Robust Multigrid

Optimal Solvers in PETSc

Matthew G. Knepley

University at Buffalo

Department of Computer Science and Engineering
University At Buffalo

February 22, 2021

I dedicate these notes to my wonderful wife Margarete, without whose patience and help they could never have been written.

Acknowledgements

TBD

Contents

1 What is Multigrid? 7

2 The Poisson Problem 9
2.1 Structured Grid . . . 9
2.2 Variable Coefficient . . . 10
2.3 Unstructured Grid . . . 11
2.4 Visualization . . . 11
2.5 Scaling . . . 11
2.6 Convergence . . . 12

3 How Does Multigrid Succeed? 15
3.1 Single-level Solves . . . 15
3.2 Multilevel Solves . . . 15
3.3 MiniProjects . . . 18

4 How Can Multigrid Fail? 21
4.1 MiniProjects . . . 21

5 How can we fix Multigrid? 23

Index 25

Chapter 1

What is Multigrid?

Multigrid is an iterative method for the solution of systems of linear and non-linear equations. In this treatment, we will think of the system of equations as arising from the discretization of a partial differential equation (PDE), but this need not be the case.

Multigrid is made up of three simple pieces:

1. a local solver on the fine grid, called a smoother,
2. a coarse grid solver, and
3. a mapping between functions on coarse and fine grids.

The idea is that the smoother handles details in the solution on the fine grid. It is inexpensive because it is local, meaning it only considers a few variables at a time. Then the coarse solve handles parts of the solution spanning the whole domain, but using a much smaller problem than the original. In fact, notice that we can use multigrid as the coarse solver, creating a tower of solvers on decreasing grids. As a first step, we will just imagine that the mapping is a simple average, or perhaps linear interpolation, from one grid to another.

Our simple conception of multigrid relies on two complementary assumptions:

1. Simple averages are good predictors of field values
2. Equations hold between the average values which are similar to the original equations

The first assumption follows from the mean value theorem for electrostatics, namely that the potential at any point is equal to the average over some ball surrounding that point which contains no charges. We expect some weighted average to work for general elliptic equations since the *maximum principle* tells us that increases in some directions are compensated by decreases in others. The second assumption is the heart of renormalization, and depends on the particular equations.

In order to see why multigrid is important, it is instructive to look at the alternatives. A direct factorization, such as Gaussian elimination (LU), will have cost in $\mathcal{O}(N^3)$ for N variables. Using a Krylov method, such as Conjugate Gradients (CG), should take about $\sqrt{\kappa} = h^{-1} = N^{1/d}$ steps, with each step linear in N, so that the total cost is in $\mathcal{O}(N^{(d+1)/d})$. Assuming that each application of the smoother takes time linear in the size of that grid, the total time will also be in $\mathcal{O}(N)$. For large N, this advantage can be substantial.

Chapter 2

The Poisson Problem

Picture showing a cube, embodying the physical domain, with arrow out to an equation (Laplacian), and then arrows pointing to entries in a matrix from the portions of the equation.

$$-\Delta u = f \tag{2.1}$$

We will use MMS to verify our implementation, which means we assume an exact solution u^* and then calculate the corresponding forcing function f which makes it solve our equation. For example, let

$$u^* = x^2 + y^2. \tag{2.2}$$

Then we have

$$\begin{aligned} -\frac{\partial^2 u^*}{\partial x^2} - \frac{\partial^2 u^*}{\partial y^2} &= f, && (2.3) \\ -2 - 2 &= f, && (2.4) \\ -4 &= f. && (2.5) \end{aligned}$$

discretization

2.1 Structured Grid

We can solve a 2D Poisson problem on a structured grid using finite differences in SNES ex5,

```
./ex5 -param 0.0 -da_grid_x 21 -da_grid_y 21 -da_refine 6
     -ksp_rtol 1.0e-9 -pc_type mg -pc_mg_levels 4 -snes_monitor -snes_view
```

This problem has 1,640,961 unknowns on the fine level, and 8,199,681 nonzeros in the system matrix. In more detail,

