CAAM 520 – Homework 4

Due: 4/24/2020

General remarks: Please submit your homework report (as a PDF file) along with any code to your personal CAAM 520 Git repository. Please provide a makefile that can be used to compile your code, and make sure that your code compiles and runs in the CAAM 520 virtual machine.

Note: Problems 1 and 2 use matrix files `nasa2146.bin`, `plbuckle.bin`, and `bcsstk08.bin` that have not been published together with this assignment due to licensing issues. You should have received copies of these files by email.

Problem 1 *(Generalized Jacobi method with PETSc, 50 pts.)*

Recall that the generalized Jacobi method for the solution of a linear system $Ax = b$ is given by the iteration

$$x^{(k+1)} = x^{(k)} + \omega D^{-1}\left(b - Ax^{(k)}\right), \tag{1}$$

starting from some initial guess $x^{(0)}$. Here, D is a matrix that contains the diagonal entries of A, i.e.,

$$D_{ij} = \begin{cases} A_{ij}, & \text{if } i = j, \\ 0, & \text{otherwise} \end{cases}.$$

a) Implement the generalized Jacobi method in the function `solve_jacobi()` in the file `jacobi.c` which is provided as part of this assignment. Your code should perform the iteration (1) until the desired tolerance is reached, i.e., until

$$\frac{\|b - Ax^{(k)}\|_2}{\|b - Ax^{(0)}\|_2} < \text{tol},$$

or until the maximum number of iterations is reached. The initial value of the input vector `x` should be used as the initial guess $x^{(0)}$ to start the iteration. Your function should return the number of Jacobi iterations that were performed. *(*25 pts.)

Hint: Consider using the PETSc function `MatGetDiagonal()`.

b) Solve the linear system $Ax = b$ using PETSc's conjugate gradient solver (`KSPCG`). Create and call the solver in the function `solve_cg()` in `jacobi.c`. Make sure that the solver stops based on the *relative* tolerance given by the `tol` argument or based on the maximum number of iterations given by the `max_iter` argument. The solver should not not stop based on an absolute tolerance or a divergence tolerance, i.e., set `abstol=0.0` and `dtol=1.0e12` when calling `KSPSetTolerances()`. Your function should return the number of CG iterations that were performed. Consult the PETSc documentation to find out how to obtain the number of iterations. *(*15 pts.)

c) The `main()` function in `jacobi.c` loads a matrix A from a file, constructs vectors x and b, and solves the linear system $Ax = b$ first with `solve_jacobi()` and then with `solve_cg()`. Solve linear systems using the following matrices and parameters:

   – Matrix from `nasa2146.bin` with $\omega = 0.25$ and a tolerance of 10^{-6}

– Matrix from `plbuckle.bin` with $\omega = 0.6$ and a tolerance of 10^{-6}
– Matrix from `bcsstk08.bin` with $\omega = 0.7$ and a tolerance of 10^{-6}

Report the number of Jacobi and CG iterations and discuss your observations. *(10 pts.)*

Hint: To run the first experiment using two MPI processes, run `mpirun -n 2 ./jacobi -A nasa2146.bin -omega 0.25 -tol 1e-6`.

Note: As always, do not modify the main function in `jacobi.c`.

Problem 2 *(Inverse iteration, 50 pts.)*

In class, we discussed the power iteration

$$\tilde{v}^{(k+1)} = Av^{(k)}, \; v^{(k+1)} = \frac{\tilde{v}^{(k+1)}}{\|\tilde{v}^{(k+1)}\|_2},$$

which converges to an eigenvector of the symmetric real matrix A. Specifically, $v^{(k)}$ converges to an eigenvector for the eigenvalue of A with the largest magnitude. Recall that given $v^{(k)}$, the corresponding eigenvalue can be estimated by the *Rayleigh quotient*

$$\lambda^{(k)} = \frac{\left(v^{(k)}\right)^T Av^{(k)}}{\left(v^{(k)}\right)^T v^{(k)}} = \left(v^{(k)}\right)^T Av^{(k)}.$$

Now consider the *inverse iteration*

$$\tilde{v}^{(k+1)} = (A - \mu I)^{-1} v^{(k)}, \; v^{(k+1)} = \frac{\tilde{v}^{(k+1)}}{\|\tilde{v}^{(k+1)}\|_2}, \tag{2}$$

where I is the identity matrix. Note that (2) is simply the power iteration applied to the matrix $(A-\mu I)^{-1}$. Hence, $v^{(k)}$ converges to an eigenvector v for the eigenvalue of $(A - \mu I)^{-1}$ with the largest magnitude. It is easy to see that the same v is also an eigenvector of A for the eigenvalue closest to μ.

a) Using PETSc's `Mat` and `Vec` classes, implement the inverse iteration in the `inverse_iteration()` function in the file `inverse_iteration.c` which is provided as part of this assignment. Your code should perform the iteration (2) until the desired tolerance is reached, i.e., until

$$\|Av^{(k)} - \lambda^{(k)} v^{(k)}\|_2 < \text{tol},$$

or until a maximum number of iterations is reached. The initial value of the input vector `v` should be used as the initial guess $v^{(0)}$ to start the iteration. Your function should return the number of iterations that were performed.

Instead of inverting the matrix $A - \mu I$, use PETSc's `KSP` class to solve the linear system

$$(A - \mu I)\tilde{v}^{(k+1)} = v^{(k)} \tag{3}$$

during the update (2). Specifically, use the *generalized minimal residual method (GMRES)*, i.e., the PETSc solver `KSPGMRES`, to solve the system (3). Use tolerances of `rtol=1.0e-9`, `atol=0.0`, and `dtol=1.0e12` and a maximum number of `maxits=10000` iterations when calling `KSPSetTolerance()`. Your implementation may assume that the initial vector $v^{(0)}$ is normalized, i.e., $\|v^{(0)}\|_2 = 1$. *(*30 pts.*)*

Hint: Consider using the PETSc function `MatAXPY()` for the construction of $A - \mu I$.

Note: Do not confuse the tolerance `tol` and the maximum number of iterations `max_iter` for the inverse iteration with the tolerances `rtol`, `atol`, and `dtol` and the maximum number of iterations `maxits` for PETSc's GMRES solver.

b) The `main()` function in `inverse_iteration.c` loads a matrix and calls `inverse_iteration()`. Using the matrix from `plbuckle.bin`, perform the inverse iteration using a tolerance of 10^{-3} and values of $\mu \in \{0, 2.5, 5, 7.5, 10\}$. Report the computed eigenvalue estimate and the number of iterations for each choice of μ. If multiple choices of μ yield the same eigenvalue, is the number of iterations the same in each case? If not, can you see a pattern? *(20 pts.)*

   Hint: To run the first experiment using two MPI processes, run `mpirun -n 2 ./inverse_iteration -A plbuckle.bin -mu 0.0 -tol 1e-3`.

Note: As always, do not modify the main function in `inverse_iteration.c`.