Norwegian University of Science and Technology
Department of Mathematical Sciences

TMA4205 Numerical Linear Algebra Fall 2013

Semester project – part 2

Part 2 of the semester project focuses on implementation of multigrid and the preconditioned conjugate gradient (PCG) method for solving the 2D Poisson equation

$$\begin{aligned} -\nabla^2 u &= f(x,y), \quad (x,y) \in \Omega = (0,1) \times (0,1), \\ u &= g(x,y), \quad (x,y) \in \partial\Omega. \end{aligned} \tag{1}$$

The discretization should be done using a 5-point stencil (as used earlier in the course, see `poisson2.m`) with step-size $h = 1/N$ in both x- and y-directions, resulting in a system of linear equations $Av = b$. Multigrid will be tested both as a basic solver and as a preconditioner in a PCG iteration.

All your routines should operate directly on the grid. This means that you should not store the matrix A at any time. The right-hand side, which has elements coming from b, and the numerical solution U where $U_{i,j} \approx u(x_i, y_j)$, should be stored as $(N+1) \times (N+1)$-matrices (including boundary points).

Please read through the whole project text before starting on the problems. The text assumes that you are using MATLAB, but you may use other suitable languages, like Octave or Python with SciPy.

1 Implement a version of the conjugate gradient (CG) method operating directly on the grid. The program should contain a convergence test so that it terminates when

$$\frac{\|r_k\|_2}{\|r_0\|_2} < \text{tol}.$$

The function header could for instance look like this:

```
function [U, res, iter] = my_cg(U0, rhs, N, tol, max_it)
%
% [U, res, iter] = my_cg(U0, rhs, N, tol, max_it) performs CG for the
% 2D Poisson problem on the unit square [0, 1] * [0, 1].
%
% input:  U0     - initial guess
%         rhs    - right-hand side
%         N      - U0 and rhs are (N + 1) * (N + 1) matrices
%         tol    - relative residual tolerance
%         max_it - maximum number of iterations
%
% output: U      - numerical solution
%         res    - norm of residuals for each iteration
%         iter   - vector of iteration numbers
```

Test the program on (1) with

$$f(x,y) = 20\pi^2 \sin(2\pi x)\sin(4\pi y), \qquad g(x,y) = \sin(2\pi x)\sin(4\pi y).$$

2 Implement a multigrid V-cycle for (1). As input, the program should take the initial guess, the right-hand side, the number of levels (or grids), the number of pre-smoothings, and the number of post-smoothings. Use the CG algorithm from **1)** to solve the problem on the coarsest level. Both weighted Jacobi and red-black Gauss–Seidel should be used as relaxation method. Use linear interpolation and full-weighting to transfer information between grids. You should not explicitly form the matrices representing these transfers.

It is advantageous to break the program down into several parts. The main routine could for instance look like this:

```
function U = mgv(U0, rhs, N, nu1, nu2, level, max_level)
%
% U = mgv(U0, rhs, N, nu1, nu2, level, max_level) performs one multigrid
% V-cycle for the 2D Poisson problem on the unit square.
%
% input:  U0        - initial guess
%         rhs       - right-hand side
%         N         - U0 and rhs are (N + 1) * (N + 1) matrices
%         nu1       - number of pre-smoothings
%         nu2       - number of post-smoothings
%         level     - current level
%         max_level - total number of levels
%
% output: U         - numerical solution

if level == max_level
    [U, ~, ~] = my_cg(U0, rhs, N, 1e-12, 1000);
else
    U   = jacobi(U0, rhs, 2 / 3, N, nu1);
    rh  = residual(U, rhs, N);
    r2h = restriction(rh, N);
    e2h = mgv(zeros(N / 2 + 1), r2h, N / 2, nu1, nu2, level + 1, ...
              max_level);
    eh  = interpolation(e2h, N / 2);
    U   = U + eh;
    U   = jacobi(U, rhs, 2 / 3, N, nu2);
end
```

Notice that the routine is defined recursively. What is left then is to implement the routines `jacobi.m`, `residual.m`, `restriction.m` and `interpolation.m`. In addition, red-black Gauss–Seidel (`gs_rb.m`) must be implemented.

Test the program on the problem

$$\begin{aligned} f(x,y) &= -1, \\ g(0,y) &= 4y(1-y), \\ g(1,y) &= g(x,0) = g(x,1) = 0. \end{aligned}$$

Use a random initial guess for the interior points (MATLAB function **rand**). Test the program for different choices of the number of pre- and post-smoothings. Are there any differences between using weighted Jacobi and red-black Gauss–Seidel as relaxation method? How does the number of iterations to convergence depend on the size of the problem?

3 Modify the routine in **1)** to solve the 2D Poisson problem with a PCG method using the multigrid V-cycle from **2)** as preconditioner. From the course, we had that application of a preconditioner M implied that we had to solve the linear system $Mz = r$ for each CG iteration. Using the multigrid V-cycle as preconditioner corresponds to setting z equal to the result of one V-cycle with zero initial guess and the residual as right-hand side, i.e.

```
z = mgv(zeros(N + 1), r, N, nu1, nu2, 1, max_level);
```

The preconditioner must however be applied with some care. What properties must a preconditioner for the CG method have, and what does this mean for the multigrid V-cycle?

Test the algorithm on the same problem as in **2)**. How does the number of iterations to reach convergence depend on the size of the problem?

Hint: Use matrix- or vector-operations whenever possible. E.g. for the 5-point stencil, the Jacobi iteration on component form is

$$U_{i,j}^{(k+1)} = \tfrac{1}{4}\left(U_{i,j-1}^{(k)} + U_{i-1,j}^{(k)} + U_{i,j+1}^{(k)} + U_{i+1,j}^{(k)} + h^2 F_{i,j}\right), \quad 2 \le i, j \le N.$$

In MATLAB this can be written as the matrix operation

```
index = 2 : N;
U(index, index) = 0.25 * (  U(index, index - 1) + U(index - 1, index) ...
                          + U(index, index + 1) + U(index + 1, index) ...
                          + h^2 * f(index, index));
```

which is much more efficient than

```
Unew = U;
for i = 2 : N
    for j = 2 : N
        Unew(i, j) = 0.25 * (U(i, j - 1) + U(i - 1, j) + U(i, j + 1) ...
                           + U(i + 1, j) + h^2 * f(i, j));
    end
end
U = Unew;
```

The report should include a short description of the discretization, and for the problems:

1) A plot of the solution and the convergence history for 3 different grid sizes (for instance $N = 10, 20, 50$), i.e. a plot of the Euclidean norm of the residual against the number of iterations.

2) A plot of the initial guess and the solution after each of the first 5 multigrid V-cycles for $N = 2^5$ (use **subplot** in MATLAB). Include a plot of the convergence history. This should be done with both weighted Jacobi and red-black Gauss–Seidel as relaxation method.

In addition, you should give a plot of the number of multigrid V-cycles to reach convergence as a function of the problem size $N = 2^L$. Start for instance with $L = 5$ and then increase L as far as the machine/program allows.

3) A plot of the initial guess and the solution after each of the first 5 PCG iterations for $N = 2^5$.

As in the previous problem, include a plot of the convergence history and report how the number of iterations to convergence depends on the problem size.

Use $\|r_k\|_2/\|r_0\|_2 < 10^{-12}$ as convergence criterion in all the computations.