	Options	Explanation
`./ex5`	`-da_grid_x 21 -da_grid_y 21`	Original grid is 21x21
	`-ksp_rtol 1.0e-9`	Solver tolerance
	`-da_refine 6`	6 levels of refinement
	`-pc_type mg`	4 levels of multigrid
	`-pc_mg_levels 4`	
	`-snes_monitor -snes_view`	Describe solver

2.2 Variable Coefficient

We will introduce a coefficient κ into our equation.

$$-\nabla \cdot \kappa \nabla u = f \tag{2.6}$$

Physically, this coefficient can be a diffusivity, or thermal conductivity, or electric permittivity. When κ varies in space, we have an extra term in our equation

$$-\nabla \cdot \kappa \nabla u = f \tag{2.7}$$

$$-\kappa \Delta u - \nabla \kappa \cdot \nabla u = f \tag{2.8}$$

If we want to model a discontinuous coefficient, let us start with a continuous approximant

$$\kappa = 1 + (\alpha - 1)/2 \left(1 + \tanh\left(\beta\left(x - \frac{1}{2}\right)\right)\right) \tag{2.9}$$

$$\frac{\partial \kappa}{\partial x} = \frac{\alpha - 1}{2} \beta \operatorname{sech}^2\left(\beta\left(x - \frac{1}{2}\right)\right) \tag{2.10}$$

so that our equation becomes

$$-\kappa \Delta u - \frac{\alpha - 1}{2} \beta \operatorname{sech}^2\left(\beta\left(x - \frac{1}{2}\right)\right) \frac{\partial u}{\partial x} = f \tag{2.11}$$

As $\beta \to \infty$, the transition region shrinks, it is roughly of size $2/\beta$, so that the coefficient looks constant in the left (1) and right (α) halves of the domain and the derivative tends toward a delta function of strength $\alpha - 1$ at $x = 1/2$. The moral is that form we use for finite elements Eq. 2.6 will look like the prior equation with constant coefficient, plus a jump term proportional to the normal derivative of the solution where the coefficient changes.

Since we are interested in large coefficient changes, we will set the value of $\kappa = 10^{-k}$, where `-k <k>` is the command line argument. We will have three coefficient distributions in our tests. First, the constant distribution. Second, a step function at $x = 0.5$ which is one to the left and 10^{-k} to the right. Third, a checkerboard pattern alternating between one and 10^{-k}. The number of squares per dimension m is set using `-div <m>`.

κ	Option
10^{-k}	`-coeff_type constant`
$1 + (10^{-k} - 1)\Theta_x(0.5)$	`-coeff_type step`
$(\lfloor mx \rfloor + \lfloor my \rfloor) \bmod 2 + 10^{-k} (\lfloor mx \rfloor + \lfloor my \rfloor + 1) \bmod 2$	`-coeff_type checkerboard`

2.3 Unstructured Grid

We can solve a 2D Poisson problem on a unstructured grid using finite elements in the `poisson.c` code in our project. By default, we use `DMPlexCreateBoxMesh()`, which makes a small structured mesh of a box using either simplicies or tensor product cells. The user can set the cell type, domain dimensions, periodicity, and number of divisions from the command line. After the mesh is created, it can be regularly refined n times using `-dm_refine n`. The polynomial degree k for approximation of the potential is set using `-potential_petscspace_degree k`.

2.4 Visualization

We can visualize the grid using either X-Windows

```
./poisson -dm_refine 2 -potential_petscspace_degree 1 -dm_view draw -draw_pause 2
```

or Paraview

```
./poisson -dm_refine 2 -potential_petscspace_degree 1 -dm_view hdf5:sol.h5
${PETSC_DIR}/lib/petsc/bin/petsc_gen_xdmf.py sol.h5
```

