The Solution of Linear Systems $\mathbf{Ax} = \mathbf{b}$
Lecture Notes

BM 531
Numerical Methods and C/C++ Programming

Ahmet Ademoglu, *PhD*
Bogazici University
Institute of Biomedical Engineering

Upper Triangular Linear Systems

Forward Elimination

$$
\begin{aligned}
u_{11}x_1 + u_{12}x_2 + u_{13}x_3 + \ldots + u_{1N-1}x_{N-1} + u_{1N}x_N &= b_1 \\
u_{22}x_2 + u_{23}x_3 + \ldots + u_{2N-1}x_{N-1} + u_{2N}x_N &= b_2 \\
u_{33}x_3 + \ldots + u_{3N-1}x_{N-1} + u_{3N}x_N &= b_3 \\
\vdots \qquad\qquad & \vdots \\
u_{N-1N-1}x_{N-1} + u_{N-1N}x_N &= b_{N-1} \\
u_{NN}x_N &= b_N
\end{aligned}
$$

Back Substitution

$x_N = b_N / u_{NN}$

$x_{N-1} = (b_{N-1} - u_{N-1N}x_N)/u_{N-1N-1}$

$x_k = (b_k - \sum_{j=k+1}^{N} u_{kj}x_j)/u_{kk}$

Lower Triangular Linear Systems

Backward Elimination

$$
\begin{aligned}
l_{11}x_1 &= b_1 \\
l_{21}x_1 + l_{22}x_2 &= b_2 \\
l_{31}x_1 + l_{32}x_2 + l_{33}x_3 &= b_3 \\
\vdots \quad &\quad \vdots \\
l_{N1}x_1 + l_{N2}x_2 + \ldots\ldots + l_{NN}x_N &= b_N
\end{aligned}
$$

Back Substitution

$x_1 = b_1/l_{11}$
$x_2 = (b_2 - l_{21}x_1)/l_{22}$
$x_k = (b_k - \sum_{j=1}^{k-1} l_{kj}x_j)/l_{kk}$

```
void lubksb(float **a, int n, int *indx, float b[])
```

Solves the set of n linear equations A·X = B. a[1..n][1..n] is LU decompositionof the matrix A the routine ludcmp. b[1..n] is input as the right-hand side vector B, and returns with the solution vector X.

Gaussian Elimination and Pivoting

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= \text{-}19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

Forward Elimination

$$\begin{array}{cccccc} -\frac{0.1}{3} & 3 & -0.1 & -0.2 & : & 7.85 \\ & 0.1 & 7 & -0.3 & : & -19.3 \\ & 0.3 & -0.2 & 10 & : & 71.4 \end{array} \qquad \begin{array}{cccccc} -\frac{0.30}{3} & 3 & -0.10 & -0.20 & : & 7.85 \\ & 0 & 7.003 & -0.293 & : & -19.56 \\ & 0.30 & -0.20 & 10.0 & : & 71.4 \end{array}$$

$$\begin{array}{cccccc} & 3 & -0.1 & -0.2 & : & 7.85 \\ \frac{0.19}{7.003} & 0 & 7.003 & -0.293 & : & -19.562 \\ & 0 & -0.19 & 10.02 & : & 70.62 \end{array} \qquad \begin{array}{ccccc} 3 & -0.1 & -0.2 & : & 7.85 \\ 0 & 7.003 & -0.293 & : & -19.562 \\ 0 & 0 & 10.012 & : & 70.08 \end{array}$$

Solution by back substitution

$x_3 = 70.08/10.012 = 7.000,$

$x_2 = -2.5$

$x_1 = 3.000$

Pivoting

Partial Pivoting

$$\begin{array}{rrcl} \downarrow & 0x_1 + 2x_2 + 3x_3 & = & 8 \\ \text{row pivoting} \uparrow & 4x_1 + 6 + 7x_3 & = & \text{-3} \\ & 2x_1 + x_2 + 6x_3 & = & 5 \end{array}$$

