Computational Physics
Homework 7

Kieran Finn

November 19, 2014

1 Problem a

write a program to solve the 2D Poisson equation in a box using the multigrid method.

1.1 Description of the program

Most of the important code is stored in*multigrid_5.py*. Here I copy the method *SOR* from my last week's assignment but revert to red/black ordering as it is quicker to implement in numpy. I also define *GS* which runs SOR with $\omega = 1$ and *smooth* which runs *GS* a given number of times.

I define restriction and prolongation operators *restrict* and *prolong* respectively. *restrict* splits the array into 2×2 blocks and defines a new grid where each point is the average of the values in the block. *prolong* inverts this operation by constructing a 2×2 block from each point and putting them together to make an array 4 times as big. *restrict* is defined with an extra factor of $\frac{1}{2}$ as will be explained in the following section. There is also a method *apply_A* that acts with the discretized Laplacian on a state. All of these methods make use of numpy's *roll* function to take account of the periodic boundary conditions. They also rely on the fact that the box size is 1.0×1.0 to calculate the lattice spacing so if it is different this must be taken into account by renormalising f.

The *main* method is what actually performs the multigrid process. It takes as arguments an initial guess, ϕ, a right hand size f, a cycle parameter γ and initial and final smoothing numbers m_1 and m_2. It proceeds as follows:

1. perform smooth m_1 times
2. calculate the (negative of the) error $d = f - A\phi$

3. restrict d to the next coarser grid and construct a new field ψ, ititialised to all 0.

4. recursively call main again to solve $A\psi = d$, γ times.

5. prolong the result back to the current grid and add it to ϕ

6. perform smooth again m_2 times.

where steps 2 to 5 are skipped when the algorithm gets to the coarsest grid (2×2) and A refers to the discretised Laplacian on the appropriate grid.

I then also have the driver program *main.py* which sets up the initial conditions, runs a specified number of iterations of the multigrid method and outputs the data and an error, calculated by taking $|\phi - \phi_{exact}|$ on the finest grid and averaging over all entries.

1.2 Theoretical results

1.2.1 Normalisation of restriction operator

We want to compare applying the galerkin choice, $r_0 A_l p_0$ to a field x to simply applying A_{2l} to it where r_0 and p_0 are restriction and prolongation operators respectively with their original normalisation. figure 1 shows graphically what happens when we apply $r_0 A_l p_0$. Figure 1a shows the initial values of the field x. We then apply p_0, leading to figure 1b. We then act with A_l on this. Figure 1c gives the results of this for the points relevant to our calculation. Finally we apply the restriction operator, r_0, to get back to the coarse grid as shown in figure 1d.

We see from this that

$$r_0 A_l p_0 x = \frac{1}{2}\frac{1}{l^2}\left(x_2 + x_4 + x_6 + x_8 - 4x_5\right) \tag{1}$$

meanwhile, if we were to directly apply A_{2l} to the coarse grid we would have

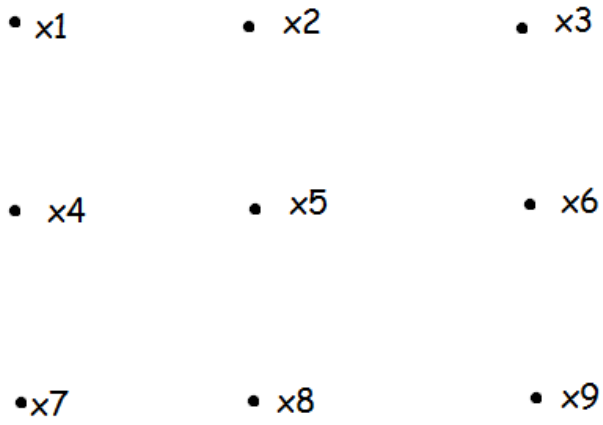
$$A_{2l}x = \frac{1}{(2l)^2}\left(x_2 + x_4 + x_6 + x_8 - 4x_5\right) = \frac{1}{2} r_0 A_l p_0 x. \tag{2}$$

