LU Decomposition

ETB 310: Data Analysis and Tools
Section 5.6 (pdf), Section 6.6 (online study guide)

Tim Bower

LU Decomposition

- An elimination based matrix factoring for square matrices
- No augmented matrix during elimination—only A
- $\mathsf{A} = \mathsf{L}\,\mathsf{U}$ and $\mathsf{P}\,\mathsf{A} = \mathsf{L}\,\mathsf{U}$
  L is lower-triangular. U is upper-triangular.
- Two calls to triangular solver—forward and back substitution
  $\mathsf{L}\,\mathsf{U}\,x = \mathsf{P}\,b \qquad \mathsf{L}\,y = \mathsf{P}\,b \qquad \mathsf{U}\,x = y$
- Faster than Gaussian elimination. Elimination algorithm is faster. Can reuse L and U with more than one b.
- Better numerical accuracy than Gaussian elimination.

Elimination

- Use elimination to change A to upper triangular as before.
- Capture row exchanges (partial pivoting) in permutation matrix, P.
  Capture row operations in elementary matrices, E_i.
  Product of elementary matrices is the elimination matrix E.
  $\mathsf{A} \mapsto \mathsf{E}\,\mathsf{P}\,\mathsf{A} = \mathsf{U}$
- Possible to skip E and build L during elimination.
  $\mathsf{P}\,\mathsf{A} = \mathsf{E}^{-1}\mathsf{U} \mapsto \mathsf{P}\,\mathsf{A} = \mathsf{L}\,\mathsf{U}$
- Solving for x:

$$\begin{aligned} \mathsf{A}x = b \quad &\mapsto \quad \mathsf{P}^T\mathsf{L}\,\mathsf{U}\,x = b \\ x = \mathsf{A}^{-1}b \quad &\mapsto \quad x = \mathsf{U}^{-1}\mathsf{L}^{-1}\mathsf{P}\,b \end{aligned}$$

```
[L, U, P] = lu(A);
x = U\(L\(P*b));
```

Example

$$\begin{cases} 2x - 3y \phantom{{}+z} = 3 \\ 4x - 5y + z = 9 \\ 2x - y - 3z = -1 \end{cases} \qquad \mathbf{U} = \begin{bmatrix} 2 & -3 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & -5 \end{bmatrix}$$

1. Add -1 of *row 1* to *row 3*, called $\mathbf{E}_1$.
2. Add -2 of *row 1* to *row 2*, called $\mathbf{E}_2$.
3. Add -2 of *row 2* to *row 3*, called $\mathbf{E}_3$.

```
>> E1
E1 =
     1     0     0
     0     1     0
    -1     0     1
```

```
>> E2
E2 =
     1     0     0
    -2     1     0
     0     0     1
```

```
>> E3
E3 =
     1     0     0
     0     1     0
     0    -2     1
```

Example continued

The order of matrices must be such that the first operation applied is next to $\mathbf{A}$ in the equation $\mathbf{U} = \mathbf{E}\,\mathbf{A}$.

```
>> E = E3*E2*E1
E =
     1     0     0
    -2     1     0
    -1    -2     1
```

```
>> U = E*A
U =
     2    -3     0
     0     1     1
     0     0    -5
```

```
>> L = inv(E1)*inv(E2)*inv(E3) % L = inv(E)
L =
     1     0     0
     2     1     0
     1     2     1
```

Notice that non-diagonal values in $\mathbf{L}$ are negative of values in $\mathbf{E}$. Skip $\mathbf{E}$ and build $\mathbf{L}$ during elimination.

Example with Row Exchanges

$$\mathbf{A} = \begin{bmatrix} 0 & 12 & -3 \\ 8 & -4 & -6 \\ -4 & -2 & 12 \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Add 1/2 of row 1 to row 3

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} 8 & -4 & -6 \\ 0 & 12 & -3 \\ 0 & -4 & 9 \end{bmatrix}$$

Add 1/3 of row 2 to row 3

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/2 & -1/3 & 1 \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} 8 & -4 & -6 \\ 0 & 12 & -3 \\ 0 & 0 & 8 \end{bmatrix}$$

Turing's *kij* Algorithm

See file `turingLU.m` for the full function

```
function [L, U, P] = turingLU(A) % help skipped
[m, n] = size(A); % skipped check for square
P = eye(n);
for k = 1:(n - 1) % Skipped row exchange code
    for i = k + 1:n
        A(i, k) = A(i, k)/A(k, k);
        for j = (k + 1):n
            A(i, j) = A(i, j) - A(i, k) * A(k, j);
        end
    end
end
L = tril(A, -1) + eye(n); % extract lower
U = triu(A);              % and upper triangular
```

See file turingLU.m for the full function

```
P = eye(n);
for k = 1:(n - 1)
    [A(k:n,:), idx] = sortrows(A(k:n,:), k, ...
        'descend', ...
        'ComparisonMethod','abs');
    I = P(k:n,:);
    P(k:n,:) = I(idx,:); % Permutation matrix

    % Next comes i,j elimination portion
    % of the k,i,j loop
```

Comments on Turing's *kij* Algorithm

- Down the diagonal: `for k = 1:(n - 1)`
- Down the column below the diagonal: `for i = k + 1:n`
- Set value for `L` matrix: `A(i, k) = A(i, k)/A(k, k);`
- Across each row: `for j = (k + 1):n`
- Row operations:
  `A(i, j) = A(i, j) - A(i, k) * A(k, j);`
- Three regions of each row
  1. `L` matrix value at `A(i, k)`
  2. For future `L` values from `A(i, k+1)` to `A(i, i-1)`
  3. For future **U** values from the diagonal to the end of the row (`A(i, i)` to `A(i, n)`)

Determinant Shortcut

The determinant of $\mathbf{A}$ is the product of its factors' determinants.

$$|\mathbf{A}| = |\mathbf{P}^T|\,|\mathbf{L}|\,|\mathbf{U}|.$$

- $|\mathbf{P}^T| = |\mathbf{P}|$ is either 1 or -1.
- $|\mathbf{L}| = 1$
- $|\mathbf{U}|$ is the product of its diagonal.

```
>> [L, U, P] = lu(A);
>> determinant = det(P)*prod(diag(U));
```