The coefficient can also be seen in the same way

```
./poisson -dm_refine 2 -potential_petscspace_degree 1 -kappa_view draw -draw_pause 2
```

or Paraview

```
./poisson -dm_refine 2 -potential_petscspace_degree 1 -dm_view hdf5:sol.h5 -kappa_view hdf5:sol.h5:hdf5_viz:append
${PETSC_DIR}/lib/petsc/bin/petsc_gen_xdmf.py sol.h5
```

Notice that we have to give the `hdf5_viz` format to the κ vector since it is local. Global vectors, like the solution, choose this format automatically. We can choose the step function distribution, both from high to low

```
./poisson -dm_refine 2 -potential_petscspace_degree 1 -coeff_type step -k 3 -kappa_view draw -draw_pause 2
```

and low to high

```
./poisson -dm_refine 2 -potential_petscspace_degree 1 -coeff_type step -k -3 -kappa_view draw -draw_pause 2
```

We can choose a checkerboard of 64 squares with values 1 and 1000 using

```
./poisson -dm_refine 2 -potential_petscspace_degree 1 -coeff_type checkerboard -k -3 -div 8 -kappa_view draw -draw_pause 2
```

and a randomized version

```
./poisson -dm_refine 2 -potential_petscspace_degree 1 -coeff_type checkerboard -k -3 -div 8 -k_random
  -dm_view hdf5:sol.h5 -kappa_view hdf5:sol.h5:hdf5_viz:append
```

2.5 Scaling

We will begin our scaling runs with GMG, but also include AMG eventually. Our first run will use

Options	Explanation
`-potential_petscspace_degree 1`	P_1 finite elements
`-dm_plex_box_faces 16,16`	Original grid is 16x16
`-ksp_type cg`	Use Conjugate Gradients
`-ksp_rtol 1.0e-10`	Krylov tolerance
`-dm_refine_hierarchy 6`	6 levels of refinement
`-pc_type mg`	6 levels of multigrid
`-mg_levels_ksp_max_it 2`	V(2,2) cycle
`-mg_levels_esteig_ksp_type cg`	Use CG for eigen-estimation
`-mg_levels_esteig_ksp_max_it 10`	Use 10 iterates to estimate the spectrum bounds
`-mg_levels_ksp_chebyshev_esteig 0,0.05,0,1.05`	Chebyshev safety margins
`-mg_levels_pc_type jacobi`	Use Jacobi since it is parallel and GPU friendly
`-snes_monitor -ksp_monitor -snes_view`	Describe solver

I notice that my Apple also likes `-malloc_debug 0`. Looking at the output of `-log_view`, the total time to solve this system of 1M unknowns on my Macbook is a about 40s, but onyl about 27s is used by the solver itself. Most of the other time is used in the mesh hierarchy setup, such as preallocating the operator and constructing the maps between meshes.

2.6 Convergence

Suppose that we start with the scalable solver above, and wish to look at the convergence of the solver. We will start with a smaller problem of size 16129, where the size is controlled by the number of refinements, using bashinline-da_refine_hierarchy 3. If we use `-ksp_monitor`, we can print the residual norm at each iterate. Here an iterate means one V-cycle, and we are using V(2, 2) meaning two Chebhyshev/Jacobi iterations for each smoother application.

```
0 KSP Residual norm 1.246687667754e+02
1 KSP Residual norm 1.173183807978e+00
2 KSP Residual norm 2.148547148913e-01
3 KSP Residual norm 4.363569022569e-02
4 KSP Residual norm 5.269255735305e-03
5 KSP Residual norm 9.652538049515e-04
6 KSP Residual norm 1.407786924015e-04
7 KSP Residual norm 1.857771028731e-05
8 KSP Residual norm 3.107505106987e-06
9 KSP Residual norm 5.461746383081e-07
10 KSP Residual norm 9.885122430428e-08
11 KSP Residual norm 1.740238017074e-08
12 KSP Residual norm 2.779256440973e-09
```

We can instead plot the residual decrease as a line graph using `-ksp_monitor draw::draw_lg`, as shown on the left in Fig. 2.1. Using just `-ksp_monitor draw`, we can plot the residual itself over the domain, as shown on the right in the figure.