The equation we solve in the multigrid method is

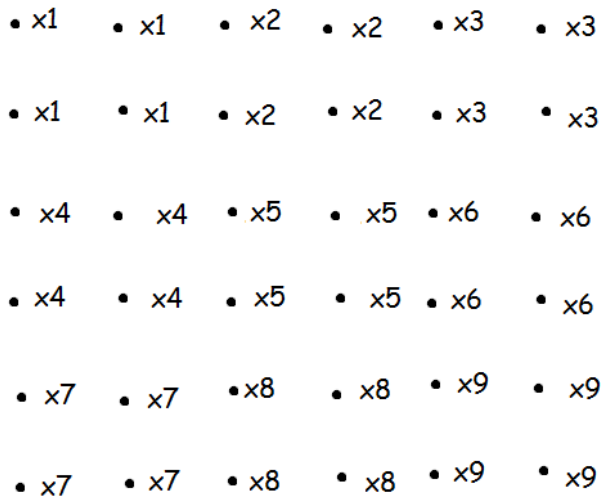
$$A_{2l}\psi = r\left(f - A_l \phi\right) \tag{3}$$

but the equation we actually want to solve is

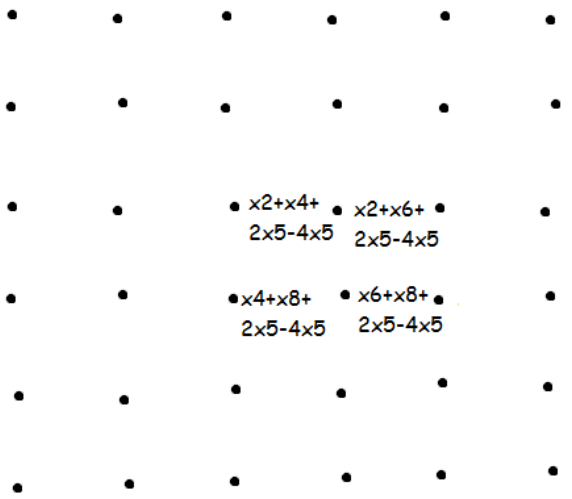
$$A_l(\phi + p\psi) = f \tag{4}$$



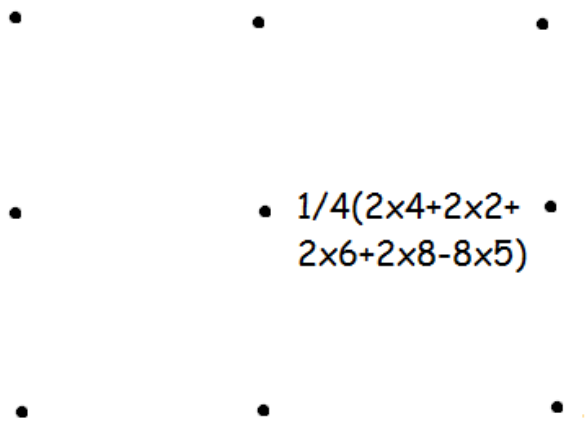
(a) Values of x on the coarse grid



(b) Values of x on the fine grid after applying the prolongation operator



(c) results of applying A_l to the fine grid



(d) The result then restricted back to the coarse grid

Figure 1: Action of the Galerkin choice operator, used to compare with $A_{2l}x$

To ensure these are the same normalisation we can renormalise our restriction operator so that $r = nr_0$. We choose not to renormalise the prolongation operator so $p = p_0$ as renormalising it as well turns out not to be necessary.

by inserting equation 2 into equation 3 we find

$$\frac{1}{2} r_0 A_l p0\psi = n r_0 (f - A_l \phi) \tag{5}$$

which, after a little rearranging gives us

$$r_0 A_l (\phi + \frac{1}{2n} p\psi) = r_0 f \tag{6}$$

which we see is equivalent to equation 4 as long as we choose $n = \frac{1}{2}$. This is the reason for the factor of $\frac{1}{2}$ in the definition of *restrict*

2 Problem b

Test your program on a case where you know the answer

2.1 Problem set up

I chose to test it on the function

$$\phi(x, y) = 4 \sin(6\pi x) \cos(2\pi y) \tag{7}$$

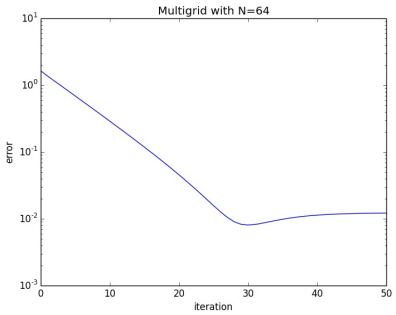
So the equation it is trying to solve is

$$\nabla^2 \phi = -160\pi^2 \sin(6\pi x) \cos(2\pi y) \tag{8}$$

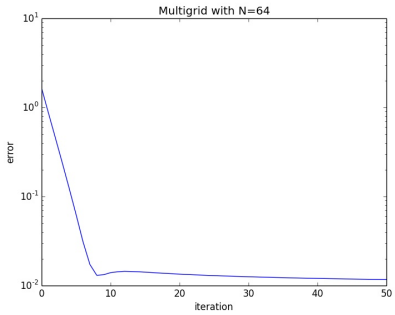
with periodic boundary conditions.

