply a Jacobi preconditioner – the diagonal of the matrix – in Fourier space, which corresponds to approximating this modified second derivative with the Laplacian. This turns out to be extremely successful in preconditioning this system, with the added benefit that the Laplacian is very easy to invert in this basis:

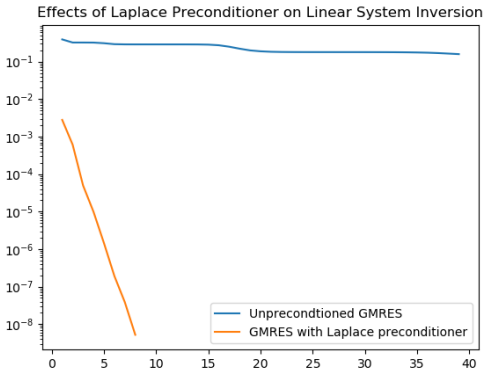

Figure 4: Comparison of convergence of GMRES on $\nabla \cdot [(1 + \epsilon u_n)\nabla]$ in preconditioned and un-preconditioned cases

Problem	Iterations on $N \times N$ grid	Running time (s)
$N = 256$	9	2.03
$N = 512$	9	7.48
$N = 1024$	9	30.9
$N = 2048$	9	145
$N = 4096$	9	729

Table 1: Performance of Naive second iteration

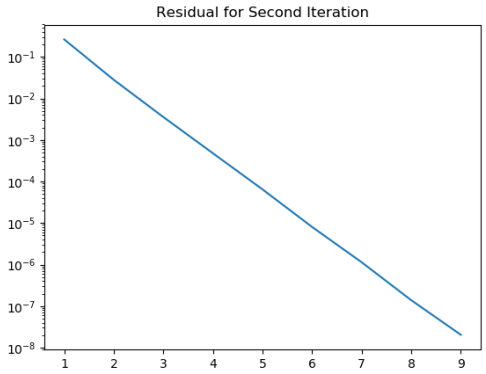

Figure 5: Residuals of second iteration with preconditioning

4.3 Anderson Acceleration

As you can see, this fixed-point iteration is quite well-behaved, with the error decreasing at a linear rate of approximately one significant digit per iteration. All the error measures in this paper are of backwards error, although forwards error are bounded by backwards error with a constant factor. Furthermore, the error is dominated by the fixed point error, as opposed to the error due to approximate linear solution or discretisation, as the error stays constant as the mesh is refined. However, the runtime is somewhat concerning, especially for larger grids. To attempt to reduce the runtime, we implemented Anderson acceleration[1].

Anderson acceleration seeks to find the solution to a fixed-point iteration in the span of previous iterates. In particular, it finds the coefficients that minimize the weighted sum of the previous k residuals with the restriction that the coefficients must sum to 1, and then uses combines the previous k iteration vectors with these vectors to produce the next guess[4].

Algorithm 7: Anderson Acceleration

1 **while** $\|x_{n+1} - x_n\| > \tau$ **do**

2 Find α for $\min_{\alpha} \| \sum_{i=0}^{k-1} \alpha_i (x_{n-i+1} - x_{n-i}) \|$;

3 $x_n = \sum_{i=0}^{k-1} \alpha_i x_{n-i}$

The essential approximation made by Anderson acceleration is that the fixed-point

function f is linear, such that

$$\sum_{i=0}^{k-1} \alpha_i (x_{n-i+1} - x_{n-i}) = \sum_{i=0}^{k-1} \alpha_i (f(x_{n-i}) - x_{n-i}) \approx 0$$

$$\implies f\left(\sum_{i=0}^{k-1} \alpha_i x_{n-i}\right) - \sum_{i=0}^{k-1} \alpha_i x_{n-i} \approx 0.$$

However, this procedure can also be viewed as attempting to extrapolate the zero of the function $f(x) - x$ as a linear combination of previous iterations, and as such a generalization of the secant method. And in fact, Anderson acceleration does yield superlinear convergence: see Figure 6. It is worth noting that both iterations plateau at