The same sequence can be repeated for the error, since we are using MMS and can produce the exact solution. Using `-ksp_monitor_error`, we produce the error norm for each iterate.

```
0 KSP Error norm 9.947802867762e-01
```

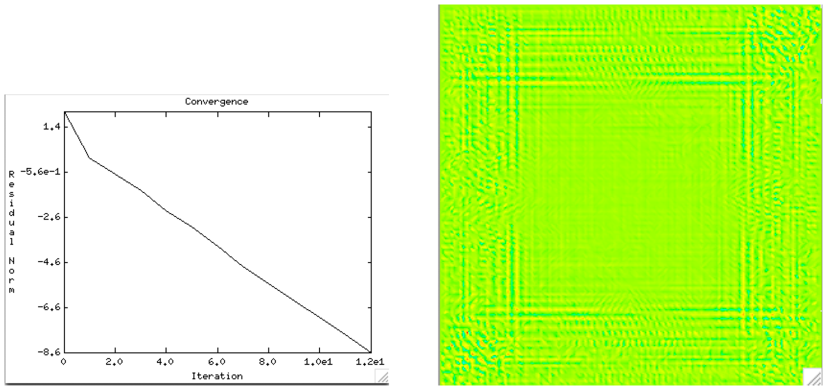


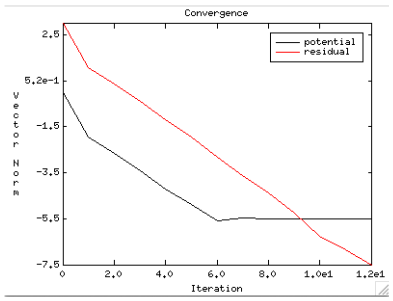
Figure 2.1: Plot of residual norm for each iterate of our multigrid solve on the left, and the residual itself for the last iterate on the right

```
1 KSP Error norm 1.000114579455e-02
2 KSP Error norm 1.628832514084e-03
3 KSP Error norm 5.600031707378e-04
4 KSP Error norm 1.947734270258e-04
5 KSP Error norm 2.221932067247e-04
6 KSP Error norm 2.171843316734e-04
7 KSP Error norm 2.176959633670e-04
8 KSP Error norm 2.176494135621e-04
9 KSP Error norm 2.176509954453e-04
10 KSP Error norm 2.176519724470e-04
11 KSP Error norm 2.176516886193e-04
12 KSP Error norm 2.176517676049e-04
```

Notice that the error stagnates well before we achieve convergence in the residual. This is because we hit the discretization error for our mesh. Refining the mesh will lower this bound, as we can see by running with `-dm_refine_hierarchy 6`,

```
0 KSP Error norm 9.993487701143e-01
1 KSP Error norm 1.232700481821e-02
2 KSP Error norm 2.400433256734e-03
3 KSP Error norm 4.230977361349e-04
4 KSP Error norm 6.442041832235e-05
5 KSP Error norm 1.421221753837e-05
6 KSP Error norm 2.668522964351e-06
7 KSP Error norm 3.579243209530e-06
8 KSP Error norm 3.382570470325e-06
9 KSP Error norm 3.409974576211e-06
10 KSP Error norm 3.408023096618e-06
11 KSP Error norm 3.407773635368e-06
12 KSP Error norm 3.407899220660e-06
```

As before we can also run using `-ksp_monitor_error draw::draw_lg` to generate a line graph of the above sequence, and `-ksp_monitor_error draw` to plot the error over the domain, which are shown in Fig. 2.2. Notice that the error follows the structure of the solution, whereas the residual is much more oscillatory and seems not to be connected to the solution structure.