2.2 Experimental Results

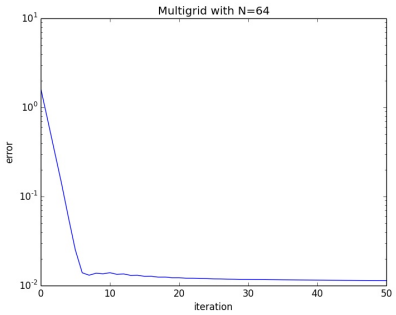
Figure 2 shows how the algorithm performs for N=64 and N=1024. We don't see much change for the small 64×64 grid but, for 1024×1024 the difference between Gauss-Seidel and the multigrid method is remarkable.



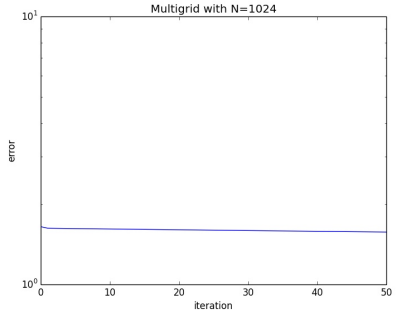
(a) N=64 Gauss Seidel



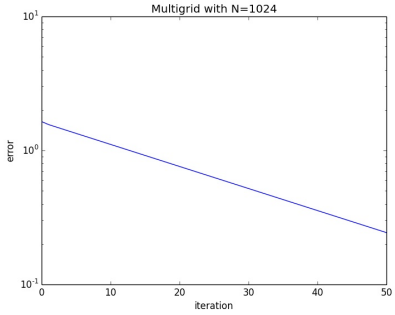
(b) N=64 V Cycle



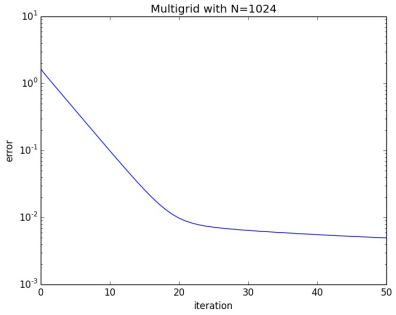
(c) N=64 W cycle



(d) N=1024 Gauss Seidel



(e) N=1024 V Cycle



(f) N=1024 W cycle

Figure 2: Comparison of Gauss Seidel V cycle and W cycle multigrid methods. See also the videos in the output directory.

The errors do not appear to be follow an eponential curve ($\epsilon \propto \rho^n$) as expected for a while before hitting a numerical floor after around 20 iterations. I must therefore be careful to use fewer than this when calculating the convergence rate later on.

3 Problem c

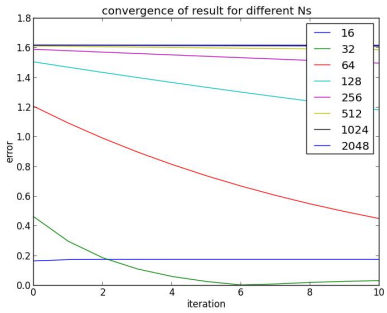
measure the asymptotic convergence rate as a function of lattice size.

3.1 Description of the program

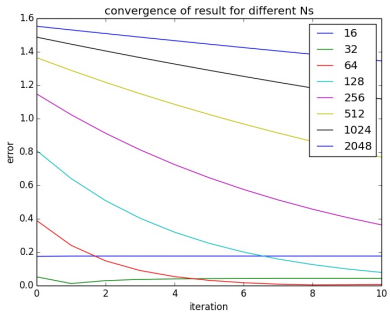
These results use the program *convergence.py*. This program takes a list of lattice sizes, N, and for each one runs the multigrid method. First it runs it a given number times to get to the asymptotic region, then it calculates the ratio of subsequent errors and averages over another given number of iterations to calculate the convergence rate ρ.

