Linear Programming

(5531)

Lecture 04 Klee-Minty Problem / Matrix Notations / Implementation of Simplex Methods

Prof. Sheng-I Chen

http://www.iem-omglab.nctu.edu.tw/OMG_LAB/

October 15, 2020

Agenda

- The worst case of the simplex algorithm (Chapter 4)
- Simplex method in matrix notation (Chapter 6)
- Implementing the simplex method (In part of chapter 8)

The Worst Case of the Simplex Algorithm

Efficiency of Simplex Method

- How fast the simplex method can solve a problem of a given size?
- Two ways to measure an algorithm's performance:
  - Worst-case analysis: the effort to solve the *hardest* problem of a given problem size
  - Average-case analysis: the average amount of effort over all problems of a given size
- The worst-case analysis is more tractable than average-case analysis, however, it is less relevant to solve real-world problems

Efficiency of Simplex Method

- CPU Speeds:
  - Intel Core i750 at 7G FLOPs (Floating-point Operations Per Second)
  - AMD Phenom II X4 at 7.5G FLOPs
  - Celeron M 540 at 0.9 G FLOPs
  - Pentium4 at 0.74G FLOP
- Which are “good” algorithms?

Time Complexity Function	Problem size *n*					
	10	20	40	60	80	100
n	0.00000001 second	0.00000002 second	0.00000004 second	0.00000006 second	0.00000008 second	0.0000001 second
n^2	0.0000001 second	0.0000004 second	0.000002 second	0.000004 second	0.0000064 second	0.00001 second
n^7	0.01 second	1.3 seconds	2.7 minutes	46.6 minutes	5.8 hours	27.7 hours
2^*n*	0.000001 second	0.001 second	18.3 minutes	36 years	3.8X10^5 Centuries	4.0X10^11 Centuries
3^*n*	0.00006 second	3.5 seconds	386 years	1.3X10^10 Centuries	4.7X10^19	1.6X10^29 Centuries
n!	0.004 second	77 years	2.6X10^29 Centuries	2.6X10^63 Centuries	2.3X10^100 Centuries	3.0X10^139 Centuries
n^*n*	10 seconds	3.3X10^7 Centuries	3.8X10^45 Centuries	1.6X10^88 Centuries	5.6X10^133 Centuries	3.2X10^181 Centuries

Efficiency of Simplex Method

- If we choose the entering variable with the largest reduced cost, how many iterations are needed in the worst case?

$$\binom{n+m}{m}$$

- This is also the number of basic feasible solutions
- The simplex algorithm needs to visit 2^n bfs's before the optimal solution is obtained
- The worst case happened when we apply the largest-reduced-cost rule for solving the **Klee-Minty problems**
- In such case, the feasible region is a *hypercube* and the simplex algorithm *visits every vertex*

The Klee-Minty Problem

- For example:

$$
\begin{aligned}
\text{maxmize} \quad & \sum_{\mathrm{j=1}}^{\mathrm{n}} 10^{n-j} x_j \\
\text{subject to} \quad & 2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \le 100^{i-1} \qquad i = 1,2,\ldots,n \\
& x_j \ge 0 \qquad j = 1,2,\ldots,n
\end{aligned}
$$

If n = 3:

$$
\begin{aligned}
\text{maximize} \quad & 100x_1 + 10x_2 + x_3 \\
\text{subject to} \quad & x_1 \le 1 \\
& 20x_1 + x_2 \le 100 \\
& 200x_1 + 20x_2 + x_3 \le 10000
\end{aligned}
$$

The Klee-Minty Problem

- Let's replace the specific right-hand-side, 100^{i-1} , with more generic value, b_i, with the following property: $1 = b_1 << b_2 << \ldots << b_n$
- *Next, change rhs to:* $\sum_{j=1}^{i-1} 10^{i-j} b_j + b_i$
- *Finally, we add a constant to the objective function, the problem become:*

$$
\begin{array}{lll}
\text{maxmize} & \sum_{j=1}^{n} 10^{n-j} x_j - \frac{1}{2} \sum_{j=1}^{n} 10^{n-j} b_j & \\
\text{subject to} & 2 \sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \le \sum_{i=1}^{i-1} 10^{i-j} b_j + b_i & i = 1,2,\ldots,n \\
& x_j \ge 0 & j = 1,2,\ldots,n
\end{array}
$$

Example

The Klee-Minty problem with $n = 3$ takes 2^3 - 1 iterations.

