Lecture 26

ACCELERATED ITERATIVE METHODS FOR LINEAR SYSTEMS

26.1 INTRODUCTION

Basic iterative solvers such as Jacobi, Gauss-Seidel and SOR are very simple to program and take very little computational time per iteration. But their convergence rate is very slow. To get over this slow convergence, a variety of accelerated iterative methods have been developed. We will have a brief look at Krylov subspace methods in this lecture. For details, refer Saad (2003).

26.2 KRYLOV SUBSPACE METHODS

Krylov subspace methods (such as conjugate gradient, GMRES) belong to a wider class of methods called projection methods. Consider the linear system of n unknowns given by

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{26.1}$$

The idea of projection methods is to find an approximate solution from a subspace $\mathcal{K}$ of $\mathbf{R}^n$ new iterate such that residual vector is orthogonal to a subspace of constraints, $\mathcal{L}$. Thus, formal algorithm can be expressed as follows:

> Given an available iterate $\mathbf{x}_0$, find $\mathbf{x} \in \mathbf{x}_0 + \mathcal{K}$ such that residual $\mathbf{b} - \mathbf{A}\mathbf{x} \perp \mathcal{L}$ where $\mathcal{K}$ is the search subspace, and $\mathcal{L}$ denotes the subspace of constraints.

Krylov subspace method is the projection method for which search subspace $\mathcal{K}_m$ is Krylov subspace defined by

$$\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0) = \text{span}\{ \mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{m-1}\mathbf{r}_0 \} \tag{26.2}$$

Approximate solution $\mathbf{x}_m$ is of the form

$$\mathbf{x}_m = \mathbf{x}_0 + q_{m-1}(\mathbf{A})\,\mathbf{r}_0 \tag{26.3}$$

where q_{m-1} is a polynomial of degree m-1. Different choices of subspace of constraints $\mathcal{L}$ leads to different Krylov subspace methods, e.g.

- $\mathcal{L}_m = \mathcal{K}_m$: Conjugate gradient method (for symmetric linear systems)
- $\mathcal{L}_m = \mathbf{A}\mathcal{K}_m$: GMRES (for general linear systems)

Performance of a Krylov subspace solver such as PCG or GMRES depends on

1. Preconditioner which affects
   - Convergence of iterations
   - Overall computing time
2. Matrix-vector products: number of operations required for it affects
   - Overall computational efficiency

Preconditioner

Pre-conditioner for an iterative method should have the following properties to ensure a robust and efficient solution algorithm:

- Essential properties:
  - Should help cluster the eigenvalues of the preconditioned system.
  - Must be a constant linear operator in all the iterations.
- Desirable properties:
  - Should take as little time as possible
  - Result in small number of iterations *largely independent of problem size*.

There is a trade-off between preceding two desirable properties. For faster convergence, the preconditioning matrix (or operator) must be a close approximation to matrix **A**. **Some** commonly used pre-conditioners are:

- Jacobi (diagonal) preconditioner
  - Very fast but poor rate of convergence
- Incomplete LU factorization
  - Effective but problem of fill-ins
- Approximate inverse preconditioners
  - Block diagonal preconditioners
  - Preconditioners based on Frobenius norm minimization

26.3 SYMMETRIC LINEAR SYSTEMS: CONJUGATE GRADIENT METHODS

For symmetric and positive definite linear systems, the simplest and most commonly used Krylov subspace method is pre-conditioned conjugate gradient (PCG) method given in the box below (Saad, 2003). Here, **M** represents the preconditioning matrix, and (**x**,**y**) represents the inner (or dot) product of two vectors.