Row pivoting only

$$\begin{array}{rcl} 0.0003x_1 + 3.0000x_2 & = & 2.0001 \\ 1.0000x_1 + 1.0000x_2 & = & 1.0000 \end{array} \qquad \begin{array}{rcl} 1.0000x_1 + 1.0000x_2 & = & 1.0000 \\ 0.0003x_1 + 3.0000x_2 & = & 2.0001 \end{array}$$

$$x_1 = -3.33 \text{ and } x_2 = 0.667$$

Row + column pivoting

$$\begin{array}{rcccc} \text{-}\frac{0.0003}{1} & 1.0000 & 1.0000 & : & 1.0000 \\ & 0.0003 & 3.0000 & : & 2.0001 \end{array} \qquad \begin{array}{cccc} 1.0000 & 1.0000 & : & 1.0000 \\ 0 & 2.9997 & : & 1.9998 \end{array}$$

$$x_1 = 0.333 \text{ and } x_2 = 0.667$$

Complete pivoting: Columns as well as rows are searched for the largest element and then switched.
Complete pivoting is rarely used because switching columns changes the order of the x's adds complexity to the algorithm.

Scaling

$$\begin{aligned} 2x_1 + 100.000x_2 &= 100{,}000 \\ x_1 + x_2 &= 2 \end{aligned}$$

$$x_1 = 0.00 \text{ and } x_2 = 1.00$$

Scaling

$$\frac{1}{100{,}000} \quad \begin{aligned} 2x_1 + 100,000x_2 &= 100{,}000 \\ x_1 + x_2 &= 2 \end{aligned} \qquad \begin{aligned} 0.00002x_1 + 1.0000x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}$$

$$x_1 = 1 \text{ and } x_2 = 1$$

Pivoting

$$\begin{aligned} \downarrow \quad 0.00002x_1 + 1.0000x_2 &= 1 \\ \uparrow \quad x_1 + x_2 &= 2 \end{aligned} \qquad \text{-0.00002} \qquad \begin{aligned} x_1 + x_2 &= 2 \\ 0.00002x_1 + 1.0000x_2 &= 1 \end{aligned}$$

$$x_1 = 1 \text{ and } x_2 = 1$$

LU Decomposition

$$
\begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & & \vdots & & \\ l_{N1} & l_{N2} & \dots & 1 \end{bmatrix}
\begin{bmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1N} \\ & u_{22} & u_{23} & \dots & u_{2N} \\ & & u_{33} & \dots & u_{3N} \\ & & \vdots & & \vdots \\ & & & & u_{NN} \end{bmatrix}
=
\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2N} \\ a_{31} & a_{32} & u_{33} & \dots & u_{3N} \\ \vdots & & \vdots & & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \dots & a_{NN} \end{bmatrix}
$$

Crout's Algorithm

1. $u_{11} = a_{11}$
2. Go with the 1st column by computing $l_{i1} = a_{i,1}/u_{11}$
3. Go with the 1st row by computing $u_{1j} = a_{1j}$
4. Go with the jth column by computing $l_{ij} = [a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}]/u_{ij}$
5. Go with the ith row by computing $u_{ij} = [a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}]/l_{ij}$

```
void ludcmp(float **a, int n, int *indx, float *d)
```

Given a matrix a[1..n][1..n], this routine replaces it by the LU decomposition of a row wise permutation of itself. a and n are input. indx[1..n] is an output vector that records the row permutation effected by the partial pivoting; d is output as ± 1 depending on whether the number of row interchanges was even or odd, respectively.

Inverse of a Matrix Using Gauss-Jordan Method

$$\begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

After augmenting the system we apply Gauss Elimination to the lower and upper directions we can find the inverse

$$\begin{array}{c|cccccc} -\frac{0.1}{3} & 3 & -0.1 & -0.2 & 1 & 0 & 0 \\ & 0.1 & 7 & -0.3 & 0 & 1 & 0 \\ & 0.3 & -0.2 & 10 & 0 & 0 & 1 \end{array} \qquad \begin{array}{c|cccccc} -\frac{0.3}{3} & 3 & -0.1 & -0.2 & 1 & 0 & 0 \\ & 0 & 7.003 & -0.293 & -0.033 & 1 & 0 \\ & 0.3 & -0.2 & 10 & 0 & 0 & 1 \end{array}$$