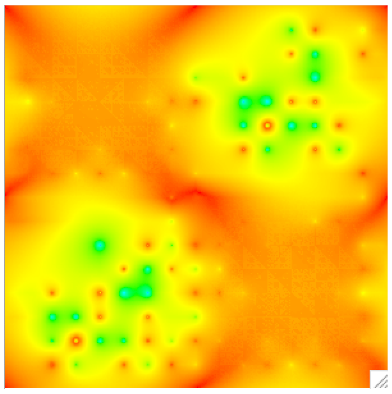


Figure 2.2: Plot of error norm for each iterate of our multigrid solve on the left, and the error itself for the last iterate on the right

Chapter 3

How Does Multigrid Succeed?

3.1 Single-level Solves

If we run a purely local solver, such as SOR, we see that the error (and residual) initially decrease quickly as we accurately solve local problems.

```
./poisson -potential_petscspace_degree 1 -dm_plex_box_faces 16,16 -dm_refine_hierarchy 3 \
  -ksp_type richardson -ksp_rtol 1e-10 -pc_type sor -ksp_type richardson -ksp_max_it 500 \
  -ksp_monitor_error draw::draw_lg -ksp_monitor_pause_final
```

However, we need increasingly distant data to make more progress, and the solver is only able to increase its reach by about one adjacent dof per iteration. Thus, we see a rapid slowdown and stagnation of the solve in Fig. 3.1. Note also that the error has distinct large scale features. These are the low frequency modes that are difficult to eliminate with a purely local solver.

If we augment SOR with a Krylov solver, we can see that for smaller problems this is enough to converge, as in the left side of Fig. 3.2.

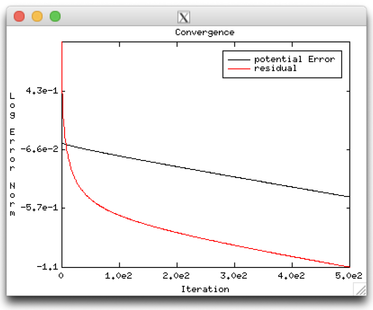
```
./poisson -potential_petscspace_degree 1 -dm_plex_box_faces 16,16 -dm_refine_hierarchy 3 \
  -ksp_type cg -ksp_rtol 1e-10 -pc_type sor -ksp_type richardson -ksp_max_it 500 \
  -ksp_monitor_error draw::draw_lg -ksp_monitor_pause_final
```

but if we just increase the problem size a little with `-dm_refine_hierarchy 6` then the stagnation returns. Asymptotically, Krylov solvers with local preconditioners make no progress at all on this problem.

Explain linear convergence and how to calculate the rate.

3.2 Multilevel Solves

Our multigrid will function properly if, at each level, the error component not eliminated by the smoother are eliminated by the coarse solver. Thus, we could start by dividing the space into coarse modes, supported on the



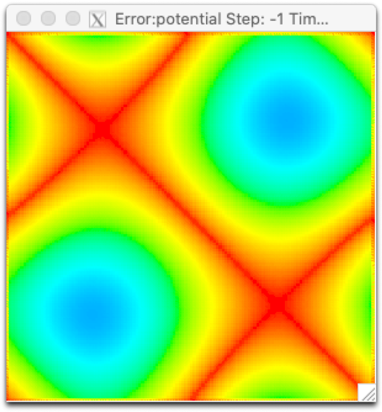


Figure 3.1: Plot of error norm for each iterate of an SOR solve on the left, and the error itself for the last iterate on the right

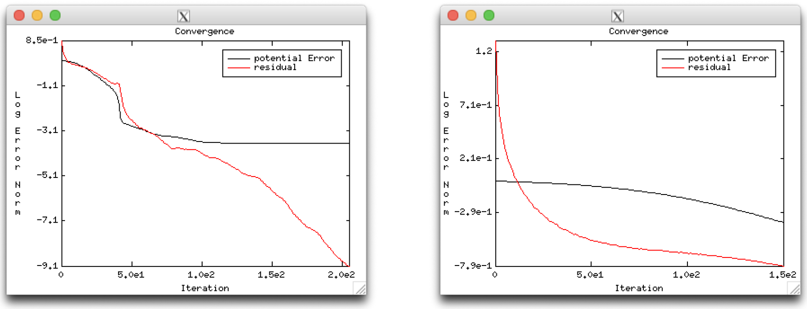


Figure 3.2: Plot of error norm for each iterate of an CG/SOR solve, with 3 refinements on the left and 6 refinements on the right.

