Lab Course
Scientific Computing
Worksheet 5

Alfredo Parra, Shulin Gao, Teng Wang

Due January 16th, 2012

In this worksheet we revisit an instationary differential equation and we examine the performance of some iterative methods.

(a) The equation we wish to solve numerically is

$$T_{xx} + T_{yy} = -2\pi^2 \sin(\pi x)\sin(\pi y), \quad \forall (x,y) \in]0,1[^2, \tag{1a}$$

$$T(x,y) = 0, \quad \forall (x,y) \in \partial]0,1[^2. \tag{1b}$$

We apply a Gauss-Seidel iterative method to solve the equation, using $T(x,y) = 1$ as an initial guess, with an accuracy of 10^{-4}. The iterative method is given by

$$T_i^{(k+1)} = \frac{1}{a_{ii}} \left(F_i - \sum_{j=1}^{i-1} a_{ij} T_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} T_j^{(k)} \right), \tag{2}$$

where a_{ij} are the components of the matrix system describing Eq. (1) and F_i are the components of the RHS. The results obtained can be observed in Table 1. The memory is calculated as simply $N_x \cdot N_y$, since it is the length of the T vector that has to be stored (plus a constant number of variables). The MATLAB routine is named *GaussSeidel.m* (ran using `method` = 1). We also calculate the factor by which the number of iterations is reduced when we halve the step size of the grid.

$N_x = N_y$	3	7	15	31	63	127	255
# iterations	19	76	305	1219	4869	19463	77827
factor	-	4	4.0132	3.9967	3.9943	3.9973	3.9987
runtime (sec)	0.005	0.002	0.032	0.4349	6.426	101.0	1626
memory (floats)	9	49	225	961	3969	16129	65025

Table 1: Results for the iterative Gauss-Seidel method to solve Eq. (1).

(b) We now compare our answers with the exact solution, given by

$$F = \sin(\pi x)\sin(\pi y) \tag{3}$$

Figure 1 shows the error $E = T - F$ for $N_x = N_y = 255$ after 10, 20, and 30 Gauss-Seidel iterations.

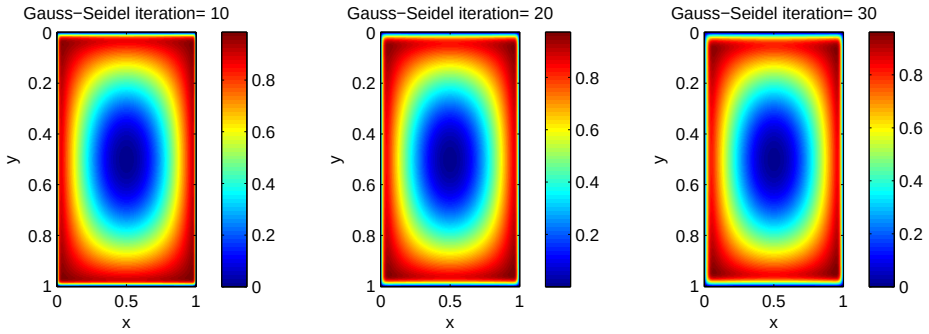

Figure 1: Error $E = T - F$ for $N_x = N_y = 255$.

The important observation is that the error is reduced very slowly (the rightmost figure takes slightly lower values, in blue), and thus the convergence of the method is poor.

(c) Now, a modified version of the Gauss-Seidel iteration (2) is implemented, namely, the SOR (*Successive over-relaxation*) Gauss-Seidel iteration. The scheme is given by

$$T_i^{(k+1)} = (1-\omega)T_i^{(k)} + \frac{\omega}{a_{ii}}\left(F_i - \sum_{j=1}^{i-1} a_{ij}T_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}T_j^{(k)}\right), \tag{4}$$

where

$$\omega = \frac{2}{1+\sin(\pi * h)} \tag{5}$$

is the overrelaxation factor. The method is implemented in `GaussSeidel.m` (setting `method` = 2).

(d) We can see in Table 2 that the implementation of the SOR method considerably improves the performance of the standard Gauss-Seidel method.

$N_x = N_y$	3	7	15	31	63	127	255
# iterations	11	23	48	99	205	426	908
factor	-	2.091	2.0870	2.0625	2.0707	2.0780	2.1315
runtime (sec)	0.006	0.001	0.005	0.034	0.2685	2.230	19.45
memory (floats)	9	49	225	961	3969	16129	65025

Table 2: Results for the iterative SOR Gauss-Seidel method to solve Eq. (1).

We can observe that the runtime for $N_x = N_y = 255$ was reduced by a factor of 84, almost two orders of magnitude. The plot for this choice of N can be seen in Fig. 2.