- Initial dictionary:

$$
\begin{array}{l}
\zeta = -\tfrac{100}{2}b_1 - \tfrac{10}{2}b_2 - \tfrac{1}{2}b_3 + 100x_1 + 10x_2 + x_3 \\
\hline
\omega_1 = b_1 - x_1 \\
\omega_2 = 10b_1 + b_2 - 20x_1 - x_2 \\
\omega_3 = 100b_1 + 10b_2 + b_3 - 200x_1 - 20x_2 - x_3
\end{array}
$$

- 1st dictionary:

$$
\begin{array}{l}
\zeta = \tfrac{100}{2}b_1 - \tfrac{10}{2}b_2 - \tfrac{1}{2}b_3 - 100\omega_1 + 10x_2 + x_3 \\
\hline
x_1 = b_1 - \omega_1 \\
\omega_2 = -10b_1 + b_2 + 20\omega_1 - x_2 \\
\omega_3 = -100b_1 + 10b_2 + b_3 + 200\omega_1 - 20x_2 - x_3
\end{array}
$$

- 2nd dictionary:

$$
\begin{array}{l}
\zeta = -\tfrac{100}{2}b_1 + \tfrac{10}{2}b_2 - \tfrac{1}{2}b_3 + 100\omega_1 - 10\omega_2 + x_3 \\
\hline
x_1 = b_1 - \omega_1 \\
x_2 = -10b_1 + b_2 + 20\omega_1 - \omega_2 \\
\omega_3 = 100b_1 - 10b_2 + b_3 - 200\omega_1 + 20\omega_2 - x_3
\end{array}
$$

- 3rd dictionary:

$$
\begin{array}{l}
\zeta = \tfrac{100}{2}b_1 + \tfrac{10}{2}b_2 - \tfrac{1}{2}b_3 - 100x_1 - 10\omega_2 + x_3 \\
\hline
\omega_1 = b_1 - x_1 \\
x_2 = 10b_1 + b_2 - 20x_1 - \omega_2 \\
\omega_3 = -100b_1 - 10b_2 + b_3 + 200x_1 + 20\omega_2 - x_3
\end{array}
$$

- 4th dictionary:

$$
\begin{array}{l}
\zeta = -\tfrac{100}{2}b_1 - \tfrac{10}{2}b_2 + \tfrac{1}{2}b_3 + 100x_1 + 10\omega_2 - \omega_3 \\
\hline
\omega_1 = b_1 - x_1 \\
x_2 = 10b_1 + b_2 - 20x_1 - \omega_2 \\
x_3 = -100b_1 - 10b_2 + b_3 + 200x_1 + 20\omega_2 - \omega_3
\end{array}
$$

- 5th dictionary:

$$
\begin{array}{l}
\zeta = \tfrac{100}{2}b_1 - \tfrac{10}{2}b_2 + \tfrac{1}{2}b_3 - 100\omega_1 + 10\omega_2 - \omega_3 \\
\hline
x_1 = b_1 - \omega_1 \\
x_2 = -10b_1 + b_2 + 20\omega_1 - \omega_2 \\
x_3 = 100b_1 - 10b_2 + b_3 - 200\omega_1 + 20\omega_2 - \omega_3
\end{array}
$$

- 6th dictionary:

$$
\begin{array}{l}
\zeta = -\tfrac{100}{2}b_1 + \tfrac{10}{2}b_2 + \tfrac{1}{2}b_3 + 100\omega_1 - 10x_2 - \omega_3 \\
\hline
x_1 = b_1 - \omega_1 \\
\omega_2 = -10b_1 + b_2 + 20\omega_1 - x_2 \\
x_3 = -100b_1 + 10b_2 + b_3 + 200\omega_1 - 20x_2 - \omega_3
\end{array}
$$

- 7th dictionary:

$$
\begin{array}{l}
\zeta = \tfrac{100}{2}b_1 + \tfrac{10}{2}b_2 + \tfrac{1}{2}b_3 - 100x_1 - 10x_2 - \omega_3 \\
\hline
\omega_1 = b_1 - x_1 \\
\omega_2 = 10b_1 + b_2 - 20x_1 - x_2 \\
x_3 = 100b_1 + 10b_2 + b_3 - 200x_1 - 20x_2 - \omega_3
\end{array}
$$

Insights from this Example

- Every pivot is the swap of an x_j with the corresponding ω_j
- All dictionaries have similar equations with the exception that the ω_i's and the x_j's have become intertwined and various signs have changed
- In the initial dictionary, if we select x_3 (*with the smallest objective coefficient*) entering the basis instead of selecting x_1, then the optimal solution can be founded in one iteration

Existence of the Best Pivoting Rule?

