BARUCH, MFE

MTH 9821 Homework Three [*]

Chu, Hongshan
Wang, Jiaxi
Wei, Zhaoyue
Yin, Gongshun
Zhou, ShengQuan

September 15, 2016

[*]Team work:
Chu, Hongshan
Wang, Jiaxi #1,#2
Wei, Zhaoyue
Yin, Gongshun
Zhou, ShengQuan #1,#2,#4

1 C++ Codes

1.1 General Linear Iteration

```
std::tuple<Eigen::VectorXd, int> linear_iterate_triangular
                    (const Eigen::VectorXd & x0, //initial estimate of solution
                     const Eigen::MatrixXd & M,  //lower triangular matrix
                     const Eigen::MatrixXd & N,  //not necessarily triangular
                     const Eigen::VectorXd & b,  //right-hand side vector
                     double tol)                 //tolerance factor
{
    Eigen::VectorXd c = forward_subst(M,b);
    Eigen::VectorXd x = x0, x_old = x0;
    Eigen::VectorXd r = b-(M+N)*x0;
    double stop_resid = tol*r.norm();
    int n = 0;
    while (r.norm() > stop_resid && n < MAX_ITERATION) {
        x = c-forward_subst(M, N*x_old);
        r = b-(M+N)*x;
        x_old = x;
        n++;
    }

    return std::make_tuple(x,n);
}

std::tuple<Eigen::VectorXd, int> linear_iterate_diagonal
                    (const Eigen::VectorXd & x0, //initial estimate of solution
                     const Eigen::VectorXd & d,  //diagonal vector
                     const Eigen::MatrixXd & N,  //not necessarily triangular
                     const Eigen::VectorXd & b,  //right-hand side vector
                     double tol)                 //tolerance factor
{
    Eigen::MatrixXd D = d.asDiagonal();
    Eigen::VectorXd c = forward_subst_banded(D,0,b);
    Eigen::VectorXd x = x0, x_old = x0;
    Eigen::VectorXd r = b-(D+N)*x0;
    double stop_resid = tol*r.norm();
    int n = 0;
    while (r.norm() > stop_resid && n < MAX_ITERATION) {
        x = c-forward_subst_banded(D,0,N*x_old);
        r = b-(D+N)*x;
        x_old = x;
        n++;
    }

    return std::make_tuple(x,n);
}
```

1.2 Jacobi Iteration

```
std::tuple<Eigen::VectorXd, int> jacobi(const Eigen::MatrixXd & A,
                                        const Eigen::VectorXd & b,
                                        double tol)
{
    int n = A.rows();
    assert(n == A.cols());

    Eigen::VectorXd d = A.diagonal();
    Eigen::MatrixXd N = A;
    N.diagonal().setZero();
    // estimate of the solution
    Eigen::VectorXd x0 = b.array()/d.array();
    return linear_iterate_diagonal(x0,d,N,b,tol);
}
```