Set tolerance ε and initial guess $\mathbf{x}_0$
begin PCG
Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$; $\mathbf{z}_0 = \mathbf{M}^{-1}\mathbf{r}_0$; and $\mathbf{p}_0 = \mathbf{z}_0$.
for $j = 0,1,\ldots$ until convergence do

$$\alpha_j = (\mathbf{r}_j, \mathbf{z}_j) / (\mathbf{A}\mathbf{p}_j, \mathbf{p}_j)$$
$$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{p}_j$$
$$\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha_j \mathbf{A}\mathbf{p}_j$$
$$\mathbf{z}_{j+1} = \mathbf{M}^{-1}\mathbf{r}_{j+1} \quad \text{// } \textbf{\textit{Preconditioner for PCG}}$$
$$\beta_j = (\mathbf{r}_{j+1}, \mathbf{z}_{j+1}) / (\mathbf{r}_j, \mathbf{z}_j)$$
$$\mathbf{p}_{j+1} = \mathbf{z}_{j+1} + \beta_j \mathbf{p}_j$$

end do
end PCG

Note that the pre-conditioning step ($\mathbf{z} = \mathbf{M}^{-1}\mathbf{r}$) essentially requires solution of a linear system $\mathbf{M}\mathbf{z} = \mathbf{r}$ (inversion of **M** is required). Very often, **M** is not even explicitly defined: we only need an operator which is constant in each iteration, and returns the correction vector **z** for the given estimate of the residual **r**.

The simplest choice for the pre-conditioner is **M** = diag (**A**), i.e. **M** is a diagonal matrix whose elements are main diagonals of **A**. This pre-conditioner is referred as Jacobi pre-conditioner.

26.4 GENERAL LINEAR SYSTEMS: BICONJUGATE GRADIENT METHOD

PCG is applicable only to symmetric and positive definite linear systems. For general systems, various generalizations have been developed. Simplest generalization of PCG for a general system is the bi-conjugate gradient method. Preconditioned version of the algorithm is given in the box below (Press et al, 2002). Once again, **M** represents the preconditioning matrix, and (**x**,**y**) represents the inner (or dot) product of two vectors. We require three additional set of vectors: $\mathbf{r}^*, \mathbf{z}^*, \mathbf{p}^*$.

Set tolerance ε and initial guess $\mathbf{x}_0$

begin PBiCG

Compute $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$; $\mathbf{z}_0 = \mathbf{M}^{-1}\mathbf{r}_0$; $\bar{\mathbf{r}}_0 = \mathbf{r}_0$, $\bar{\mathbf{z}}_0 = (\mathbf{M}^T)^{-1}\bar{\mathbf{r}}_0$, $\mathbf{p}_0 = \mathbf{z}_0$ and $\bar{\mathbf{p}}_0 = \bar{\mathbf{z}}_0$.

for $j = 0,1, \ldots$ until convergence do

$$\alpha_j = (\bar{\mathbf{r}}_j, \mathbf{z}_j) / (\bar{\mathbf{p}}_j, \mathbf{A}\mathbf{p}_j)$$

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha_j \mathbf{p}_j$$

$$\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha_j \mathbf{A}\mathbf{p}_j, \quad \bar{\mathbf{r}}_{j+1} = \bar{\mathbf{r}}_j - \alpha_j \mathbf{A}^T \bar{\mathbf{p}}_j$$

$$\mathbf{z}_{j+1} = \mathbf{M}^{-1}\mathbf{r}_{j+1}, \quad \bar{\mathbf{z}}_{j+1} = (\mathbf{M}^T)^{-1}\bar{\mathbf{r}}_{j+1} \quad \text{// } \textit{Preconditioner for BiCG}$$

$$\beta_j = (\bar{\mathbf{r}}_{j+1}, \mathbf{z}_{j+1}) / (\bar{\mathbf{r}}_j, \mathbf{z}_j)$$

$$\mathbf{p}_{j+1} = \mathbf{z}_{j+1} + \beta_j \mathbf{p}_j, \quad \bar{\mathbf{p}}_{j+1} = \bar{\mathbf{z}}_{j+1} + \beta_j \bar{\mathbf{p}}_j$$

end do

end PBiCG

There are many other algorithms for general linear systems. The most versatile is the GMRES method, and its flexible variant FGMRES. For complete details of these methods, refer to Saad (2003).