- Most pivoting rules have been shown to have exponential iterations on solving specific problems
- No one has proven that there is a pivoting rule faster than others
- Some interior-point algorithms guarantee to solve LPs in polynomial time. This gives us a hope to solve the LP effectively

Simplex Method in Matrix Notations

Simplex Method in Matrix Notation

- In the foregoing lectures, LP is expressed as:

$$\begin{aligned} &\text{maxmize} \quad \sum_{j=1}^{n} c_j x_j \\ &\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad i = 1,2,\ldots,m \\ &\qquad\qquad\quad x_j \ge 0 \qquad j = 1,2,\ldots,n \end{aligned}$$

- After adding slack variables, the LP becomes:

$$\omega_i = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1,2,\ldots,m$$

- Now, we write the LP in matrix form:

$$\begin{aligned} &\max \quad c^T x \\ &\text{s.t.} \quad Ax = b \\ &\qquad\quad x \ge 0 \end{aligned}$$

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

- Let *A* be a matrix with m rows and $m+n$ columns, where $A = [B \quad N]$
- What are the dimensions of B and N?
- Let x_B and x_N be the sets of basic and nonbasic variables, respectively
- Then,

$$\sum_{j=1}^{n+m} a_{ij} x_j = \sum_{j \in B} a_{ij} x_j + \sum_{j \in N} a_{ij} x_j$$

$$Ax = Bx_B + Nx_N$$

- The vector of objective function coefficients is $c = \begin{bmatrix} c_B \\ c_N \end{bmatrix}$
- , and the objective function is:

$$c^T x = \begin{bmatrix} c_B^T & c_N^T \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix} = c_B^T x_B + c_N^T x_N$$

- Constraints are:

$$\begin{bmatrix} B & N \end{bmatrix} \begin{bmatrix} x_B \\ x_N \end{bmatrix} = Bx_B + Nx_N = b$$

- Then, the vector of basic variables, x_B ,can be rewritten as:

$$x_B = B^{-1}b - B^{-1}Nx_N$$

- Replace x_B in the objective function, $c^T x = c_B^T x_B + c_N^T x_N$:

$$\begin{aligned} \zeta &= c_B^T x_B + c_N^T x_N \\ &= c_B^T (B^{-1}b - B^{-1}Nx_N) + c_N^T x_N \\ &= c_B^T B^{-1}b - ((B^{-1}N)^T c_B - c_N)^T x_N \end{aligned}$$

- The dictionary is:

$$\zeta = c_B^T B^{-1} b - ((B^{-1}N)^T c_B - c_N)^T x_N$$
$$x_B = B^{-1}b - B^{-1}Nx_N$$

- The basic feasible solution associated with the current dictionary is:

$$x_N^* = 0, x_B^* = B^{-1}b$$

- What does the basis look like in the next iteration?

- Select variable with index j entering the basis:

$$x_N = (0,\ldots,0,\underset{\substack{\uparrow \\ j\text{-th position}}}{t},0,\ldots,0)^T = te_j$$

- Then, the values of basic variable become:

$$x_B = x_B^* - B^{-1}Nte_j$$

- Let $\Delta x_B = B^{-1}Ne_j$
- We need the largest t such that $x_B \geq 0$ (*ratio test*)
- Thus, $x_B^* \geq t\Delta x_B$

$$t = (\max_{i \in B} \frac{\Delta x_i}{x_i^*})^{-1}$$

- Select the i–th variable leaving the basis
- Update values of the new basic variables:

$$x_j^* = t$$

$$x_B = x_B^* - t\Delta x_B$$

$$B \leftarrow B \setminus \{i\} \cup \{j\}$$

Implementing Simplex Method

Implementation Issues (Chapter 8)