1.3 Successve Over Relaxation and Gauss-Seidel Iteration

```
std::tuple<Eigen::VectorXd, int> sor(double omega,
                                     const Eigen::MatrixXd & A,
                                     const Eigen::VectorXd & b,
                                     double tol)
{
    assert(omega>0);
    assert(omega<2);
    int n = A.rows();
    assert(n == A.cols());
    double w = 1.0/omega;
    Eigen::VectorXd d = A.diagonal();
    Eigen::MatrixXd M = A;
    M.triangularView<Eigen::StrictlyUpper>().setZero();
    M += ((w-1)*d).asDiagonal();
    Eigen::MatrixXd N = A;
    N.triangularView<Eigen::StrictlyLower>().setZero();
    N -= (w*d).asDiagonal();

    Eigen::VectorXd x0 = Eigen::VectorXd::Zero(n);
    return linear_iterate_triangular(x0,M,N,b,tol);
}

std::tuple<Eigen::VectorXd, int> gs(const Eigen::MatrixXd & A,
                                    const Eigen::VectorXd & b,
                                    double tol)
{
    return sor(1,A,b,tol); // Gauss-Seidel is SOR with Omega=1
}
```

```
// The array a[0,...,n-1][0,...,m1+m2] stores a matrix A as follows:
//
// The diagonal elements are in a[0,...n-1][m1].
// Subdiagonal elements are in a[j,...,n-1][0,...,m1-1]
// with j>0 appropriate to the number of elements on each subdiagonal;
// Superdiagonal elements are in a[0,...,j][m1+1,...,m1+m2]
// with j<n-1 appropriate to the number of elements on each superdiagonal.
//
// m1: width of lower band
// m2: width of upper band

Eigen::MatrixXd dense_from_band(const Eigen::ArrayXXd & a, int m1, int m2)
{
    int n = a.rows();
    assert( a.cols() == m1+m2+1 );

    Eigen::MatrixXd A = Eigen::MatrixXd::Zero(n,n);
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            if (i-j>m1 || j-i>m2) {
                continue;
            }
            A(i,j) = a(i,m1-i+j);
        }
    }

    return A;
}

Eigen::ArrayXXd band_from_dense(const Eigen::MatrixXd & A, int m1, int m2)
{
    int n = A.rows();
    assert(n == A.cols());

    Eigen::ArrayXXd a = Eigen::ArrayXXd::Zero(n,m1+m2+1);
    a.col(m1) = A.diagonal();

    for (int i=1; i<=m1; i++) {
        a.col(m1-i).block(i,0,n-i,1) = A.diagonal(-i);
    }

    for (int i=1; i<=m2; i++) {
        a.col(m1+i).block(0,0,n-i,1) = A.diagonal(i);
    }

    return a;
}
```

```
Eigen::VectorXd band_mult(const Eigen::ArrayXXd & a, int m1, int m2,
                          const Eigen::VectorXd & x)
{
    int width=m1+m2+1;
    int n = a.rows();
    Eigen::VectorXd y = Eigen::VectorXd::Zero(n);
    for (int i=0; i<n; i++) {
        int k=i-m1;
        int tmploop = std::min(width,n-k);
        for (int j=std::max(0,-k); j<tmploop; j++) {
            y(i) += a(i,j)*x(j+k);
        }
    }

    return y;
}

Eigen::ArrayXXd band_add(const Eigen::ArrayXXd & a, int m1, int m2,
                         const Eigen::ArrayXXd & b, int n1, int n2)
{
    int n=a.rows();
    assert(n == b.rows());
    int l1 = std::max(m1,n1);
    int l2 = std::max(m2,n2);
    int l=l1+l2+1;

    Eigen::ArrayXXd c = Eigen::ArrayXXd::Zero(n,l);
    c.col(l1) = a.col(m1) + b.col(n1);
    for (int i=1; i<=l1; i++) {
        if (i <= m1) {
            c.col(l1-i) += a.col(m1-i);
        }
        if (i <= n1) {
            c.col(l1-i) += b.col(n1-i);
        }
    }
    for (int i=1; i<=l2; i++) {
        if (i <= m2) {
            c.col(l1+i) += a.col(m1+i);
        }
        if (i <= n2) {
            c.col(l1+i) += b.col(n1+i);
        }
    }

    return c;
}
```

```
Eigen::VectorXd forward_subst_banded(const Eigen::ArrayXXd & L,
                                     const Eigen::VectorXd & b)
{
    int n = b.size();
    assert(L.rows() == n);
    int m = L.cols()-1;

    Eigen::VectorXd x = Eigen::VectorXd::Zero(n);
    for (int i=0; i<n; i++) {
        double sum = 0;
        int boundary = std::max(0,i-m);
        for (int j=boundary;j<i;j++) {
            // here is how the band storage differnt from dense storage
            sum += L(i,m-i+j)*x(j);
        }
        x(i) = (b(i)-sum)/L(i,m);
    }

    return x;
}

Eigen::VectorXd backward_subst_banded(const Eigen::ArrayXXd & U,
                                      const Eigen::VectorXd & b)
{
    int n = b.size();
    assert(U.rows() == n);
    int m = U.cols()-1;

    Eigen::VectorXd x = Eigen::VectorXd::Zero(n);
    for (int i=n-1; i>=0; i--) {
        double sum = 0;
        int boundary = std::min(n,i+m+1);
        for (int j=i+1; j<boundary; j++) {
            // here is how the band storage differnt from dense storage
            sum += U(i,j-i)*x(j);
        }
        x(i) = (b(i)-sum)/U(i,0);
    }

    return x;
}
```

```
std::tuple<Eigen::VectorXd, int> linear_iterate_triangular_banded
                     (const Eigen::VectorXd & x0, //initial estimate of solution
                      const Eigen::ArrayXXd & M,  //lower triangular banded
                      const Eigen::ArrayXXd & N,  //upper triangular banded
                      const Eigen::VectorXd & b,  //right-hand side vector
                      int m, double tol)          //band width, tolerance factor
{
    Eigen::ArrayXXd MN = band_add(M,m,0,N,0,m);
    Eigen::VectorXd c = forward_subst_banded(M,b);
    Eigen::VectorXd x = x0, x_old = x0;
    Eigen::VectorXd r = b-band_mult(MN,m,m,x0);
    double stop_resid = tol*r.norm();
    int n = 0;
    while (r.norm() > stop_resid && n < MAX_ITERATION) {
        x = c-forward_subst_banded(M, band_mult(N,0,m,x_old));
        r = b-band_mult(MN,m,m,x);
        x_old = x;
        n++;
    }

    return std::make_tuple(x,n);
}

std::tuple<Eigen::VectorXd, int> linear_iterate_diagonal_banded
                     (const Eigen::VectorXd & x0, //initial estimate of solution
                      const Eigen::ArrayXd & d,   //diagonal vector
                      const Eigen::ArrayXXd & N,  //banded
                      const Eigen::VectorXd & b,  //right-hand side vector
                      int m, double tol)          //band width, tolerance factor
{
    Eigen::ArrayXXd MN = N;
    MN.col(m) += d;
    Eigen::VectorXd c = b.array()/d;
    Eigen::VectorXd x = x0, x_old = x0;
    Eigen::VectorXd r = b-band_mult(MN,m,m,x0);
    double stop_resid = tol*r.norm();
    int n = 0;
    while (r.norm() > stop_resid && n < MAX_ITERATION) {
        x = c.array()-band_mult(N,m,m,x_old).array()/d;
        r = b-band_mult(MN,m,m,x);
        x_old = x;
        n++;
    }

    return std::make_tuple(x,n);
}
```

