Lecture 8: Fast Linear Solvers (Part 4)

Iterative Methods for Solving Linear Systems

- Consider to solve $A\boldsymbol{x} = \boldsymbol{b}$ with $A \in R^{n \times n}$ and $\boldsymbol{b} \in R^n$.
- In practice, iteration terminates when residual $||\boldsymbol{b} - Ax||$ is as small as desired.
- Let $B \in R^{n \times n}$ be a non-singular matrix
- Rewrite $A\boldsymbol{x} = \boldsymbol{b}$ as $\big(B + (A - B)\big)\boldsymbol{x} = \boldsymbol{b}$
  - $\boldsymbol{x} = B^{-1}(B - A)\boldsymbol{x} + B^{-1}\boldsymbol{b}$, which is a fixed-point equation.
  - One uses a iteration for the solution of the fixed-point iteration:

$$\boldsymbol{x}^{(k+1)} = B^{-1}(B - A)\boldsymbol{x}^{(k)} + B^{-1}\boldsymbol{b}, \qquad k \in N_0 \text{ where}$$

$x^{(0)}$ is an arbitrary initial guess.

Splitting Matrix B

Algorithmic Conditions for B

- B^{-1} must exist.
- The sequence $(x_i)^{(k)}$ converges for $1 \leq i \leq n$ as $k \to \infty$. Ideally, this convergences should be fast.
- Efficient solution of the system $B\boldsymbol{v} = \boldsymbol{g}$
- Efficient computation of $(B - A)\boldsymbol{v}$

Lipschitz Continuity

- Define $F(\boldsymbol{x}) = B^{-1}(B - A)\boldsymbol{x} + B^{-1}\boldsymbol{b}$
- $\left||F(\boldsymbol{x}) - F(\boldsymbol{y})|\right| = \left||B^{-1}(B - A)(\boldsymbol{x} - \boldsymbol{y})|\right| \leq \left||B^{-1}(B - A)|\right|\left||\boldsymbol{x} - \boldsymbol{y}|\right| \equiv \delta\left||\boldsymbol{x} - \boldsymbol{y}|\right|$, $\boldsymbol{x}, \boldsymbol{y} \in R^n$

  With $\delta \coloneqq ||B^{-1}(B - A)||$

Convergence

Theorem. Let $||\cdot||$ be a vector norm in R^n and $\left||C|\right| := sup_{\boldsymbol{x}\in R^n} \frac{||C\boldsymbol{x}||}{||\boldsymbol{x}||}$, $C \in R^{n\times n}$ the induced matrix norm. Assume $\delta := \left||B^{-1}(B-A)|\right| < 1$, then the sequence $(x_i)^{(k)}$ converges for all initial values $\boldsymbol{x}^{(0)}$ to the solution $\boldsymbol{x} \in R^n$ of $A\boldsymbol{x} = \boldsymbol{b}$. The error is bounded by

$$||\boldsymbol{x}^{(k+1)} - \boldsymbol{x}|| \leq \frac{\delta^k}{1-\delta} ||\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}||$$

Jacobi Method

Decompose matrix $A = [a_{ij}]$ into

$A = D + L + U,\ L, D, U \in R^{n\times n}$

$D = diag(a_{11}, a_{22}, \ldots, a_{nn})$ is a diagonal matrix and

$$L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix} \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

- Choose $B = D$, $D\boldsymbol{x} = -(L+U)\boldsymbol{x} + \boldsymbol{b}$
- The Jacobi method can be written as

$$\boldsymbol{x}^{(k+1)} = D^{-1}\left(\boldsymbol{b} - (L+U)\boldsymbol{x}^{(k)}\right)$$

- Jacobi method requires nonzero diagonal entries, which can be obtained by permuting rows and columns.
- Requires storage for both $\boldsymbol{x}^{(k+1)}$ and $\boldsymbol{x}^{(k)}$.
- components of new iterate do not depend on each other. So they can be computed in parallel.
- Define $T_j = -D^{-1}(L+U), \boldsymbol{c}_j = D^{-1}\boldsymbol{b}$

Jacobi method can be written as

$$\boldsymbol{x}^{(k+1)} = T_j \boldsymbol{x}^{(k)} + \boldsymbol{c}_j$$

Algorithm of Jacobi Method

- Choose initial vector $\boldsymbol{x}^0 \in R^n$

  Set $k = 1$

  while $(k \leq N)$ do

      for $i = 1$ **to** n

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j \neq i}^{n} a_{ij} xo_j\right)$$

      end for

      if $\left\|\boldsymbol{x} - \boldsymbol{xo}\right\| < TOL$ **stop**.