- The most time-consuming step in simplex method is:

$$\Delta x_B = B^{-1} N e_j$$

- , and the difficulty is to compute the inverse of the basis matrix, B^{-1}
- In fact, we calculate Δx_B by solving the system of equations:

$$B \Delta x_B = a_j$$

Solving the System of Equations

- Given a system of equations:

$$Bx = b$$

- , where B is invertible $m \times m$ matrix and b is a m-vector
- How to solve this system of equations?
- Our first thought is to use the **Gaussian Elimination**

Gaussian Elimination

- Example: In order to get zero entries in the first column below the diagonal. We subtract 3/2 times the first row from the second row and 1/2 times the first row from the third row.

$$B = \begin{array}{c} \left[\begin{array}{ccccc} 2 & & 4 & & -2 \\ 3 & 1 & & 1 & \\ -1 & & -1 & & -2 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{array}\right] \end{array} \begin{array}{l} \\ -\frac{3}{2} \\ \frac{1}{2} \\ \\ \\ \end{array}$$

- The result is:

$$\left[\begin{array}{ccccc} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{array}\right]$$

- Next, we subtract the second row from the fourth row, and then we have:

$$\begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{bmatrix} \quad 1$$

- Subtract 6 times the third row from the fourth row:

$$\begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & -6 & 1 & -3 \\ & & 1 & & 4 \end{bmatrix} \quad 6$$

- Finally, we obtain:

$$\begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & -6 & 1 & -21 \\ & & 1 & & 7 \end{bmatrix}$$

LU-factorization

- We keep tracking of the elimination matrixes (L) and remaining part of the original matrix (U):

$$B = \begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & -6 & 1 & 3 \\ -1 & & 1 & & -3 \\ & -1 & -6 & 1 & -21 \\ & & 1 & & 7 \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}$$

- $B = LU$ (we said an LU-factorization of B)

- Surprise, the *B* matrix is equal to the product of lower triangular matrix, the inverse of diagonal matrix and upper-triangular matrix:

$$B=\begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix}\begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1}\begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}$$

- , where the product of the lower triangular matrix and the inverse of diagonal matrix is denoted by L:

$$L=\begin{bmatrix} 2 & & & & \\ 3 & 1 & & & \\ -1 & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 7 \end{bmatrix}\begin{bmatrix} 2 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 7 \end{bmatrix}^{-1}=\begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix}$$

- , and the upper triangular matrix is denoted by U:

$$U=\begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}$$

Solving the System of Equations: LU-factorization

- Let's back to the simplex method, where we need to calculate Δx_B by solving the system of equations:

$$B\Delta x_B = a_j$$

- First we substitute LU for B, thus, the system become:

$$LU\Delta x_B = a_j$$

- Let $y = U\Delta x_B$, then:

$$Ly = a_j$$

- L and a_j are known, and we can solve for y easily (*Step 1*)
- We have $y = U\Delta x_B$, where y is obtained in Step 1 and U is know. So, we can solve for Δx_B (*Step 2*)

- Example:

$$B=\begin{bmatrix} 2 & & 4 & & -2 \\ 3 & 1 & & 1 & \\ -1 & & -1 & & -2 \\ & -1 & & & -6 \\ & & 1 & & 4 \end{bmatrix} \qquad a_j=\begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

$B\Delta x_B = a_j$, we want to solve for Δx_B

- In the previous slides, we've done LU-factorization for B:

$$L=\begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix} \qquad U=\begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix}$$

- Step 1. Solve *y* by using $Ly = a_j$:

$$Ly = \begin{bmatrix} 1 & & & & \\ \frac{3}{2} & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & -1 & -6 & 1 & \\ & & 1 & & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -2 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

  - $y_1$an be obtained immediately, y_1 = 7. Then, y_2 = -2 –(3/2) y_1 = -25/2…
  - Final, we find values for every entries in y:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

- Step 2. Solve Δx_B by using $U\Delta x_B = y$:

$$\begin{bmatrix} 2 & & 4 & & -2 \\ & 1 & -6 & 1 & 3 \\ & & 1 & & -3 \\ & & & 1 & -21 \\ & & & & 7 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} 7 \\ -\frac{25}{2} \\ \frac{7}{2} \\ \frac{23}{2} \\ -\frac{7}{2} \end{bmatrix}$$