1.8 Banded Jacobi Iteration

```
std::tuple<Eigen::VectorXd, int> jacobi(const Eigen::ArrayXXd & A, int m,
                                        const Eigen::VectorXd & b,
                                        double tol)
{
    assert(2*m+1 == A.cols());
    Eigen::ArrayXd d = A.col(m);
    Eigen::ArrayXXd N = A;
    N.col(m).setZero();

    // estimate of the solution
    Eigen::VectorXd x0 = b.array()/d;
    return linear_iterate_diagonal_banded(x0,d,N,b,m,tol);
}
```

1.9 Banded Successve Over Relaxation and Gauss-Seidel Iteration

```
std::tuple<Eigen::VectorXd, int> sor(double omega,
                                     const Eigen::ArrayXXd & A, int m,
                                     const Eigen::VectorXd & b,
                                     double tol)
{   //the only difference from the dense implementation is band width m
    assert(omega>0);
    assert(omega<2);
    int nrow = A.rows();
    int ncol = A.cols();
    assert(2*m+1 == ncol);
    double w = 1.0/omega;

    Eigen::ArrayXd d = A.col(m);
    Eigen::ArrayXXd M = A.block(0,0,nrow,m+1);
    M.col(m) += (w-1)*d;
    Eigen::ArrayXXd N = A.block(0,m,nrow,ncol-m);
    N.col(0) -= w*d;

    Eigen::VectorXd x0 = Eigen::VectorXd::Zero(nrow);
    return linear_iterate_triangular_banded(x0,M,N,b,m,tol);
}

std::tuple<Eigen::VectorXd, int> gs(const Eigen::ArrayXXd & A, int m,
                                    const Eigen::VectorXd & b,
                                    double tol)
{
    return sor(1,A,m,b,tol);
}
```

2 Divergence of Linear Iteration

If $\rho(R) \geq 1$, then there exists an eigenvalue λ of R with $|\lambda| \geq 1$. Show that if $\rho(R) \geq 1$, then there exist iterations of the form, given x_0,

$$x_{n+1} = Rx_n + c, \quad \forall n > 0$$

which do not converge.
Proof: Note that the matrix $\mathbb{1} - R$ has the same eigenvectors as R with eigenvalues of the form $1 - \lambda$ if λ is an eigenvalue of R.

- Consider the first case where one eigenvalue $\lambda > 1$ and none of the eigenvalues are equal to 1, then the matrix $\mathbb{1} - R$ have non-zero eigenvalues and is non-singular. Thus, the matrix $\mathbb{1} - R$ has an inverse. Let

  $$b = (\mathbb{1} - R)^{-1} c,$$

  we can write the matrix iteration as

  $$x_{n+1} - b = R(x_n - b).$$

  Choose x_0 to be the eigenvector of R corresponding to eigenvalue λ, i.e. $Rx_0 = \lambda x_0$, then $R(x_0 - b) = \lambda(x_0 - b)$ and

  $$x_n - b = \lambda^n (x_0 - b)$$

  or

  $$x_n = b + \lambda^n (x_0 - b) \to \infty, \quad \text{as } n \to \infty.$$

  In other words, the iteration does not converge.