26.5 MULTIGRID METHOD

Another important class of accelerated methods are multi-grid or multilevel methods. The basis of multi-grid acceleration is the observation that the low frequency components of error become high frequency components on a coarser grid (on which these can be easily reduced using a relaxation or smoothing scheme such as Jacobi or Gauss-Seidel iterations). The main components of multi-grid algorithm are:

- Restriction operator **R** which maps (restricts) residuals on a finer grid to coarser grid.
- Prolongation operator **P** which extends (interpolates) solution on a coarser grid to a finer grid.
- Smoothing procedure, SMOOTH()

The prolongation and restriction operators depend on underlying differential equation. Further, these also depend on the chosen multi-grid strategy which may itself be classified as (a) geometric multi-grid (based on a sequence of nested grids) or (b) algebraic multi-grid (in which multi-grid components are generated based on a purely algebraic procedure). Irrespective of the choice, the basic multi-grid method can be algorithmically outlined as follows.

Let the superscript (j) denote the quantity associated with the jth grid level. Thus, $\mathbf{A}^{(j)}$ is the matrix obtained by discretization (or algebraic procedure) on the jth grid. Let $\mathbf{R}^{(j)}$ and $\mathbf{P}^{(j)}$ be the restriction and prolongation operators respectively on the jth grid level. Further, let us define a smoothing procedure denoted as

$$\bar{\mathbf{w}}^{(j)} = \text{SMOOTH}^{\nu}(\mathbf{w}^{(j)}, \mathbf{A}^{(j)}, \mathbf{r}^{(j)})$$

which essentially represents the use of ν iterations of a classical iteration method such as Gauss–Seidel or Jacobi method applied to the linear system $\mathbf{A}^{(j)}\mathbf{x}^{(j)} = \mathbf{r}^{(j)}$ starting with the initial approximation $\mathbf{w}^{(j)}$. We can represent the standard V-cycle multi-grid as follows. Starting with an approximation $\mathbf{x}_m^{(j)}$ at iteration m , the algorithm obtains the next iterate

$$\mathbf{x}_{m+1}^{(j)} = \text{VCYC}(\mathbf{r}_m^{(j)}, j, \mathbf{x}_m^{(j)})$$

where the algorithm VCYC consists of the following recursive procedure:

- If j = 0, then use a direct solver to solve $\mathbf{A}^{(0)}\mathbf{x}^{(0)} = \mathbf{r}^{(0)}$
- Else
  - *Pre-smoothing:* $\bar{\mathbf{x}}_m^{(j)} = \text{SMOOTH}^{\nu_1}(\mathbf{x}_m^{(j)}, \mathbf{A}^{(j)}, \mathbf{r}_m^{(j)})$
  - *Defect computation:* $\bar{\mathbf{r}}_m^{(j)} = \mathbf{r}_m^{(j)} - \mathbf{A}^{(j)}\bar{\mathbf{x}}_m^{(j)}$
  - *Restriction:* $\bar{\mathbf{r}}_m^{(j-1)} = \mathbf{R}^{(j-1)}\bar{\mathbf{r}}_m^{(j)}$
  - *Coarse grid correction:* $\mathbf{x}_{m'}^{(j)} = \mathbf{x}_m^{(j)} + \mathbf{P}^{(j)}\text{VCYC}(\bar{\mathbf{r}}_m^{(j-1)}, j-1, \mathbf{0})$
  - *Post-smoothing*: $\mathbf{x}_{m+1}^{(j)} = \text{SMOOTH}^{\nu_2}(\mathbf{x}_{m'}^{(j)}, \mathbf{A}^{(j)}, \mathbf{r}_m^{(j)})$
- End If

For further details of multi-grid methods, see Saad (2003) and Trottenberg et al. (2000).

REFERENCES/FURTHER READING

Chung, T. J. (2010). *Computational Fluid Dynamics*. 2nd Ed., Cambridge University Press, Cambridge, UK.

Ferziger, J. H. And Perić, M. (2003). *Computational Methods for Fluid Dynamics*. Springer.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2002). *Numerical Recipes in C++*. Cambridge University Press, Cambridge.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia.

Trottenberg, U., C. W. Oosterlee, C. W. and A. Schüller (2000). *Multigrid*, Academic Press, London.

WEB RESOURCES

http://www.netlib.org
http://www.mgnet.org