3.2 Experimental Results

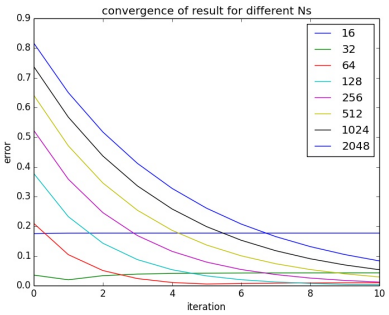
Figure 3 shows how the algorithm performs for N=64 and N=1024. We don't see much change for the small 64×64 grid but, for 1024×1024 the difference between Gauss-Seidel and the multigrid method is remarkable.



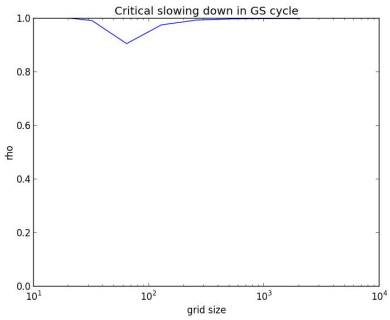
(a) Gauss Seidel



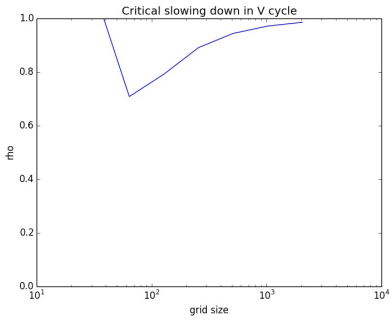
(b) V Cycle



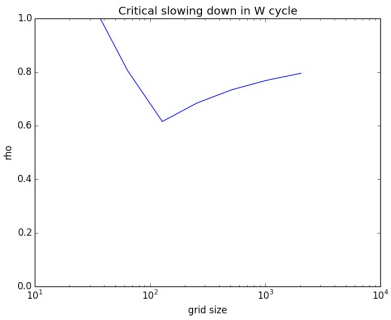
(c) W cycle



(d) Gauss Seidel



(e) V Cycle



(f) W cycle

Figure 3: Critical slowing down for Gauss Seidel and the V and W cycles.

There does seem to be some critical slowing down with both the V and W cycles, and I think this indicates a possible bug in either my code or the way I calculate the errors. However, we see t is significantly reduced compared to Gauss Seidel.

4 Problem d

Experimentally determine how the convergence rate depends on the number of pre and post sweeps, and on the cycle parameter γ

4.1 Independence of ratio

The first step was to test the the claim that the convergence rate depends only on the sum $M = m_1 + m_2$ and not on the individual values of m_1 and m_2 separately. This was done by comparing different ratios of $m_1 : m_2$ which all had the same sum using the program *compare_ratios.py*. Figure 4 shows that this is indeed the case.