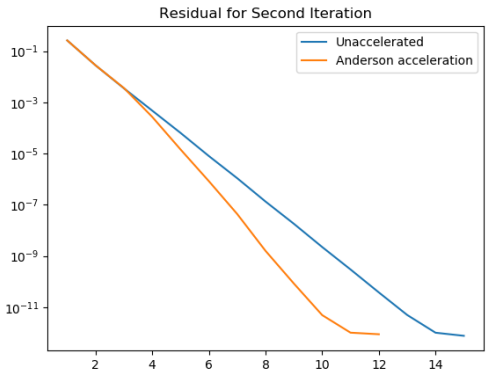

Figure 6: Residuals of second fixed-point iteration with and without Anderson acceleration.

an accuracy of about 10^{-12}; this appears to be due to floating-point error, as the residual of the function shows uncoordinated noise of order 10^{-15}.

Table 2: Performance of Second Iteration with Anderson Acceleration

Problem	Iterations on $N \times N$ grid	Running time (s)
$N = 256$	7	1.99
$N = 512$	7	6.55
$N = 1024$	7	27.6
$N = 2048$	7	127
$N = 4096$	7	614

5 Conclusions

In our tests, both iterations convergs to the correct solution both in terms of forward and backward error. However, the first iteration, despite taking four times as many iterations, converged asymptotically around a factor of four faster due to its reliance on the explicitly invertible Laplacian in the Fourier basis. Applying the fixed-point iteration at multiple scales as part of the geometric multigrid method accelerated

the convergence of the first iteration by a factor of as much as 40%. The second iteration converged in few iterations, and reduces the error by a factor of ~ 10 each iteration. Anderson acceleration improves convergence of the second iteration considerably, by a factor of approximately 20%. On the whole, on this discretisation the first iteration is much faster than the second, especially with multigrid acceleration, and as such we recommend this type of fixed-point iteration, splitting the operator to make the linear system as close to a Laplacian as possible, as a method for solving more general elliptic PDEs on a periodic domain.

6 Future work

The relative timing of our two methods depended strongly on the direct inversion of the Laplacian made possible by the FFT. To test their relative merits somewhat more generally, it would be extremely informative to consider different discretizations of the PDE. Finite-element and finite-difference methods, although lower in order of the discretization, lead to extremely sparse operators, making it possible to consider, for example, a Newton's method-based iteration. Solving the Poisson equation would be more difficult in these discretizations, but would be approximately the same cost as solving $\nabla \cdot [A\nabla]u = f$, and could be efficiently inverted by the multigrid method with Gauss-Seidel or banded Gauss-Seidel smoothing. Importantly, other discretizations do not assume periodicity, so other, perhaps more common conditions like the hard-wall or Dirichlet conditions could be considered.

References

[1] Donald G. Anderson. "Iterative Procedures for Nonlinear Integral Equations". In: *J. ACM* 12.4 (Oct. 1965), pp. 547–560. ISSN: 0004-5411. DOI: 10.1145/321296.321305. URL: http://doi.acm.org/10.1145/321296.321305.

[2] William Briggs, Van Henson, and Steve McCormick. *A Multigrid Tutorial, 2nd Edition*. Jan. 2000. ISBN: 978-0-89871-462-3.

[3] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*. Vol. 4. Jan. 1985. ISBN: 3-540-12761-5.

[4] Olli Malli. *TIES594 PDE-solvers Lecture 14: Acceleration Methods*. Avaliable at http://users.jyu.fi/~oljumali/teaching/TIES594/14/fixedpoint.pdf. 2015.

[5] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. 2nd ed. SIAM, 2003.

[6] P. Wesseling. *An introduction to multigrid methods*. Pure and applied mathematics. John Wiley & Sons Australia, Limited, 1992. ISBN: 9780471930839. URL: https://books.google.com/books?id=MznvAAAAMAAJ.