coarse mesh, and the remaining fine modes, and then design solvers to handle these two components. However, a design methodology for the solvers is not clear to us. Instead, we will first select a smoother, and then adapt the coarse space to catch error components not handled by the fine solve. The smoother will certainly be local, but can be selected for other properties, such as preservation of a null space (**FarrellMitchellWechsung2018**; **FarrellKnepleyWechsungMitchell2020**).

In (**BrannickEtAl2018**), the authors characterize the optimal coarse grid space, which is the n eigenvectors of lowest eigenvalue for the generalized eigenproblem

$$A\mathbf{x} = \lambda \tilde{M}\mathbf{x} \tag{3.1}$$

where $\tilde{M}$ is the symmetrized smoothing operator, but we will drop the tilde from now on. The idea is that iterative smoothers capture best the modes for large eigenvalues of $M^{-1}A$, and thus we should design our coarse space $\mathcal{P}$ to support the lowest modes of this operator. The space handled well by the smoother is called $\mathcal{S}$, and is the M-orthogonal complement of the coarse space $\mathcal{P}$. The important point to keep in mind here is that the optimal coarse space *depends on* the smoother, so it does not make much sense to evaluate them independently.

If the coarse space captures modes not handled by the smoother, prolongation is accurate for these modes, and prolongation also does not dump too much energy in the modes the smoother does handle, then convergence will be rapid. Suppose that I had the exact solution in the coarse space. I would prolong this into the original fine space and then run the smoother. If the error is reduced quickly, without disturbing the coarse solution, then the smoother and coarse space are working well together. In short, if we smooth in the complement of the coarse space, we want to see rapid convergence. This kind of smoothing is called *compatible relaxation* (**Brandt2000**; **BrannickFalgout2007**) (CR), and one cycle can be written

$$\mathbf{v}_{k+1} = \left(I - S(S^T M S)^{-1} S^T A\right)\mathbf{v}_k. \tag{3.2}$$

Since $RS = 0$, we can write the whole thing as

$$\mathbf{v}_{k+1} = \left(I - M_S^{-1} A_S\right)\mathbf{v}_k. \tag{3.3}$$

where $A_S = S^T A S$ and likewise for M, as shown in (**BrannickEtAl2018**). In this nice talk by Rob Falgout, we suggests that the CR iteration should have a convergence rate of about 0.7. Above this, the coarse grid might be missing things.

Here is the monitor output from our Poisson run with constant coefficient, where we calculate the convergence rate r_V for each V-cycle.

```
0 SNES Function norm 4.525655418801e+01 Rate
  0 KSP Residual norm 9.984017906980e+02
  1 KSP Residual norm 1.180567977130e+01 0.012
  2 KSP Residual norm 2.332568775645e+00 0.20
  3 KSP Residual norm 4.124747107685e-01 0.18
  4 KSP Residual norm 6.605608734491e-02 0.16
  5 KSP Residual norm 1.195672600240e-02 0.18
  6 KSP Residual norm 1.532925985255e-03 0.13
  7 KSP Residual norm 2.442640618761e-04 0.16
  8 KSP Residual norm 4.660857239237e-05 0.19
  9 KSP Residual norm 5.762600750047e-06 0.12
 10 KSP Residual norm 5.834713047805e-07 0.10
 11 KSP Residual norm 1.505628327212e-07 0.26
1 SNES Function norm 6.902366392273e-09
```

so we have a pretty good contraction rate for the V-cycle. We can turn on a CR computation at each level on the homogeneous equation $Ax = 0$, and a monitor on the CR iteration to give the convergence rate, using

```
-pc_mg_adapt_cr
-mg_levels_cr_ksp_max_it 5 -mg_levels_cr_ksp_converged_rate
  -mg_levels_cr_ksp_converged_rate_type error
```