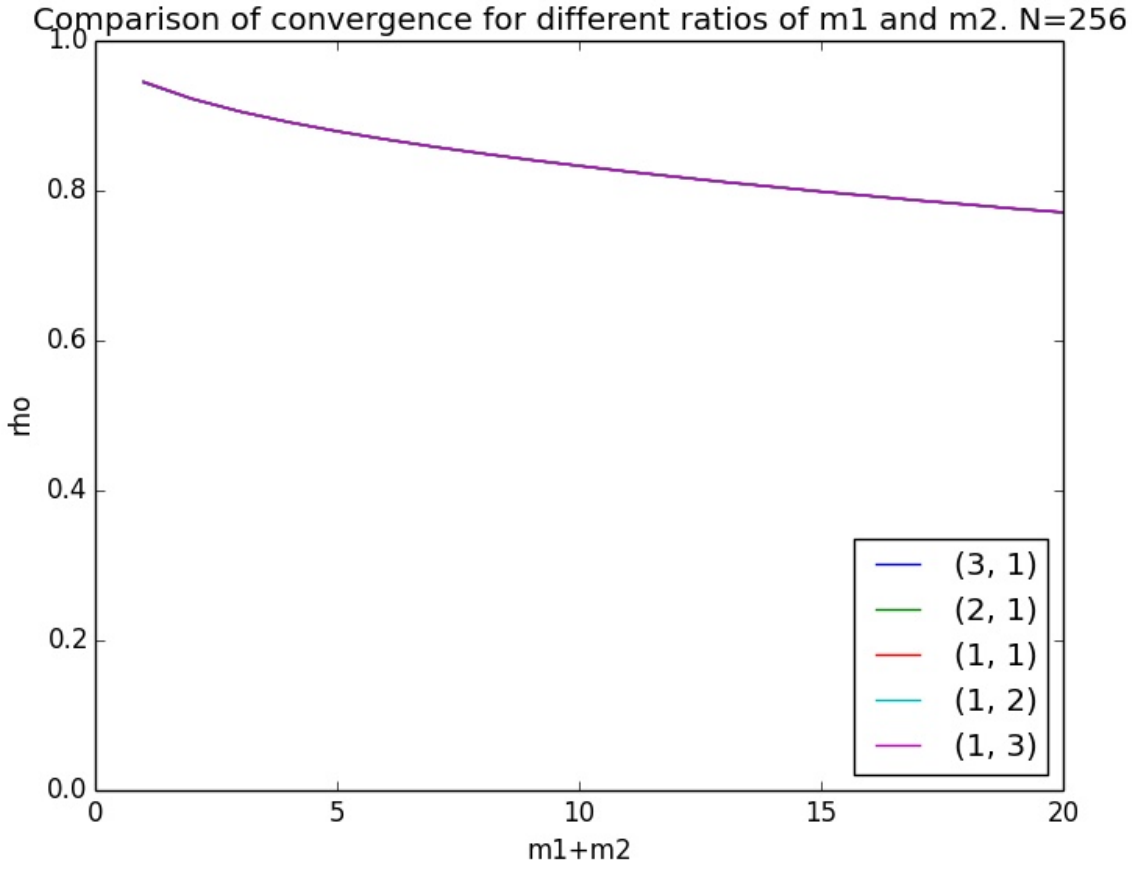


Figure 4: Comparison of the convergence rate of a V-cycle multigrid method for different ratios $m_1 : m_2$ but with the same sum $M = m_1 + m_2$.

4.2 Comparison of cycle parameters

The program *compare_m.py* was then used to produce a plot of the convergence rate vs $M = m_1 + m_2$ for Gauss Seidel, V-cycle and W-cycle. It is shown in figure 5.

We see that both multigrid methods perform better than Gauss Seidel, and the W cycle performs better than the V cycle. Finally, we see that increasing M does not have a significant effect on the convergence rate.

4.3 timing

compare_m.py also contains code which times how long each iteration takes to run. Figure 6 compares these.

We see that the V-cycle does not take much longer to perform than Gauss Seidel, whereas the W cycle takes 5-6 times longer, however, it does not result in as big an increase in performance so is probably not worth the extra effort. Based on these results the optimal combination of parameters would be

- $\gamma = 1$
- $m_1 = 1$
- $m_2 = 1$.

