Lab Assignment 11
Multigrid Method

ACS II
Spring 2020

Assigned March 26th, 2020
Due April 2, 2020

Introduction

It has been shown analytically (through eigenvalue analysis) that iterative methods to solve linear systems damp high-frequency residuals very quickly, while damping low frequency residuals much more slowly. The effect of this when using an iterative linear solver is very fast convergence toward the exact solution initially (while the high frequency residuals are sent very nearly to zero) that slows down markedly once low frequency errors are all that remain. For highly resolved grids with residuals existing on a range of different frequencies, this can lead to agonizingly slow convergence to a desired level of tolerance.

A clever approach to this challenge is the use of multiple grid levels to beat down different frequency residuals. Recall that a low frequency signal on a very resolved grid is a relatively higher frequency component on a lower resolution grid. Thus, doing iterations an a variety of different grid levels allows you to remove residuals of all frequencies much more quickly. In order to move between the grid levels, we need to make use of several transformation.

We will use the multigrid method applied to the finite difference method to solve the one-dimensional Poission equation on the unit interval with Dirichlet boundary conditions,

$$u''(x) = -f(x), \qquad 0 \leq x \leq 1, \qquad u(0) = a,\ u(1) = b.$$

The finite difference approximation will result in a tri-diagonal linear system

$$A_h u_h = -f_h,$$

which must be inverted to obtain the discrete solution u_h. The subscript h denotes the grid spacing.

Algorithm

A single step of the multigrid method proceeds as follows:

1. Starting from an initial guess u_h^0 on a fine mesh, apply ν_1 steps of the weighted Jacobi method to get u_h^1
2. Compute the residual r_h on the fine mesh: $r_h = f_h - A_h u_h^1$
3. Restrict r_h to a coarser mesh to get r_{2h}
4. Solve $A_{2h} e_{2h} = r_{2h}$ for e_{2h} using the Thomas algorithm
5. Prolong e_{2h} to e_h
6. Correct u_h^1 by adding on e^h, i.e. $u_h^1 = u_h^1 + e_h$
7. Set u_h^0 to u_h^1

This algorithm loops until the residual on the fine mesh falls below some tolerance. Note that A_h is the finite difference matrix with grid spacing h, i.e.

$$A_h = \frac{1}{h^2} \begin{pmatrix} -2 & 1 & & & 0 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 0 & & & 1 & -2 \end{pmatrix}$$

There are a few parts to this algorithm which we will now go over in detail.

Weighted Jacobi

In the previous lab, we looked at the Jacobi method for 2D finite differences. Multigrid makes use of the *weighted* Jacobi method to smooth out the high frequencies in the solution. A second-order accurate finite difference stencil in 1D is given by:

$$u''(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}.$$

As we saw last week this leads to a system of linear equations, namely

$$u_{i-1} - 2u_i + u_{i+1} = -h^2 f_i, \qquad i = 1, \cdots, n-1.$$

As was the case last week u_0 and u_n are known from the boundary conditions.
Starting from an initial guess u^0 we can use the Jacobi method to solve this system by iteratively applying the relation

$$u_i^1 = \frac{u_{i-1}^0 + u_{i+1}^0 + h^2 f_i}{2},$$

until $||u^1 - u^0||$ is below some tolerance.
The weighted Jacobi method is similar, but it keeps part of u_i^0 to compute u_i^1,

$$u_i^1 = \omega \frac{u_{i-1}^0 + u_{i+1}^0 + h^2 f_i}{2} + (1-\omega)u_i^0. \tag{1}$$

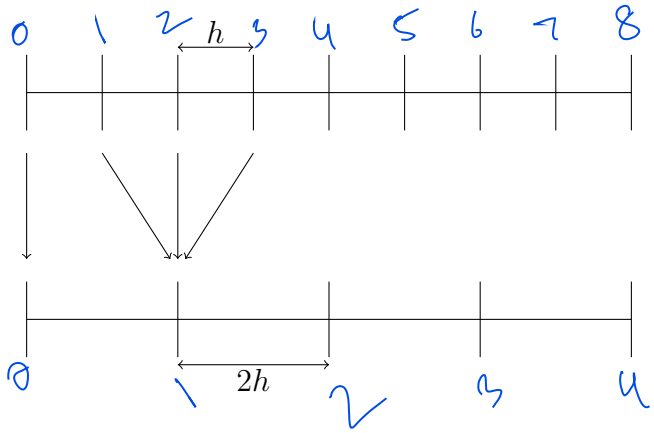
The constant ω is known as the weighting factor. Multigrid methods typically use $\omega = 2/3$. Since we are using Dirichlet conditions, the boundaries are trivial to implement.