Below, we can see that the CR iterations are converging well on each level for the first few iterations, and it continues in this way until convergence.

```
0 SNES Function norm 4.525655418801e+01
            Linear mg_levels_1_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.671787 R^2 0.960782
          Linear mg_levels_2_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.642287 R^2 0.976622
        Linear mg_levels_3_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.642214 R^2 0.973073
      Linear mg_levels_4_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.646824 R^2 0.974903
    Linear mg_levels_5_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.645552 R^2 0.975915
  Linear mg_levels_6_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.645276 R^2 0.976259
  0 KSP Residual norm 9.984017906980e+02
            Linear mg_levels_1_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.671787 R^2 0.960782
          Linear mg_levels_2_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.642287 R^2 0.976622
        Linear mg_levels_3_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.642214 R^2 0.973073
      Linear mg_levels_4_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.646824 R^2 0.974903
    Linear mg_levels_5_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.645552 R^2 0.975915
  Linear mg_levels_6_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.645276 R^2 0.976259
  1 KSP Residual norm 1.180567977130e+01
            Linear mg_levels_1_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.671787 R^2 0.960782
          Linear mg_levels_2_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.642287 R^2 0.976622
        Linear mg_levels_3_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.642214 R^2 0.973073
      Linear mg_levels_4_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.646824 R^2 0.974903
    Linear mg_levels_5_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.645552 R^2 0.975915
  Linear mg_levels_6_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.645276 R^2 0.976259
  2 KSP Residual norm 2.332568775645e+00
            Linear mg_levels_1_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.671787 R^2 0.960782
          Linear mg_levels_2_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.642287 R^2 0.976622
        Linear mg_levels_3_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.642214 R^2 0.973073
      Linear mg_levels_4_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.646824 R^2 0.974903
    Linear mg_levels_5_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.645552 R^2 0.975915
  Linear mg_levels_6_cr_ solve converged due to CONVERGED_ITS iterations 5 error rate 0.645276 R^2 0.976259
  3 KSP Residual norm 4.124747107685e-01
```

3.3 MiniProjects

Use FFT to calculate the projection onto the eigenbasis for the Laplacian

- Project the error to show smoother decreasing coefficients of high frequencies
- Project error onto coarse space to show that it captures modes with appreciable coefficients

- Project corrected solution to show that mostly high modes are left

Use SLEPc to calculate the generalized eigenmodes

- Project the error to show smoother decreasing coefficients of high frequencies
- Project error onto coarse space to show that it captures modes with appreciable coefficients
- Project corrected solution to show that mostly high modes are left

Create monitor to display projected error

Chapter 4

How Can Multigrid Fail?

If the coarse space cannot capture modes not handled by the smoother, then those error components will not be reduced and the convergence will stall. Thus, we would like to have a way to monitor the quality of the coarse space. To do this, we run the process in reverse. Suppose that I had the exact solution in the coarse space. I would prolong this into the original fine space and then run the smoother. If the error is reduced quickly, without disturbing the coarse solution, then the smoother is working well. In short, if we smooth in the complement of the coarse space, we should see good convergence. This kind of smoothing is called *compatible relaxation* (**Brandt2000**; **BrannickFalgout2007**), and one cycle can be written

$$\mathbf{v}_{k+1} = \left(I - S(S^T M S)^{-1} S^T A\right) \mathbf{v}_k. \tag{4.1}$$

Since $RS = 0$, we can write the whole thing as

$$\mathbf{v}_{k+1} = \left(I - M_S^{-1} A_S\right) \mathbf{v}_k. \tag{4.2}$$

where $A_S = S^T A S$ and likewise for M, as shown in (**BrannickEtAl2018**).

4.1 MiniProjects

Create monitor to show CR error

Create monitor to show difference between projected error with and without coarse adaptation

Chapter 5

How can we fix Multigrid?

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the definition; numbers in roman refer to the pages where the entry is used.

compatible relaxation, 17, 21
discretization, 9
maximum principle, 7
smoother, 7