Mathematical Software

November 26, 2013

- for and while

  Example 1 *Let* $x_0 = 1.5$. *Define*

  $$x_n = \frac{1}{2}\left(10 - x_{n-1}^3\right)^{1/2}, \quad \text{for } n = 1, 2, \ldots.$$

  Note that $\{x_n\}$ *converges to the root of* $x^3 + 4x^2 - 10 = 0$.

  Example 2 (infinite loop) *Consider the following recurrence algorithm*

  $$\begin{cases} x_0 = 1, & x_1 = \frac{1}{3} \\ x_{n+1} = \frac{13}{3}x_n - \frac{4}{3}x_{n-1} \end{cases}$$

  for computing the sequence of $\{x_n = (\frac{1}{3})^n\}$.

  Example 3 *Let* $x_0 \in [0, 2\pi]$. *Define*

  $$x_n = \pi + \frac{1}{2}\sin\left(\frac{x_{n-1}}{2}\right), \quad \text{for } n = 1, 2, \ldots.$$

  Note that $\{x_n\}$ *converges to the root of* $x = \pi + \frac{1}{2}\sin(x/2)$.

  Question: Is there any other root for $x = \pi + \frac{1}{2}\sin(x/2)$? (plot the figure)

- array

  Example 4 *Solve following linear system*

  $$\begin{bmatrix} 2 & 3 & 4 & \cdots & n+1 \\ & 4 & 5 & \cdots & n+2 \\ & & \ddots & \ddots & \vdots \\ & & & 2n-2 & 2n-1 \\ & & & & 2n \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix}.$$

  Note that $u_{ij} = i + j$, *for* $j = 1, \ldots, n$, $i = 1, \ldots, j$.

Backward substitution: Solve the linear system $Ux = b$:

$$\begin{array}{ccccccccc} u_{11}x_1 & + & u_{12}x_2 & + & \cdots & + & u_{1n}x_n & = & b_1, \\ & & u_{22}x_2 & + & \cdots & + & u_{2n}x_n & = & b_2, \\ & & & & & & \vdots & & \\ & & & & & & u_{nn}x_n & = & b_n \end{array}$$

- Solving the nth equation for x_n gives

$$x_n = \frac{b_n}{u_{nn}}.$$

- Solving the $(n-1)$th equation for x_{n-1} and using the value for x_n yields

$$x_{n-1} = \frac{b_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}}.$$

- In general,

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}}, \ \forall\ i = n-1, n-2, \ldots, 1.$$

Require: Nonsingular upper triangular matrix U and right hand side vector b.
Ensure: The solution x.
1: **for** $i = n, \ldots, 1$ **do**
2: Set $d = 0$;
3: **for** $j = i+1, \ldots, n$ **do**
4: Compute $d = d + U(i,j) * x(j)$;
5: **end for**
6: Compute $x(i) = (b(i) - d)/U(i,i)$.
7: **end for**

Algorithm 1: Backward Substitution

Forward Substitution: When a linear system $Lx = b$ is lower triangular of the form

$$\begin{bmatrix} \ell_{11} & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$

where all diagonals $\ell_{ii} \neq 0$, x_i can be obtained by the following procedure

$$\begin{aligned} x_1 &= b_1/\ell_{11}, \\ x_2 &= (b_2 - \ell_{21}x_1)/\ell_{22}, \\ x_3 &= (b_3 - \ell_{31}x_1 - \ell_{32}x_2)/\ell_{33}, \\ &\vdots \\ x_n &= (b_n - \ell_{n1}x_1 - \ell_{n2}x_2 - \cdots - \ell_{n,n-1}x_{n-1})/\ell_{nn}. \end{aligned}$$

The general formulation for computing x_i is

$$x_i = \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} x_j \right) \Big/ \ell_{ii}, \qquad i = 1, 2, \ldots, n.$$

Require: Nonsingular lower triangular matrix L and right hand side vector b.
Ensure: The solution x.
1: **for** $i = 1, \ldots, n$ **do**
2: Compute $x(i) = b(i)/L(i,i)$;
3: Update $b(i+1:n) = b(i+1:n) - x(i)L(i+1:n, i)$;
4: **end for**

Algorithm 2: Forward Substitution (Column version)

- Solve following linear system

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

where

$$b_i = \frac{1}{i}, \quad \text{for } i = 1, \ldots, 3n$$

and

$$A = C^*C + I \in \mathbb{C}^{3n \times 3n}$$

in which C is defined as follows.