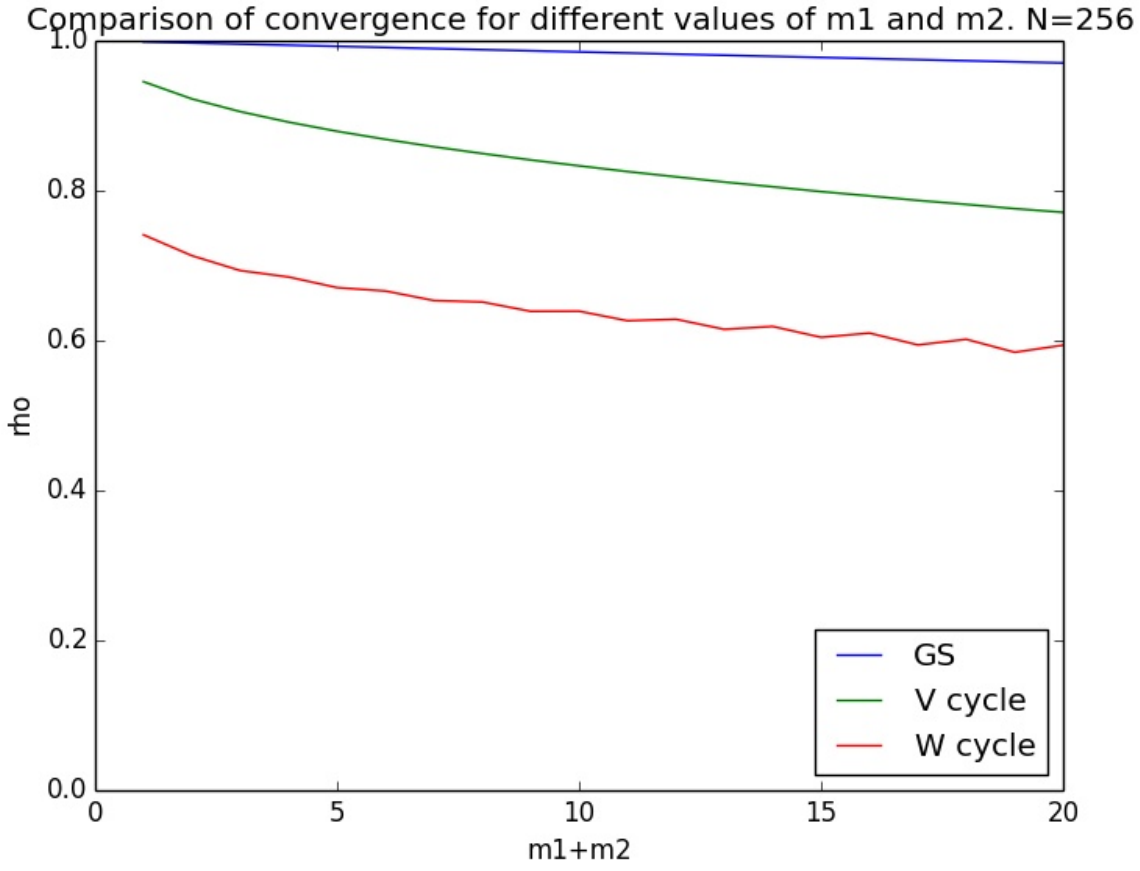


Figure 5: Comparison of the convergence rate of Gauss Seidel, V cycle and W cycle vs $M - m_1 + m_2$.

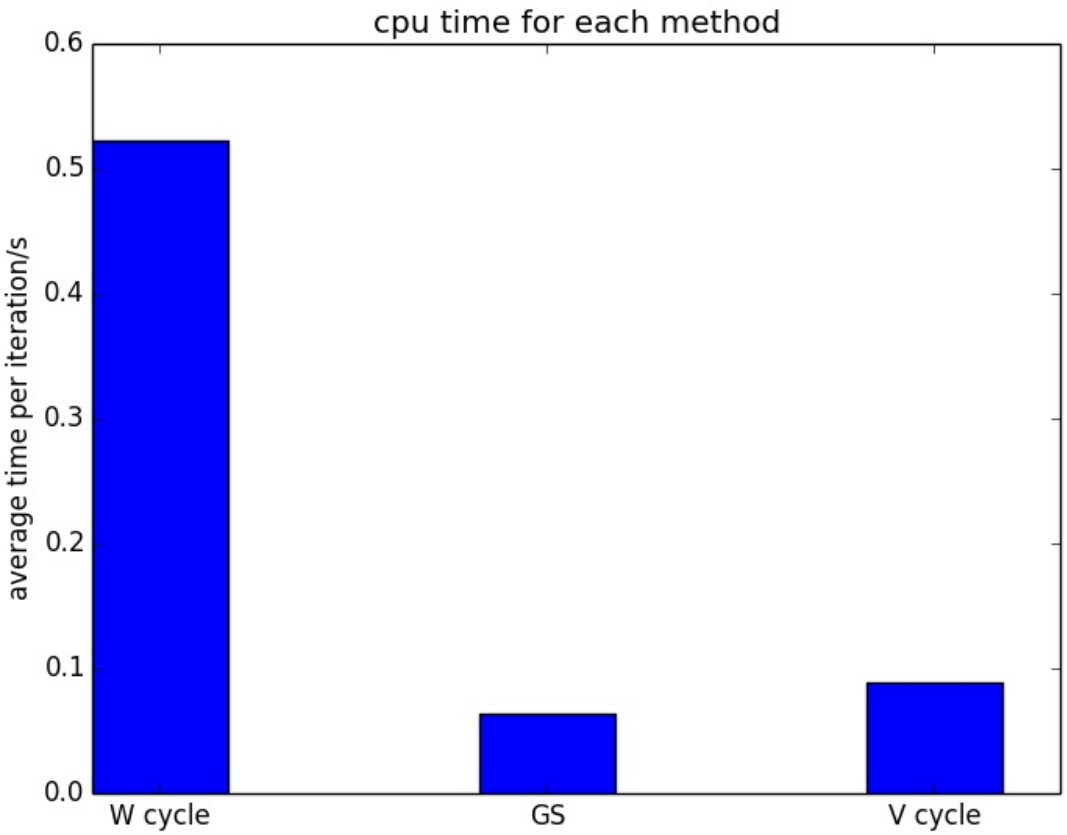


Figure 6: Average time per iteration for each method. $N = 256$.