Deliverable

Write a function `weighted_jacobi_step` that takes as an input an initial guess u^0, a function handle $f(x)$ and a number of points and performs a single step of the weighted Jacobi method according to (1) to return u^1. Starting from an initial u^0 that is random noise on the unit interval with $h = 1/128$, apply two steps of the weighted Jacobi method. Use the formulation (1) given above and let $f(x) = -4e^{2x}$. Apply the boundary conditions $u(0) = 1$ and $u(1) = e^2$. Plot the difference between the exact solution and your Jacobi iterates, $|u_{exact} - u^0|$ and $|u_{exact} - u^2|$ in the same figure, but with different colors or markers. Make sure that both curves can be seen clearly. Describe the key differences between the residuals.

Restriction

Moving from a fine grid to a coarse grid is known as *restriction*.



There are a couple ways to restrict a function from a fine grid to a coarse grid. The simplest is just to take the function values from the fine grid at points where the fine and coarse grid coincide. For this lab we will use the weighted average restriction. On the boundaries the value of the function on the coarse grid takes the value of the function on the fine grid. In the interior we take the weighted average,

$$u_i^{2h} = \frac{1}{4}(u_{2i-1}^h + 2u_{2i}^h + u_{2i+1}^h).$$

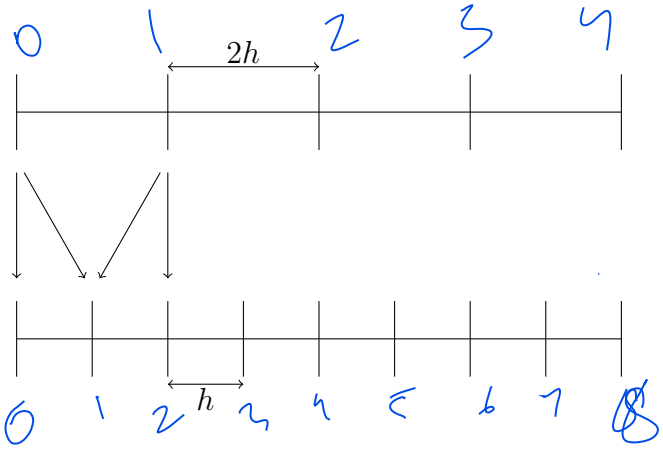
Deliverable

Write a function `function_restriction` that takes as an input function values on a grid with spacing h and returns the restricted function values on a grid with spacing $2h$.

Given a random vector a_h defined on the unit interval with $h = 1/128$, plot both a_h and a_{2h} on the same plot.

Prolongation

Moving from a course grid level to a fine grid level requires increasing the number of grid points and approximating new points from values at the coarser level. This is called *prolongation* in the multigrid community. While a variety of interpolation schemes can accomplish this task, for our case we'll use a linear interpolant to move from a grid with spacing $2h$ to a grid with spacing h.



On nodes where the fine and coarse mesh coincide we can just take the value from the coarse mesh and go directly to the fine mesh. For points that do no coincide we take the average of the two closest points on the coarse mesh to find the value on the fine mesh. This can be expressed mathematically as:

$$u_i^h = \begin{cases} u_{i/2}^{2h} & i \text{ even} \\ \dfrac{u_{(i-1)/2}^{2h} + u_{(i+1)/2}^{2h}}{2} & i \text{ odd} \end{cases}$$

Deliverable

Write a function `function_prolongation` that takes as an input function values on a coarse mesh and returns the function values on a fine mesh.
Given a random vector a^{2h} defined on the unit interval with $h = 1/32$, plot both a^{2h} and a^h on the same plot.