- Consider the second case where the eigenvalue $\lambda = 1$. In this case, the matrix $\mathbb{1} - R$ is singular and does not have an inverse. Again, choose norm 1 vector x_0 to be the eigenvector of R corresponding to eigenvalue $\lambda = 1$, i.e. $Rx_0 = x_0$, then

  $$x_n = x_0 + \left(\mathbb{1} + R + R^2 + \cdots + R^{n-1}\right) c, \quad n \geq 1.$$

  If the vector c has non-zero overlap with x_0 such that $x_0^T c \neq 0$,
  Multiplies x_0^T to the left side of the equation, we have:

  $$x_0^T (x_{n+1} - x_0) = n x_0^T c$$

  When $x_0^T c \neq 0$, then the right side goes to infinity when $n \to \infty$. Recall that x_0 is bounded, then

  $$\| x_{n+1} - x_0 \| \to \infty$$

  Thus, the iteration does not converge. Notice that when $c = 0$ and $\lambda = 1$, the above iteration does converge.

3 Tridiagonal Linear System

Let A be a 14×14 matrix given by

$$\begin{aligned} A_{ii} &= 2, \quad \forall i = 0, \cdots, 13, \\ A_{i+1,i} &= -1, \quad \forall i = 0, \cdots, 12, \\ A_{i,i+1} &= -1, \quad \forall i = 0, \cdots, 12 \end{aligned}$$

and let b be a column vector given by

$$b_i = i^2, \quad \forall i = 0, \cdots, 13.$$

Our goal is to solve the linear system $Ax = b$ using iterative methods. For all the problems below, use tolerance $tol = 10^{-6}$ and the initial guess vector x_0 , with $x_0(0) = 1, \forall i = 0, \cdots, 13$.

(i) Use the Jacobi iteration to solve $Ax = b$. Use first the residual-based stopping criterion and then the consecutive approximation stopping criterion. Report the solution and the number of iterations for each algorithm.
Solution: See spreadsheet.

(ii) Use the Gauss-Siedel iteration to solve $Ax = b$. Use first the residual-based stopping criterion and then the consecutive approximation stopping criterion. Report the solution and the number of iterations for each algorithm.
Solution: See spreadsheet.

(iii) Use the SOR iteration with $\omega = 1.15$ to solve $Ax = b$. Use first the residual-based stopping criterion and then the consecutive approximation stopping criterion. Report the solution and the number of iterations for each algorithm.
Solution: See spreadsheet.

(iv) For only this part of the problem, use only the residual-based stopping criterion. Solve $Ax = b$ using the SOR iteration for the following values of ω:

$$\omega = 1.02 : 0.02 : 1.98.$$

Report the number of iterations to convergence for each value of ω. Comment on the results.
Solution: See spreadsheet.

4 Finite Difference Solution to Poisson Equation

Consider the following two-dimensional second order PDE on the unit square $[0,1]\times[0,1]$:

$$-\frac{\partial^2 u}{\partial x^2}-\frac{\partial^2 u}{\partial y^2}=f(x,y),\quad \forall (x,y)\in(0,1)\times(0,1),$$

where

$$f(x,y)=(x^2+y^2-2)\sin(x)\sin(y)-2x\cos(x)\sin(y)-2y\sin(x)\cos(y)$$

with boundary conditions

$$\begin{aligned}
u(x,0)&=0, &&\forall x\in[0,1];\\
u(0,y)&=0, &&\forall y\in[0,1];\\
u(x,1)&=\frac{1}{2}(x^2+1)\sin(x)\sin(1), &&\forall x\in[0,1];\\
u(1,y)&=\frac{1}{2}(y^2+1)\sin(y)\sin(1), &&\forall y\in[0,1].
\end{aligned}$$

Note that the exact solution of this PDE is

$$u_{\text{exact}}(x,y)=\frac{1}{2}(x^2+y^2)\sin(x)\sin(y).$$

(i) Discretize this PDE using $N+2$ equidistant nodes on the $[0,1]$ interval on both the x-axis and the y-axis and central finite difference approximations. Suppose that the whole mesh is

$$\{x_1,x_2,x_3,\cdots,x_{N+1},xN+2\}\otimes\{y_1,y_2,y_3,\cdots,y_{N+1},y_{N+2}\}.$$