$$\begin{array}{c|cccccc} & 3 & -0.1 & -0.2 & 1 & 0 & 0 \\ -\frac{0.190}{7.00} & 0 & 7.00 & -0.293 & -0.033 & 1 & 0 \\ & 0 & -0.190 & 10.02 & -0.1 & 0 & 1 \end{array} \qquad \begin{array}{c|ccccc} & 3 & -0.1 & -0.2 & 1 & 0 \\ \frac{0.1}{7.00} & 0 & 7.00 & -0.29 & -0.033 & 1 \\ & 0 & 0 & 10.012 & -0.101 & 0.027 \end{array}$$

$$\begin{array}{c|cccccc} & 3 & -0.1 & -0.2 & 1 & 0 & 0 \\ \frac{0.1}{7.00} & 0 & 7.00 & -0.29 & -0.033 & 1 & 0 \\ & 0 & 0 & 10.012 & -0.101 & 0.027 & 1 \end{array} \qquad \begin{array}{c|ccccc} & 3 & 0 & -0.204 & 1 & 0.014 \\ & 0 & 7.00 & -0.293 & -0.033 & 1 \\ \frac{0.204}{10.012} & 0 & 0 & 10.012 & -0.101 & 0.027 \end{array}$$

$$\begin{array}{c|cccccc} & 3 & 0 & 0 & 0.979 & 0.02 & 0.202 \\ & 0 & 7.00 & -0.29 & -0.033 & 1 & 0 \\ \frac{0.29}{10.01} & 0 & 0 & 10.01 & -0.101 & 0.027 & 1 \end{array} \qquad \begin{array}{cccccc} 3 & 0 & 0 & 0.98 & 0.02 & 0.20 \\ 0 & 7.00 & 0 & -0.063 & 1.008 & 0.29 \\ 0 & 0 & 1.012 & -0.101 & 0.027 & 1 \end{array}$$

$$\begin{array}{c|cccccc} 1/3 & 3 & 0 & 0 & 0.98 & 0.02 & 0.202 \\ 1/7 & 0 & 7.00 & 0 & -0.063 & 1.008 & 0.29 \\ 1/10.012 & 0 & 0 & 1.012 & -0.101 & 0.027 & 1 \end{array} \qquad \begin{array}{cccccc} 1 & 0 & 0 & 0.333 & 0.005 & 0.007 \\ 0 & 1 & 0 & -0.005 & 0.143 & 0.004 \\ 0 & 0 & 1 & -0.010 & 0.003 & 0.010 \end{array}$$

```
void gaussj(float **a, int n, float **b, int m)
```

Linear equation solution by Gauss-Jordan elimination. a[1..n][1..n]is the input matrix. b[1..n][1..m] is input containing the m right-hand side vectors. On output, a is replaced by its matrix inverse, and b is replaced by the corresponding set of solution vectors.

Cholesky Algorithm Applied to positive definite matrices

Positive Definite Matrix

A matrix $\mathbf{X}$ is positive definite if $\mathbf{v}^T\mathbf{X}\mathbf{v} > 0$ for all vectors $\mathbf{v}$.
An example of a positive definite matrix is $\mathbf{A}^T\mathbf{A}$ assuming that $\mathbf{A}$ is any matrix.

$$L = U^T$$

1. Set $u_{11} = \sqrt{a_{11}}$
2. For the 1st row compute $u_{1j} = a_{1j}/u_{11}$
3. Compute $u_{ii} = \{a_{ii} - \sum_{k=1}^{i-1}(u_{ki})^2\}^{1/2}$
4. Compute $u_{ij} = 1/u_{ii}\{a_{ij} - \sum_{k=1}^{i-1}(u_{ki})(u_{kj})\}$

```
void choldc(float **a, int n, float p[])
```

Constructs the Cholesky decomposition, A = L · LT of a positive-definite symmetric matrix a[1..n][1..n]. On input, only the upper triangle of A need be given; it is not modified. The Cholesky factor L is returned in the lower triangle of a, except for its diagonal elements which are returned in p[1..n].