      Set $k = k + 1$

      for $i = 1$ **to** n

          $xo_i = x_i$

      end for

  end while

Gauss-Seidel Method

- Choose $B = D + L$, $(D + L)\boldsymbol{x} = -(U)\boldsymbol{x} + \boldsymbol{b}$
- The Gauss-Seidel method can be written as

$$\boldsymbol{x}^{(k+1)} = (D)^{-1}\left(\boldsymbol{b} - U\boldsymbol{x}^{(k)} - L\boldsymbol{x}^{(k+1)}\right) \text{ or}$$

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

- Gauss-Seidel requires nonzero diagonal entries
- Gauss-Seidel does not need to duplicate storage for $\boldsymbol{x}$, since component values of $\boldsymbol{x}$ can be overwritten as they are computed.
- Computing $x_j^{(k+1)}$ depends on previous $x_{j-1}^{(k+1)}$, $x_{j-2}^{(k+1)}$, … so they must be computed successively.
- Gauss-Seidel converges about twice as fast as Jacobi method.
- Define $T_g = -(D+L)^{-1}U,\ \boldsymbol{c}_g = (D+L)^{-1}\boldsymbol{b}$

Gauss-Seidel method can be written as

$$\boldsymbol{x}^{(k+1)} = T_g\boldsymbol{x}^{(k)} + \boldsymbol{c}_g$$

Algorithm of Gauss-Seidel

- Choose initial vector $\boldsymbol{x}^0 \in R^n$

Set $k = 1$

while $(k \leq N)$ do

for $i = 1$ **to** n

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij} xo_j - \sum_{j=1}^{i-1} a_{ij} x_j\right)$$

end for

if $\left|\left|\boldsymbol{x} - \boldsymbol{xo}\right|\right| < TOL$ **stop**.

Set $k = k + 1$

for $i = 1$ **to** n

$xo_i = x_i$

end for

end while

- M matrices
  - A matrix $A = [a_{ij}] \in R^{n\times n}$ is a M-matrix if the following conditions are satisfied
    - $a_{ij} \leq 0,\ i, j = 1, \dots, n, \quad i \neq j$.
    - $A^{-1} \geq 0$ exists.
- If a matrix A is strongly diagonally dominant, then Gauss-Seidel and Jacobi method converges.
- Let A be M-matrix. Then Gauss-Seidel and Jacobi method converges.
- The spectral radius of Gauss-Seidel method is smaller than that of Jacobi method if both methods converges.

SOR Method

- Successive over-relaxation (SOR) method computes next iterate as

  $\boldsymbol{x}^{(k+1)} = (1-\omega)\boldsymbol{x}^{(k)} + \omega(\boldsymbol{x}_g^{(k+1)})$ where $\boldsymbol{x}_g^{(k+1)}$ is next iterate computed by Gauss-Seidel method

- ω is fixed relaxation parameter.
  - SOR can converge only if $0 < \omega < 2$.
  - $\omega > 1$ gives over-relaxation; while $\omega < 1$ gives under-relaxation.

- Using matrix notation, SOR can be written as

  $$(D + \omega L)\boldsymbol{x}^{(k+1)} = [(1-\omega)D - \omega U]\boldsymbol{x}^{(k)} + \omega \boldsymbol{b}$$

Parallelization of Jacobi and Gauss-Seidel Method

- Parallelization of Jacobi method is straight forward in contrast to Gauss-Seidel method
- Jacobi and Gauss-Seidel method are rarely used in practical applications due to slow convergence
- Krylov space methods are more often used
- Jacobi and Gauss-Seidel method are often used as preconditioners for Krylov space methods for smoothers for multi-grid methods.

Parallel Jacobi Method

- Decompose the matrix $A = [a_{ij}]$ into sub-matrices and use 2D block mapping.

while error > TOL

On each process , compute all own components $(a_{ij}x_j^{(k)})$ of the current iteration .

Tasks in each row of the task grid perform a sum-reduction to compute $\sum_{j \neq i} a_{ij}x_j^{(k)}$

After the sum-reduction, compute $b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}$ among the tasks in the first column of the task grid and these tasks compute $x_j^{(k+1)}$

Distribute $x_j^{(k+1)}$ on task grid