Problem Solving in Computer Science: Lecture 11

Scribe notes by Ali Galip Bayrak

October 23, 2008

Introduction

The lecture started with a quick progress report of each group for Project 2. Then, Verena introduced three numerical methods to solve linear system of equations: *Jacobi*, *Gauss-Seidel* and *Successive Over-Relaxation*. Finally, she gave an example for *Sparse Storage Schemes* and finished the lecture.

1 Ideas for Project 2

- Use macro-states and try to solve each macro.
- Use ordinary differential equations to approximate Markov chains.

  For $A + B \xrightarrow{c} C$, we have

  - $\frac{d}{dt}A(t) = -c \cdot A(t) \cdot B(t)$
  - $\frac{d}{dt}B(t) = -c \cdot A(t) \cdot B(t)$
  - $\frac{d}{dt}C(t) = c \cdot A(t) \cdot B(t)$.

  For $A + B \underset{d}{\overset{c}{\rightleftharpoons}} C$, we have

  - $\frac{d}{dt}A(t) = -c \cdot A(t) \cdot B(t) + d \cdot C(t)$
  - $\frac{d}{dt}B(t) = -c \cdot A(t) \cdot B(t) + d \cdot C(t)$
  - $\frac{d}{dt}C(t) = c \cdot A(t) \cdot B(t) - d \cdot C(t)$.
- All possible states can be generated by a recursive algorithm. For example, if $m = 4$, the possible states are:

  - 1 1 1 1
  - 1 2 1
  - 1 3
  - 4
  - 2 2

  Also, *Dollar Change Problem* can be useful for generating the states.
- The vector $E(t)$ holding the expected number of speckles for each speckle size at time t, converges to a value when t goes to infinity. By solving the equation $E(t) = E(t + \Delta t)$, this vector can be determined.

2 Numerical Methods to Solve Linear System of Equations

Problem: Given a matrix A and a vector b, find the vector x which satisfies the equation $A \cdot x = b$.

2.1 Jacobi Iteration

The idea: Decompose A in order to have x on both sides of the equation.

Method: Let $A = D - (L + U)$ where D, L and U represent the diagonal, strictly lower triangular and strictly upper triangular parts of A, respectively. Then, we can rewrite the equation as

$$D \cdot x = (L + U) \cdot x + b \tag{1}$$

If D is invertible, then we have

$$x = D^{-1} \cdot (L + U) \cdot x + D^{-1} \cdot b \tag{2}$$

Now, we can iterate x:

$$x^{(k+1)} = \underbrace{D^{-1} \cdot (L + U)}_{IterationMatrix} \cdot x^{(k)} + D^{-1} \cdot b \tag{3}$$

where $x^{(k)}$ denotes the k^{th} iteration of x vector. Each member of the x vector can be determined by the equation

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)}), i = \{1, 2, .., n\} \tag{4}$$

This method converges if the spectral norm of the iteration matrix is smaller than 1.

2.2 Gauss-Seidel Iteration

The idea: Similar to Jacobi, but uses $(k+1)^{th}$ approximation of x_j for $j = \{1, 2, .., i-1\}$ during the computation of $x_i^{(k+1)}$.

Method: Replace Equation 4 by this:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}), i = \{1, 2, .., n\} \tag{5}$$

2.3 Successive Over-Relaxation

The idea: Similar to Gauss-Seidel, but this time introduces a scalar ω value.

Method: After multiplying each side of the equation $A = D - (L + U)$ of Gauss-Seidel method with ω, we have

$$\omega A = (D - \omega L) - (\omega U + (1 - \omega) D) \tag{6}$$

$$(D - \omega L) \cdot x^{(k+1)} = [\omega U + (1 - \omega) D] \cdot x^{(k)} + \omega b \tag{7}$$

If $(D - \omega L)$ is invertible, we can rewrite the equation as:

$$x^{(k+1)} = \underbrace{(D - \omega L)^{-1} \cdot [\omega U + (1 - \omega)D]}_{Iterationmatrix} \cdot x^{(k)} + \omega(D - \omega L)^{-1} b \tag{8}$$

Playing with ω value, where $0 < \omega < 2$, we can affect the convergence speed. The method is called *successive over-relaxation* if $\omega > 1$, and *successive under-relaxation* if $\omega < 1$.

2.4 Test for Convergence for the Methods

For the methods described above, you can check the convergence by checking

$$\left\| x^{(k)} - x^{(k-m)} \right\| < \epsilon \tag{9}$$

There is no rule for determining m, but it is suggested that

- $m = 5$, if $k < 100$
- $m = 10$, if $100 \leq k < 500$
- $m = 20$, if $500 \leq k < 1000$
- $m = 50$, if $k \geq 1000$

If the x vector consists of very small values, we can test convergence by:

$$\max \left(\frac{\left| x_i^{(k)} - x_i^{(k-m)} \right|}{x_i^{(k)}} \right) < \epsilon \tag{10}$$

3 Sparse Storage Schemes

In this section, we give an algorithm for multiplication of a sparse matrix with a vector. To explain with an example, assume that A is a sparse matrix and we want to find the result of $A \cdot x = z$, where x and z are vectors.

$$A = \begin{pmatrix} -2.1 & 0 & 1.7 & 0.4 \\ 0.8 & -0.8 & 0 & 0 \\ 0.2 & 1.5 & -1.7 & 0 \\ 0 & 0.3 & 0.2 & -0.4 \end{pmatrix}$$

Let us define three vectors:

- *aa*: non-zero elements of A rowwise ([-2.1,1.7,0.4,0.8,-0.8,0.2,...])
- *ia*: row number of each element of *aa* ([1,1,1,2,2,3,...])
- *ja*: column number of each element of *aa* ([1,3,4,1,2,1,...])

Algorithm 1 SparseMultiplication

Given: aa, ia and ja of a matrix A and a vector x
Result: a vector z
$z \leftarrow 0$ {Initialize z as zero}
for $next \leftarrow 1$ to $n_a a$ **do**
  $nrow \leftarrow ia_{next}$
  $ncol \leftarrow ja_{next}$
  $z_{nrow} \leftarrow z_{nrow} + aa_{next} * x_{ncol}$
end for
return z

Algorithm 2 SparseMultiplicationUnordered

Given: aa, ia and ja of a matrix A and a vector x
Result: a vector z
$next \leftarrow 1$
for $i \leftarrow 1$ to n_z **do**
  $sum \leftarrow 0$
  while $ia_{next} = i$ **do**
    $ncol \leftarrow ja_{next}$
    $sum \leftarrow sum + aa_{next} * x_{ncol}$
    $next \leftarrow next + 1$
  end while
  $z_i \leftarrow sum$
end for
return z

We can use Algorithm 1 to do the multiplication. If the rows of aa are not ordered, then we can use Algorithm 2.