Iterative Methods

Jacobi Method
$\mathbf{A} = \mathbf{V} + \mathbf{D} + \mathbf{W}$
$\mathbf{A}$: Upper Diagonal elements, $\mathbf{D}$: Diagonal elements , $\mathbf{W}$: Lower Diagonal elements
$\mathbf{Ax} = \mathbf{b} \rightarrow (\mathbf{V} + \mathbf{D} + \mathbf{W})\mathbf{x} = \mathbf{b} \rightarrow \mathbf{x} = \mathbf{D}^{-1}[\mathbf{b} - (\mathbf{V} + \mathbf{W})\mathbf{x}]$
$x_i^k = x_i^{k-1} + (1/d_{ii})[b_i - \sum_{j=1}^{N} a_{i,j} x_j^{k-1}]$

Gauss-Seidel Method
$\mathbf{Ax} = \mathbf{b}$
$x_1^k = (b_1 - a_{12}x_2^{k-1} - a_{13}x_3^{k-1} - \ldots - a_{1N}x_N^{k-1})/a_{11}$
$x_2^k = (b_2 - a_{21}x_1^{k} - a_{23}x_3^{k-1} - \ldots - a_{2N}x_N^{k-1})/a_{22}$
$x_i^k = (b_i - a_{i1}x_1^{k} - a_{i2}x_2^{k} - \ldots - a_{ik-1}x_{k-1}^{k} - a_{ik+1}x_{k+1}^{k-1} - \ldots - a_{iN}x_N^{k-1})/a_{kk}$

For convergence $\mathbf{A}$ must be diagonally dominant i.e.
$|a_{kk}| > |a_{k1}| + \ldots + |a_{kk-1}| + |a_{kk+1}| + \ldots + |a_{kN}|$

for $k = 1, 2, \ldots, N$

Singular Value Decomposition

$\mathbf{A} = \mathbf{UWT}^T$

$$\begin{bmatrix} \vec{\mathbf{u}}_1 & \vec{\mathbf{u}}_2 \ldots \vec{\mathbf{u}}_n \\ \downarrow & \downarrow \quad \downarrow \end{bmatrix}_{mxn} \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_n \end{bmatrix}_{nxn} \begin{bmatrix} \vec{\mathbf{v}}_1 \rightarrow \\ \vec{\mathbf{v}}_2 \rightarrow \\ \vdots \\ \vec{\mathbf{v}}_n \rightarrow \end{bmatrix}_{nxn}$$

$\vec{\mathbf{u}}_i$: left eigenvectors of $\mathbf{A}$ (which span the Range Space of $\mathbf{A}$),
$\vec{\mathbf{v}}_i$: right eigenvectors of $\mathbf{A}$ (which span the Null Space of $\mathbf{A}$),
w_{ii} : singular values of $\mathbf{A}$.

Dimension of the Range is equal to the Rank of $\mathbf{A}$.
$\vec{\mathbf{u}}_i$ for which $w_{ii} \neq 0$, span the Range Space of $\mathbf{A}$.
$\vec{\mathbf{v}}_i$ for which $w_{ii} = 0$, span the Null Space of $\mathbf{A}$.

$\vec{\mathbf{u}}_i \perp \vec{\mathbf{u}}_j$, $\vec{\mathbf{v}}_i \perp \vec{\mathbf{v}}_j$, $\vec{\mathbf{u}}_i \perp \vec{\mathbf{v}}_j$
$\mathbf{A}$ can be expressed as $\mathbf{A} = \sum_{i=1}^{n} w_{ii}(\vec{\mathbf{u}}_i \vec{\mathbf{v}}_i^T)$
$\mathbf{A}^{-1}$ is called the Pseudo-inverse and determined as: $\mathbf{A}^{-1} = \sum_{i=1}^{n} 1/w_{ii}(\vec{\mathbf{v}}_i \vec{\mathbf{u}}_i^T)$

```
void svdcmp(float **a, int m, int n, float w[], float **v)
```

Computes its singular value decomposition, A =U·W·V T of a matrix a[1..m][1..n]. The matrix U replaces a on output. The diagonal matrix of singular values W is output as a vector w[1..n]. The matrix V (not the transpose V T) is output as v[1..n][1..n].

```
void svbksb(float **u, float w[], float **v, int m, int n, float b[],
float x[])
```

Solves A·X = B for a vector X, where A is specified by the arrays u[1..m][1..n], w[1..n],v[1..n][1..n] as returned by svdcmp. m and n are the dimensions of a, and will be equal for square matrices. b[1..m] is the input right-hand side. x[1..n] is the output solution vector. No input quantities are destroyed, so the routine may be called sequentially with different b's.