Exculding the boundaries, the PDE is discretized at the interior points:

$$\{x_2,x_2,x_3,\cdots,x_{N+1}\}\otimes\{y_2,y_3,\cdots,y_{N+1},\}.$$

The resulting linear system can be written as

$$T_N x=b,$$

where T_N is an $N^2\times N^2$ matrix given by

$$\begin{aligned}
A_{ii}&=4, &&\forall i=1,\cdots,N^2;\\
A_{i+1,i}&=-1, &&\forall i=1,\cdots,N^2-1,\ \text{such that } N \text{ does not divide } i;\\
A_{i,i+1}&=-1, &&\forall i=1,\cdots,N^2-1,\ \text{such that } N \text{ does not divide } i;\\
A_{i,i+N}&=-1, &&\forall i=1,\cdots,N^2-N;\\
A_{i+N,i}&=-1, &&\forall i=1,\cdots,N^2-N.
\end{aligned}$$

The right-hand-side b is a column vector of length N^2:

$$b=\begin{pmatrix} b_1\\ b_2\\ \vdots\\ b_N\end{pmatrix},$$

where each b_j is a column vector of length N, collocated at y_{j+1}:

$$b_1 = \frac{1}{(N+1)^2}\begin{pmatrix} f(x_2,y_2)\\ f(x_3,y_2)\\ \vdots \\ f(x_{N+1},y_2)\end{pmatrix} + \begin{pmatrix} u(x_1,y_2)\\ 0\\ \vdots \\ u(x_{N+2},y_2)\end{pmatrix} + \begin{pmatrix} u(x_2,y_1)\\ u(x_3,y_1)\\ \vdots \\ u(x_{N+1},y_1)\end{pmatrix},$$

$$b_j = \frac{1}{(N+1)^2}\begin{pmatrix} f(x_2,y_{j+1})\\ f(x_3,y_{j+1})\\ \vdots \\ f(x_{N+1},y_{j+1})\end{pmatrix} + \begin{pmatrix} u(x_1,y_{j+1})\\ 0\\ \vdots \\ u(x_{N+2},y_{j+1})\end{pmatrix}, \quad 2 \le j \le N-1,$$

$$b_N = \frac{1}{(N+1)^2}\begin{pmatrix} f(x_2,y_{N+1})\\ f(x_3,y_{N+1})\\ \vdots \\ f(x_{N+1},y_{N+1})\end{pmatrix} + \begin{pmatrix} u(x_1,y_{N+1})\\ 0\\ \vdots \\ u(x_{N+2},y_{N+1})\end{pmatrix} + \begin{pmatrix} u(x_2,y_{N+2})\\ u(x_3,y_{N+2})\\ \vdots \\ u(x_{N+1},y_{N+2})\end{pmatrix}.$$

(ii) Let $N \in \{2,4,8,16,32,64,128,256\}$. Solve the linear system $T_N x = b$ using Cholesky. Report the approximation error (that is, the maximum elementwise error) of your solution.
Solution: See spreadsheet.

(iii) Let $N \in \{2,4,8,16,32,64,128\}$. Use the Gauss-Siedel iteration to solve $T_N x = b$ using the residual-based stopping criterion, with tolerance $tol = 10^{-6}$ and initial guess vector $x_0 = 0$. Report the number of iterations and the approximation error of the solution.
Solution: See spreadsheet.

(iv) Let $N \in \{2,4,8,16,32,64\}$. Solve $T_N x = b$b using the SOR iteration with tolerance $tol = 10^{-6}$ and initial guess vector $x_0 = 0$, and using the residual-based stopping criterion, for the following values of ω:

$$\omega = 1.02 : .02 : 1.98.$$

Report the number of iterations to convergence for each value of ω. Comment on the results.
Solution: Numerical results see spreadsheet. An over-relaxation parameter greater than one generally accelerates the convergence of SOR iterations. According to the numerical results reported in the spreadsheet,

- the optimal over-relaxation parameter occurs somewhere between 1 and 2;
- the optimal over-relaxation parameter increases as the matrix size grows (or the grid size for the finite difference solution of the Poisson problem);
- the reduction of iteration steps with respect to Gauss-Seidel also improves as the matrix size grows, for example, the reduction ratio is about $7/11 \sim 0.64$ for $N = 2$ and the reduction ratio is about $208/3539 \sim 0.06$ for $N = 64$. For $N = 64$, this corresponds to approximately 17-fold reduction in iteration steps, which is a significant improvement of convergence rate.