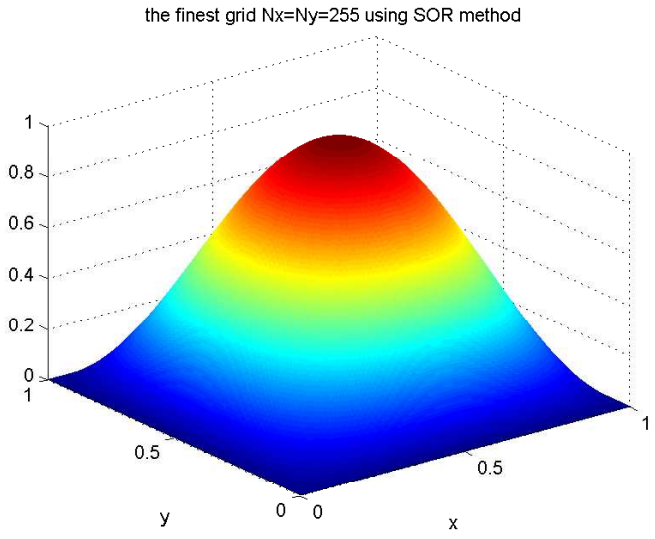

Figure 2: Solution for the finest grid using SOR Gauss Seidel

(e) We now implement a multigrid solver that works recursively. The idea is the following (this whole discussion follows from [1]). We want to solve a system $Au = f$, and we do so

starting with an approximation v of u. The residual r is then $r = f - Av$, and the error e is how much the calculated solution resembles the exact solution, that is, $e = u - v$. Therefore, the next equation holds

$$Ae = r = f - Av, \tag{6}$$

which says that the error can be relaxed with the RHS being the residual r. In other words, the relaxation of the original equation $Au = f$ with an arbitrary initial guess v is equivalent to relaxing on the residual equation $Ae = r$ with initial guess $e = 0$. Furthermore, we can use the fact that coarse grids can help us find a better initial guess to our problem. This means that in a grid with steps $h_x = h_y = h$ (which we denote $\Omega^{h\times h}$) we can do the following:

- Relax on $Au = f$ on a very coarse grid to obtain an initial guess for the next finer grid.
  $\vdots$
- Relax on $Au = f$ on $\Omega^{4h\times 4h}$ to obtain an initial guess for $\Omega^{2h\times 2h}$.
- Relax on $Au = f$ on $\Omega^{2h\times 2h}$ to obtain an initial guess for $\Omega^{h\times h}$.
- Relax on $Au = f$ on $\Omega^{h\times h}$ to obtain a final approximation to the solution.

If we put together the idea of using coarser grids with our discussion of the residual formula, we can come up with a more complete procedure, namely:

- Relax on $Au = f$ on $\Omega^{h\times h}$ to obtain an approximation of v^h.
- Compute the residual $r = f - Av^h$.
  - Relax on the residual equation $Ae = r$ on $\Omega^{2h\times 2h}$ to obtain an approximation to the error e^{2h}.
- Correct the approximation obtained on $\Omega^{h\times h}$ with the error estimate obtained on $\Omega^{2h\times 2h}$: $v^h \leftarrow v^h + e^{2h}$.

This procedure involves only two grids, but it can easily be extended to more grids recursively. The resulting routine is one *v-cycle*. But before looking at it, we need some routines to transfer data among grids and to calculate the residuals:

1. The routine `GS.m` does two Gauss-Seidel iterations for given N_x, N_y, and a specific RHS. It also receives the initial approximation of the solution.
2. The residual of a certain approximation and a given RHS is calculated using `RES.m`. The steps N_x, N_y also have to be specified.
3. To go from a fine grid to a coarse-grid, we simple send the fine-grid function to the routine `RESTRICT.m`, specifying the fine-grid and coarse-grid resolution. We call this

operator I_{hf}^{hc}, such that $I_{hf}^{hc} v^{hf} = v^{hc}$ (where hf and hc stand for h-fine and h-coarse). The restriction rule is given by

$$v_j^{2h} = v_{2j,2i}^h. \tag{7}$$

(This is valid for adyacent grids, but the function `RESTRICT.m` allows for larger jumps among grids.)

4. Finally, to go from a coarse to a fine-grid, the coarse-grid function is linearly interpolated using the function `INTRP.m`. We can call this operator I_{hc}^{hf}, such that $I_{hc}^{hf} v^{hc} = v^{hf}$. The linear interpolation rule in 2D is given by

$$v_{2i,2j}^h = v_{i,j}^{2h}, \tag{8a}$$

$$v_{2i+1,2j}^h = \frac{1}{2}(v_{i,j}^{2h} + v_{i+1,j}^{2h}), \tag{8b}$$

$$v_{2i,2j+1}^h = \frac{1}{2}(v_{i,j}^{2h} + v_{i,j+1}^{2h}), \tag{8c}$$

$$v_{2i+1,2j+1}^h = \frac{1}{4}(v_{i,j}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h}), \quad 0 \le i,j \le \frac{n}{2} - 1. \tag{8d}$$

(f) Now we have all the elements to implement the *v-cycle*. Using the recursive definition, it looks as follows:

$$v^h \leftarrow vCycle(v^h, N, f)$$

1. Relax 2 times on $A^h u^h = f^h$ with a given initial guess v^h.
2. If $\Omega^{h\times h}$ is the coarsest grid, go to step 4.
   Else

$$f^{2h} \leftarrow I_h^{2h}(f^h - A^h v^h),$$
$$v^{2h} \leftarrow 0,$$
$$v^{2h} \leftarrow vCycle(v^{2h}, N_{coarse}, f^{2h}).$$

3. Correct $v^h \leftarrow v^h + I_{2h}^h v^{2h}$.
4. Relax two times on $A^h u^h = f^h$ with initial guess v^h.

(g) The MATLAB routine `vSolve.m` runs the multigrid solver until an accuracy of 10^{-4} is reached. It receives the grid dimension as a parameter, and uses $T(x,y) = 1$ as initial guess.