(a) Construct the matrix $(C^*C + I)$ where

$$C = \begin{bmatrix} 0 & \frac{1}{h_3} C_3 & \frac{-1}{h_2} \text{diag}\,(C_2, \cdots, C_2) \\ \frac{-1}{h_3} C_3 & 0 & \frac{1}{h_1} \text{diag}\,(C_1, \cdots, C_1) \\ \frac{1}{h_2} \text{diag}\,(C_2, \cdots, C_2) & \frac{-1}{h_1} \text{diag}\,(C_1, \cdots, C_1) & 0 \end{bmatrix}$$

is a $3n \times 3n$ complex matrix. Here,

$$C_1 = \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \\ e^{\imath 2\pi \mathrm{k}_1} & & & -1 \end{bmatrix} \in \mathbb{C}^{n_1 \times n_1}, \tag{2a}$$

$$C_2 = \begin{bmatrix} -I_{n_1} & I_{n_1} & & \\ & \ddots & \ddots & \\ & & -I_{n_1} & I_{n_1} \\ e^{\imath 2\pi \mathrm{k}_2} I_{n_1} & & & -I_{n_1} \end{bmatrix} \in \mathbb{C}^{(n_1 n_2) \times (n_1 n_2)}, \tag{2b}$$

$$C_3 = \begin{bmatrix} -I_{n_1 \times n_2} & I_{n_1 \times n_2} & & \\ & \ddots & \ddots & \\ & & -I_{n_1 \times n_2} & I_{n_1 \times n_2} \\ e^{\imath 2\pi \mathrm{k}_3} I_{n_1 \times n_2} & & & -I_{n_1 \times n_2} \end{bmatrix} \in \mathbb{C}^{n \times n}, \tag{2c}$$

$\mathbf{k} = (\mathrm{k}_1, \mathrm{k}_2, \mathrm{k}_3)$, $h_1 = 1/n_1$, $h_2 = 1/n_2$, and $h_3 = 1/n_3$ denote the mesh length along the x, y and z axial directions, respectively. The constants n_1, n_2 and n_3 are the numbers of the grid points in the x, y and z directions, respectively, and $n = n_1 n_2 n_3$.

Require: Nonsingular matrix A and right hand side vector b.
Ensure: This algorithm implements the Gaussian elimination procedure to reduce A to upper triangular and modify the entries of b accordingly.

```
for k = 1, ..., n − 1 do
  Let p be the smallest integer with k ≤ p ≤ n and a_pk ≠ 0.
  If ∄ p, then stop.
  If p ≠ k, then perform (E_p) ↔ (E_k).
  for i = k + 1, ..., n do
    Compute t = A(i, k)/A(k, k);
    Set A(i, k) = 0;
    Update b(i) = b(i) − t × b(k);
    for j = k + 1, ..., n do
      Update A(i, j) = A(i, j) − t × A(k, j);
    end for
  end for
end for
```

Algorithm 3: Gaussian elimination

(b) Use Gaussian elimination to solve linear system (1).

1. Use Algorithm 3 (Gaussian elimination) to reduce A to an upper triangular matrix and modify the entries of b accordingly. Plot

the CPU times for reducing A to upper triangular with various n.

2. Use backward substitution to solve the upper triangular linear system in (b.1). Plot the CPU times for solving such linear system with various n.
3. Compare the CPU times for using left matrix divide "$A \setminus b$" in MATLAB with that in (b.1) and (b.2).

(c) Use Jacobi method to solve linear system (1).
Given an initial vector $x^{(0)}$, rewrite the linear system as:

$$\begin{aligned} a_{11}x_1^{(k)} + a_{12}x_2^{(k-1)} + a_{13}x_3^{(k-1)} + \cdots + a_{1n}x_n^{(k-1)} &= b_1 \\ a_{21}x_1^{(k-1)} + a_{22}x_2^{(k)} + a_{23}x_3^{(k-1)} + \cdots + a_{2n}x_n^{(k-1)} &= b_2 \\ &\vdots \\ a_{n1}x_1^{(k-1)} + a_{n2}x_2^{(k-1)} + a_{n3}x_3^{(k-1)} + \cdots + a_{nn}x_n^{(k)} &= b_n. \end{aligned}$$

If we decompose the coefficient matrix A as

$$A = L + D + U,$$

where D is the diagonal part, L is the strictly lower triangular part, and U is the strictly upper triangular part, of A, then we derive the iterative formulation for Jacobi method:

$$x^{(k)} = -D^{-1}(L+U)x^{(k-1)} + D^{-1}b.$$

Require: Given $x^{(0)}$, tolerance TOL, maximum number of iteration M.
Ensure: The solution x.
1: Set $k = 1$.
2: Compute $x = -D^{-1}(L+U)x^{(0)} + D^{-1}b$.
3: **while** $k \leq M$ and $\|x - x^{(0)}\|_2 \geq TOL$ **do**
4: Set $k = k + 1$, $x^{(0)} = x$;
5: Compute $x = -D^{-1}(L+U)x^{(0)} + D^{-1}b$;
6: **end while**

Algorithm 4: Jacobi method

• Use Algorithm 4 with initial vector $x^{(0)} = [1, \cdots, 1]^\top$ to solve linear system (1). Plot the CPU times and iteration numbers k for solving such linear system with various n.

