AMS213A Project 4

Sutirtha Sengupta (Student id#:1566652)

March 15, 2017

Problem 1: Gauss-Jacobi/Gauss-Seidel methods

For a given $m \times n$ matrix A, the GaussJacobi subroutine (in the module LinAl.f90) solves the linear system of equations given by

$$Ax = b \tag{1}$$

for a given $m \times m$ matrix A and an m-long vector b using the Gauss-Jacobi algorithm, by effectively writing A as :

$$A \quad = \quad D + R \tag{2}$$

where D is the diagonal matrix containing the diagonal elements of A and R contains the rest of A. Hence,

$$x^{k+1} = D^{-1}(b - Rx^k) \tag{3}$$

which is used to compute the solution x iteratively since the (diagonal) elements of D^{-1} are simply given by $\frac{1}{a_{ii}}$. Written in component form, it is given by

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1} r_{ij} x_j^k) \tag{4}$$

with $r_{ii} = 0$.
In the actual GuassJacobi subroutine, D and R are not created separately but Eq. 4 is implemented directly noting $r_{ij} = a_{ij}$ for all $i \neq j$. The error in the solution at the k-th iteration is computed as

$$E_k = ||x^k - x^{k-1}||. \tag{5}$$

The Gauss-Seidel algorithm is similar to the Gauss-Jacobi one except that it effectively

uses the updated values of x as they are computed along to calculate the next coefficients. In component form, it is given by

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{m} a_{ij}x_j^k) \tag{6}$$

remembering that $r_{ii} = 0$. The error in the solution at the k-th iteration for this case is computed as

$$E_k = ||b - Ax^k||. \tag{7}$$

Convergence

The Gauss-Jacobi/Gauss-Seidel algorithms converge for matrices whose diagonal elements are large-compared to off-diagonal ones ("diagonally dominant" matrices) for which D^{-1} has small eigenvalues and so should $D^{-1}R$ and hence, $\rho(D^{-1}R) = max_i|\lambda_i| < 1$ which is the formal criterion for convergence for these algorithms. The rate of convergence of the Gauss-Seidel algorithm is always greater than that of the Gauss-Jacobi algorithm.

Results for varying D (for 10×10 A)

The driver program myprog.f90 creates a matrix A full of ones except on the diagonal where

$$a_{ii} = D \tag{8}$$

where D is a user-defined value. Also, it sets the RHS vector as

$$b_i = i \tag{9}$$

The code then calls either the Gauss-Jacobi (for argument 1) or Gauss Seidel (for argument 2) subroutines to solve for Eq. 1 following which b is updated with the solution, accurate to within a value *acc*, set to 10^{-6}.
For varying values of $D = 2$, 5, 10, 100, 1000, $||b - Ax||_k$ as a function of iteration number (k) is plotted in Figure 1.
Conclusion: The Gauss-Jacobi method only converges for $D = 10$ or above, and is always slower in convergence than the Gauss-Seidel method.

Test with $a_{ii} = i$:

With the diagonal entries of A given by $a_{ii} = i$, the Gauss-Jacobi does not converge to a solution within the specified accuracy (10^{-6}) but the Gauss-Seidel method does converge within 24 iterations giving $x_1 = -7.99999905$ and $x_i \simeq 1$(within machine accuracy) for all other $i = 2, 10$.

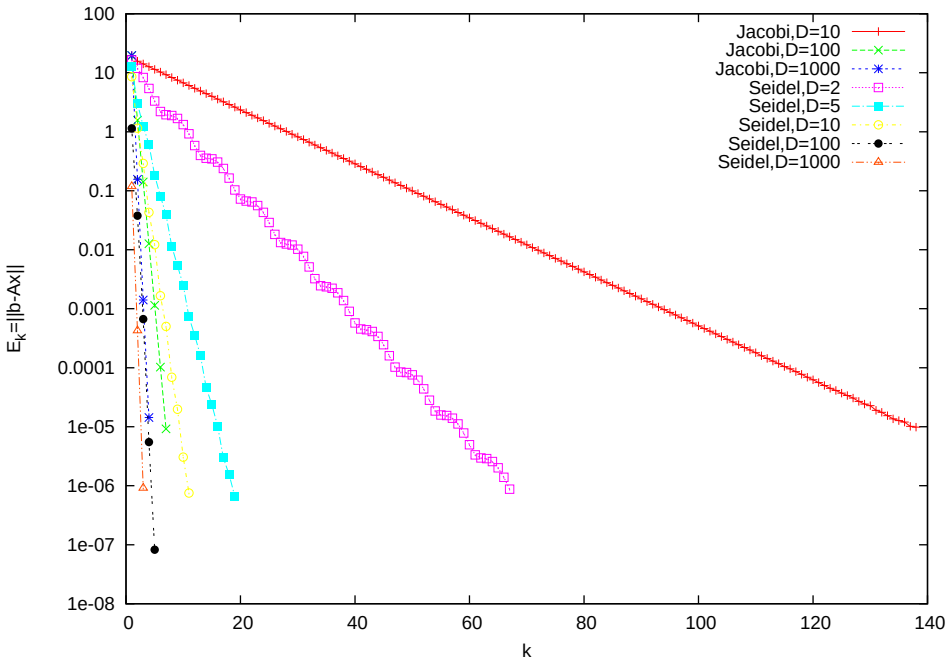

Figure 1: Convergence of the Gauss-Jacobi vs Gauss-Seidel algorithms

Problem 2: Conjugate Gradient method

The ConjGrad subroutine (in LinAl.f90 module) iteratively solves the linear system (1), by constructing a series of conjugate directions p_k (i.e.$p_k^T A p_i = 0$,for any $i < k$), starting with an initial guess for the solution x(in this case, we chose it to be $x_0 = 0$) through the following steps:

- $x_{k+1} = x_k + \alpha_k p_k$ where $\alpha_k = \frac{p_k^T r_k}{p_k^T A p_k}$
- $r_{k+1} = b - A x_{k+1}$
- $p_{k+1} = r_{k+1} + \beta_k p_k$ where $\beta_k = -\frac{r_{k+1}^T A p_k}{p_k^T A p_k}$

At the end, the RHS vector b is updated with the solution.
From the discussion in Lecture 15, we know that the error $e_k = x - x_k$measuring the difference between the true solution and the approximate solution at the k-th iteration satisfies

$$||e_{k+1}||_A \leq ||e_k||_A, \tag{10}$$

which implies that the error in the solution necessarily decreases monotonically and hence, the algorithm is guaranteed to converge to the true solution in a finite number of iterations.

To show that the smart version of the conjugate gradient algorithm is equivalent to the basic one, it suffices to show that, at the k-th iteration:

$$r_k^T p_k = p_k^T r_k \tag{11}$$

&

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \tag{12}$$

Proof : To prove the relation 11, we note that at $k = 0$,

$$r_0 = p_0 = b \Rightarrow r_0^T r_0 = p_0^T p_0 \tag{13}$$

and assume at $k - th$ iteration,

$$r_k^T r_k = p_k^T r_k. \tag{14}$$

From the Conjugate Gradient algorithm, we have,

$$\begin{aligned} r_{k+1} &= r_k - \alpha_k A p_k \\ p_{k+1} &= r_{k+1} + \beta_{k+1} p_k \\ \Longrightarrow p_{k+1}^T r_{k+1} &= r_{k+1}^T r_{k+1} + \beta_{k+1} p_k^T r_{k+1} \end{aligned}$$

Hence it suffices to show that $p_k^T r_{k+1} = 0$.
Now,

$$\begin{aligned} p_k^T r_{k+1} &= p_k^T r_k - \alpha_k p_k^T A p_k \\ &= p_k^T r_k - \frac{p_k^T r_k}{p_k^T A p_k} p_k^T A p_k \\ &= 0. \end{aligned}$$

Hence, $p_{k+1}^T r_{k+1} = r_{k+1}^T r_{k+1}$, thereby proving the relation 11 (via induction).
To prove relation 12 we note that

$$r_{k+1}^T r_k = 0 \tag{15}$$

and since

$$r_{k+1} - r_k = -\alpha_k A p_k \tag{16}$$

we can write,

$$r_{k+1}^T A p_k = \frac{1}{\alpha_k} r_{k+1}^T r_{k+1}$$

where by definition, $\alpha_k = \frac{p_k^T r_k}{p_k^T A p_k} = \frac{r_k^T r_k}{p_k^T A p_k}$ (using Eq. 14).
Hence,

$$\begin{aligned} r_{k+1}^T A p_k &= -\frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} p_k^T A p_k \\ \implies -\frac{r_{k+1}^T A p_k}{p_k^T A p_k} &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \end{aligned}$$

which proves relation (12), since the LHS above is β_k (by construction).
When the conjugate gradient algorithm is run on the 10×10 matrix A with $a_{ii} = D$ & $a_{ij} = 1\,(i \neq j)$, it converges much faster compared to both the Gauss-Jacobi and Gauss-Seidel algorithms for all values of D as shown by the number of iterations until complete convergence for different values of D for all the 3 algorithms in Table 1. Also, the number of iterations taken to converge decreases with increasing values of D as the matrix A becomes more "diagonally dominant" with increasing values of the diagonal elements.

D	GJ	GS	CG
2	-	67	4
5	-	19	3
10	138	11	2
100	7	5	2
1000	4	3	2

Table 1: Number of iterations until convergence for Gauss-Jacobi (GJ), Gauss-Seidel (GS) and Conjugate Gradient (CG) algorithms for the 10×10 matrix A with $a_{ii} = D$ & $a_{ij} = 1\,(i \neq j)$.

The diagonal preconditioner ($M_{ii} = a_{ii}$) is not very useful for these matrices, as all the diagonal elements of A are all equal, so the condition number of $M^{-1}A$ (that determines the rate of convergence of the CG algorithm) is no smaller than that of the original matrix A.

Tests with $a_{ii} = i$:

1.without diagonal preconditioning:

With the diagonal entries of A given by $a_{ii} = i$, the CG algorithm converges to the solution within 11 iterations for a 10×10 matrix, and 81 iterations for a 100×100 matrix.

2.with diagonal pre-conditioning:

Using diagonal pre-conditioning, the CG algorithm converges even faster. For the 10×10 case, it converges within 9 iterations while for a 100×100 case, it takes 13 iterations.