Thomas Algorithm

The multigrid method requires us to solve the linear system $A_{2h}e_{2h} = r_{2h}$. This is a smaller system to solve than the original system. There are many ways to solve this system, for example we could use the Jacobi method to solve it. In the 1D case A_{2h} is a tridiagonal system and there is a direct $O(N)$ method known as the Thomas algorithm available to us. For higher dimensional problems A_{2h} is banded, but not tridiagonal so other solvers would have to be used.
Consider the system

$$\begin{pmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix} \tag{2}$$

The Thomas algorithm creates two intermediate vectors c' and d' according to

$$c'_i = \begin{cases} \dfrac{c_i}{b_i} & i = 1 \\ \dfrac{c_i}{b_i - a_i c'_{i-1}} & i = 2, \cdots, n, \end{cases} \qquad d'_i = \begin{cases} \dfrac{d_i}{b_i} & i = 1 \\ \dfrac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}} & i = 1, \cdots n. \end{cases}$$

The solution is then computed using back substitution,

$$\begin{aligned} x_n &= d'_n, \\ x_i &= d'_i - c'_i x_{i+1}. \end{aligned}$$

Deliverable

Write a function `thomas_solver` that takes as inputs the vectors a, b, c and d (as defined in (2)) and returns the solution x computed using the Thomas algorithm.
Test your code on the 10×10 linear system $Ax = b$ where

$$A = \begin{pmatrix} 3 & -1 & & & 0 \\ -1 & 3 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 3 & -1 \\ 0 & & & -1 & 3 \end{pmatrix}, \qquad b = \begin{pmatrix} 2 \\ 1 \\ \vdots \\ 1 \\ 2 \end{pmatrix}.$$

The exact solution x should be $[1, 1, \cdots, 1, 1]^T$.

Experiment

Your task is as follows

- Use the weighted Jacobi method to solve the Poisson equation

$$u''(x) = 4e^{2x}, \qquad u(0) = 1,\ u(1) = e^2, \tag{3}$$

on the domain $x \in [0, 1]$ with $h = 1/256$. This ODE has the exact solution $u(x) = e^{2x}$. Compute the L_2 norm of the difference between your exact and numerical solution for each iteration for 2000 iterations. For the initial guess use uniform random numbers.

- Implement the multigrid method outlined above, and compute the solution to (3), with $h = 1/256$ as the fine mesh. Do $\nu_1 = 2$ weighted Jacobi iterations at the fine mesh and at most 1000 multigrid cycles. Compute the same L_2 metric. Plot the L_2 norm of both methods in the same figure with a log-log scale. Be sure to use a legend and different colors or markers. Use the exact same initial vector as above.

- Compare the computational cost of the two methods. To do this we will solve (3) again using weighted Jacobi and multigrid. This time we will use $h = 1/64$ and look at the effect of varying the stopping tolerance. Solve the ODE with the following stopping tolerances: $1 \times 10^{-1},\ 1 \times 10^{-2},\ 1 \times 10^{-3},\ 1 \times 10^{-4},\ 1 \times 10^{-5}$. Don't set a maximum number of iterations. For each tolerance report the number of weighted Jacobi iterations it took for each method to converge, i.e. complete the following table:

tolerance	iterations (Jacobi solver)	iterations (multigrid)
1×10^{-1}		
1×10^{-2}		
1×10^{-3}		
1×10^{-4}		
1×10^{-5}		

Use this table to estimate the computational cost. Clearly for the mutligrid method there are more steps than just the Jacobi iterations. Assuming a Thomas solve costs three times as much as a Jacobi iteration on the same mesh (since you're looping over x three times) and remembering that it is $\mathcal{O}(N)$, we can say that a Thomas solve on the coarse grid costs as much as 1.5 Jacobi iterations on the fine grid. If we neglect the cost of restriction and prolongation, the total cost of the multigrid method is then #Jacobi iterations $\times$ 2.5. Use this estimate to add a new column to your table that gives this approximate cost.

- Finally, it is well known that the Jacobi method is parallelizable. Use `openMP` to distribute the amount of work during the smoothing process. Using your jacobi step function, solve the problem only with jacobi iterations (no multigrid), but in parallel. Let $h = 1/256$ and run for 500,000 iterations. Compute the speedup gained with 2, 3, and 4 threads, and plot speedup versus number of threads. Average your results over several runs.

Submission and Grading

To get credit for this assignment, you must submit the following information to Canvas by 11:59pm, April 2, 2020

- your source code files
- Your report as a single file in pdf format, including results from your work and relevant discussion of your observations, results, and conclusions.

This information must be received by 11:59pm, April 2, 2020. Upload the required documents to Canvas. As stated in the course syllabus, late assignment submissions will be subject to a 10% point penalty per 24 hours past the due date at time of submission, to a maximum reduction of 50%, according to the formula:

*[final score] = [raw score] - min(0.5, 0.1 * [# of days past due]) * [maximum score]*