(h) The routine `vSolve.m` is ran through `Worksheet5.m` using $T(x,y) = 1$ as an initial guess. The results for the different mesh sizes are reported in Table 3. Note that the number of iterations includes the Gauss-Seidel iterations carried out for both systems $AT = F$ and $Ae = r$. The storage is calculated as $S = \frac{8}{3}((N_x+1)\cdot(N_y+1)-4)$ (as explained in question 2, at the end).

$N_x = N_y$	3	7	15	31	63	127	255
# iterations	20	32	60	80	120	144	168
factor	-	1.6	1.875	1.333	1.5	1.2	1.166
runtime (sec)	0.0195	0.0132	0.0107	0.0190	0.0645	0.2095	0.7069
memory (floats)	32	160	672	2720	10912	43680	174752

Table 3: Results for the iterative vCycle method to solve Eq. (1).

We immediately see that the multigrid method yields much smaller runtimes with less iterations. For $N = 255$, the runtime is reduced by a factor of 27 compared to the SOR Gauss-Seidel method. The plot for the finest grid is shown in Fig. 3.

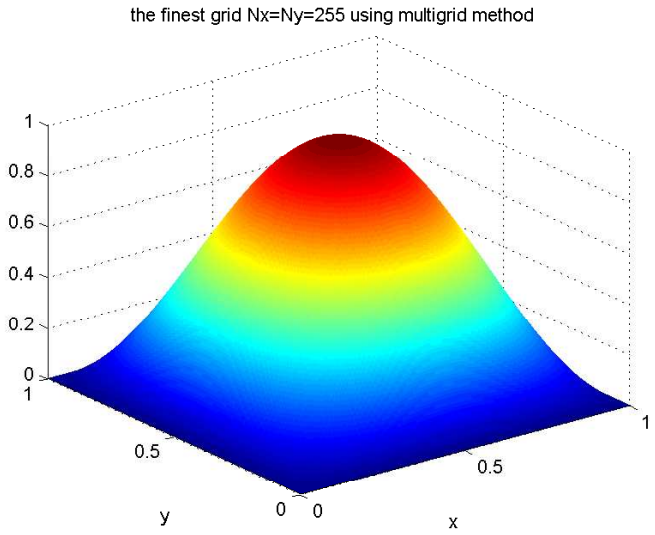

Figure 3: Solution for the finest grid using vCycle.

Questions

1. Denoting $N = N_x \cdot N_y$, we know that both Gauss-Seidel and SOR methods require $O(N)$ FLOPS at every iteration. Furthermore, the Gauss-Seidel method requires $O(N)$ iterations to obtain convergence (with a constant somewhere between 1 and 2), so the total computational effort is $O(N^2)$ (which can be observed in the runtime).

The SOR method requires $O(\sqrt{N})$ iterations to reach convergence, so the total effort is $O(N^{3/2})$. This might be an improvement, but is still not optimal. The optimal behavior would require $O(1)$ iterations to converge, thus a total effort of $O(N)$ (since the number of FLOPS per iteration is necessarily $O(N)$).

2. At the ith level of the multigrid method, we need to store two vectors of length $N = (N_x + 1) \cdot (N_y + 1)$ (the solution and the residual vectors). If the finest grid has $N = 4^k$ points, the next coarse level has 4^{k-1} points, until we go down to 4^2 grid points. That is, the grid at the ith level has $N_i = 4^i$ discretization points, and there are $\log_4 N$ levels. The total amount of storage is then

$$S = \sum_{i=2}^{\log_4 N} 2 \cdot 4^i = \frac{8}{3}(N - 4) \tag{10}$$

so $S = O(N)$, like the two previous methods.

3. The computational cost of one v-cycle is also $O(N)$. To see why this is, we observe that each step of the v-cycle requires either Gauss-Seidel iterations or interpolation operations (restriction does not cost FLOPS). One Gauss-Seidel iteration at level i requires cN_i FLOPS, as well as interpolation. The total amount of FLOPS would be calculated similarly to (10), only with a larger constant in front of the N. Although all three methods require $O(N)$ operations per iteration, the v-cycle requires only a constant amount of iterations, $O(1)$. Therefore, the total computational effort is $O(N)$.

4. Since we can never avoid doing $O(N)$ operations at each iteration, the v-cycle is optimal, since it preserves this asymptotic behavior for multiple iterations.

References

[1] Briggs W, Henson v. E, McCormick S, A Multigrid Tutorial, 2nd Edition. Philadelphia: SIAM; 2000.