(d) Use Gauss-Seidel method to solve linear system (1).
Given an initial vector $x^{(0)}$, rewrite the linear system as:

$$\begin{aligned} a_{11}x_1^{(k)} + a_{12}x_2^{(k-1)} + a_{13}x_3^{(k-1)} + \cdots + a_{1n}x_n^{(k-1)} &= b_1 \\ a_{21}x_1^{(k)} + a_{22}x_2^{(k)} + a_{23}x_3^{(k-1)} + \cdots + a_{2n}x_n^{(k-1)} &= b_2 \\ a_{31}x_1^{(k)} + a_{32}x_2^{(k)} + a_{33}x_3^{(k)} + \cdots + a_{3n}x_n^{(k-1)} &= b_3 \\ &\vdots \\ a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + a_{n3}x_3^{(k)} + \cdots + a_{nn}x_n^{(k)} &= b_n. \end{aligned}$$

This improvement induce the Gauss-Seidel method. The iteration of the Gauss-Seidel method is defined as follows:

$$x^{(k)} = -(D+L)^{-1}Ux^{(k-1)} + (D+L)^{-1}b.$$

Require: Given $x^{(0)}$, tolerance TOL, maximum number of iteration M.
Ensure: The solution x.
1: Set $k = 1$.
2: Compute $x = -(D+L)^{-1}Ux^{(0)} + (D+L)^{-1}b$.
3: **while** $k \leq M$ and $\|x - x^{(0)}\|_2 \geq TOL$ **do**
4: Set $k = k+1$, $x^{(0)} = x$;
5: Compute $x = -(D+L)^{-1}Ux^{(0)} + (D+L)^{-1}b$;
6: **end while**

Algorithm 5: Gauss-Seidel method

1. Use MATLAB functions "triu(A,1)" and "tril(A,-1)" to extract the strictly upper and lower triangular parts of A, respectively.
2. Use Algorithm 5 with initial vector $x^{(0)} = [1, \cdots, 1]^\top$ to solve linear system (1). Plot the CPU times and iteration numbers k for solving such linear system with various n.
3. Compare the results produced by Jacobi and Gauss-Seidel methods.

(e) Use SSOR method to solve linear system (1).
Given an initial vector $x^{(0)}$, rewrite the linear system as:

$$\begin{aligned}
a_{11}x_1^{(k)} + a_{12}x_2^{(k-1)} + a_{13}x_3^{(k-1)} + \cdots + a_{1n}x_n^{(k-1)} &= b_1 \\
a_{21}x_1^{(k)} + a_{22}x_2^{(k)} + a_{23}x_3^{(k-1)} + \cdots + a_{2n}x_n^{(k-1)} &= b_2 \\
a_{31}x_1^{(k)} + a_{32}x_2^{(k)} + a_{33}x_3^{(k)} + \cdots + a_{3n}x_n^{(k-1)} &= b_3 \\
&\vdots \\
a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + a_{n3}x_3^{(k)} + \cdots + a_{nn}x_n^{(k)} &= b_n.
\end{aligned}$$

Let the approximate solution $\mathbf{x}^{(k,i)}$ produced by Gauss-Seidel method be defined by

$$\mathbf{x}^{(k,i)} = \left[x_1^{(k)}, \ldots, x_{i-1}^{(k)}, x_i^{(k-1)}, \ldots, x_n^{(k-1)}\right]^T$$

and

$$r_i^{(k)} = \left[r_{1i}^{(k)}, r_{2i}^{(k)}, \ldots, r_{ni}^{(k)}\right]^T = b - A\mathbf{x}^{(k,i)}$$

be the corresponding residual vector. Then the ith component of $r_i^{(k)}$ is

$$r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} - a_{ii}x_i^{(k-1)},$$

so

$$a_{ii}x_i^{(k-1)} + r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} = a_{ii}x_i^{(k)}.$$

Consequently, the Gauss-Seidel method can be characterized as choosing $x_i^{(k)}$ to satisfy

$$x_i^{(k)} = x_i^{(k-1)} + \frac{r_{ii}^{(k)}}{a_{ii}}.$$

Relaxation method is modified the Gauss-Seidel procedure to

$$\begin{aligned} x_i^{(k)} &= x_i^{(k-1)} + \omega \frac{r_{ii}^{(k)}}{a_{ii}} \\ &= x_i^{(k-1)} + \frac{\omega}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} - a_{ii}x_i^{(k-1)} \right] \\ &= (1-\omega)x_i^{(k-1)} + \frac{\omega}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} \right] \end{aligned} \quad (3)$$

for certain choices of positive ω. These methods are called for

$\omega < 1$: under relaxation,
$\omega = 1$: Gauss-Seidel method,
$\omega > 1$: over relaxation.

Over-relaxation methods are called SOR (Successive over-relaxation). To determine the matrix of the SOR method, we rewrite (3) as

$$a_{ii}x_i^{(k)} + \omega \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} = (1-\omega)a_{ii}x_i^{(k-1)} - \omega \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} + \omega b_i,$$

so that if $A = L + D + U$, then we have

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

Theorem 1 (Ostrowski-Reich) *If A is positive definite and the relaxation parameter ω satisfying $0 < \omega < 2$, then the SOR iteration converges for any initial vector $x^{(0)}$.*

Let A be symmetric and $A = D + L + L^T$. The idea is in fact to implement the SOR formulation twice, one forward and one backward, at each iteration. That is, SSOR method defines

$$(D + \omega L)x^{(k-\frac{1}{2})} = \left[(1-\omega)D - \omega L^T\right] x^{(k-1)} + \omega b, \quad (4)$$

$$(D + \omega L^T)x^{(k)} = \left[(1-\omega)D - \omega L\right] x^{(k-\frac{1}{2})} + \omega b. \quad (5)$$

Define

$$\begin{cases} M_\omega := D + \omega L, \\ N_\omega := (1-\omega)D - \omega L^T. \end{cases}$$

Then from the iterations (4) and (5), it follows that

$$\begin{aligned} x^{(k)} &= \left(M_\omega^{-T} N_\omega^T M_\omega^{-1} N_\omega\right) x^{(k-1)} + \omega \left(M_\omega^{-T} N_\omega^T M_\omega^{-1} + M_\omega^{-T}\right) b \\ &\equiv T(\omega) x^{(k-1)} + M(\omega)^{-1} b, \end{aligned}$$

where

$$M(\omega) = \frac{1}{\omega(2-\omega)} (D + \omega L) D^{-1} \left(D + \omega L^T\right).$$

1. Take $x^{(0)} = [1, \cdots, 1]^\top$ as an initial vector.
2. Use MATLAB functions "triu(A,1)" and "tril(A,-1)" to extract the strictly upper and lower triangular parts of A, respectively.
3. Fixed $n_1 = 10$ and uniformly took 40 values for the parameter ω in the interval $(0, 2)$, show the iteration numbers and CPU times of SSOR iterative method for each ω. Find the optimal value ω^* of the parameter ω.
4. Compare the iteration numbers and CPU times for Jacobi, Gauss-Seidel and SSOR(ω^*) iterative methods with various n.

(f) Use conjugate gradients method to solve linear system (1).

1. Use MATLAB function `pcg` without any preconditioner:

   [x, flag, relres, iter] = pcg(A, b, tol, maxit)

2. Use MATLAB function `pcg` with a given preconditioner:

   [x, flag, relres, iter] = pcg(A, b, tol, maxit, M),
   [x, flag, relres, iter] = pcg(A, b, tol, maxit, M1, M2),
   [x, flag, relres, iter] = pcg(A, b, tol, maxit, [], M2),
   [x, flag, relres, iter] = pcg(A, b, tol, maxit, MFUN).

   (i) **Jacobi method:** $A = D + (L + U), \quad M = D$

   $$x_{k+1} = -D^{-1}(L+U)x_k + D^{-1}b$$

   (ii) **Gauss-Seidel:** $A = (D + L) + U, \quad M = D + L$

   $$x_{k+1} = -(D+L)^{-1} U x_k + (D+L)^{-1} b.$$

   (iii) **SSOR:** $A = D + L + L^T, \quad M = M(\omega)$

   $$x^{(k)} = \left(M_\omega^{-T} N_\omega^T M_\omega^{-1} N_\omega\right) x^{(k-1)} + M(\omega)^{-1} b,$$

   where

   $$M(\omega) = \frac{1}{\omega(2-\omega)} (D + \omega L) D^{-1} \left(D + \omega L^T\right).$$

(iv) M may be a function handle MFUN returning $M^{-1}x$

```
[x, flag, relres, iter] = pcg(A, b, tol, maxit, ...
    @(x)precSSOR(x,omega,mtxLower,mtxdiag)
```

3. Compare the iteration numbers and CPU times for `pcg` by using different preconditioner with various n.