  - Start from Δx_5 , and we find the value is -1/2. Then, solve for Δx_4 , and etc.
  - Finally, we obtain:

$$\Delta x_B = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \\ \Delta x_5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix}$$

LU-factorization for the Sparse Matrix

- In the case of zero diagonal element:
  - The nonzero elements under it are not able to be eliminated (Cons)
  - Additional computational efficiency can be obtained by keeping sparsity in L and U (Pros)
- The **minimum degree ordering heuristic** is to find to best permutation by the following procedure:
  - Find the row having the fewest nonzeros (sparse) in its uneliminated part. Swap this row with pivot row
  - Within the row we just swap, scan the uneliminated nonzeros and select the one whose column has the fewest nonzeros in its uneliminated part. Swap this column with pivot row

Minimum Degree Ordering Heuristic

- Example:

$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 2 & & 4 & & -2 \\
2 & 3 & 1 & & 1 & \\
B=3 & -1 & & -1 & & -2 \\
4 & & -1 & & & -6 \\
5 & & & 1 & & 4
\end{array}
$$

- As row 4 has the fewest nonzeros, we swap that row with the pivot row (row 1). Next, we scan nonzeros elements at row 4 (-1 and -6), and check their columns (column 2 and 5). As column 2 has less nonzeros than column 5, we swap columns 1 and 2:

$$
\begin{array}{c|ccccc}
 & 2 & 1 & 3 & 4 & 5 \\
\hline
4 & -1 & & & & -6 \\
2 & 1 & 3 & & 1 & \\
B=3 & & -1 & -1 & & -2 \\
1 & & 2 & 4 & & -2 \\
5 & & & 1 & & 4
\end{array}
$$

- Eliminate the nonzeros under the first diagonal element:

$$
\begin{array}{c}
 \\ 4 \\ 2 \\ 3 \\ 1 \\ 5
\end{array}
\begin{array}{c}
\begin{array}{ccccc} 2 & 1 & 3 & 4 & 5 \end{array} \\
\left[\begin{array}{c|cccc}
-1 & & & & -6 \\ \hline
1 & 3 & & 1 & -6 \\
 & -1 & -1 & & -2 \\
 & 2 & 4 & & -2 \\
 & & 1 & & 4
\end{array}\right]
\end{array}
$$

- Row 5 has fewest nonzeros, so we swap rows 2 and 5. There are two nonzero elements on row 5, 1 and 4 are at column 3 and 4, respectively. As column 3 has less nonzeros than column 5, we swap columns 1 and 3:

$$
\begin{array}{c}
 \\ 4 \\ 5 \\ 3 \\ 1 \\ 2
\end{array}
\begin{array}{c}
\begin{array}{ccccc} 2 & 3 & 1 & 4 & 5 \end{array} \\
\left[\begin{array}{c|cccc}
-1 & & & & -6 \\ \hline
 & 1 & & & 4 \\
 & -1 & -1 & & -2 \\
 & 4 & 2 & & -2 \\
1 & & 3 & 1 & -6
\end{array}\right]
\end{array}
$$

- Eliminate the nonzeros under the second diagonal element:

$$
\begin{array}{c c}
 & \begin{array}{ccccc} \;\;2\;\; & \;\;3\;\; & \;\;1\;\; & \;\;4\;\; & \;\;5\;\; \end{array} \\
\begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} &
\begin{bmatrix}
-1 & & & & -6 \\
 & 1 & & & 4 \\
 & -1 & -1 & & 2 \\
 & 4 & 2 & & -18 \\
1 & & 3 & 1 & -6
\end{bmatrix}
\end{array}
$$

- Row 3 is a minimum-degree row, and among nonzero elements of that row, the -1 is in a minimum-degree column. No permutations are needed

- Eliminate the nonzeros under the third diagonal element:

$$\begin{array}{cc} & \begin{array}{ccccc} 2 & 3 & 1 & 4 & 5 \end{array} \\ \begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} & \begin{bmatrix} -1 & & & & -6 \\ & 1 & & & 4 \\ & -1 & -1 & & 2 \\ & 4 & 2 & & -14 \\ 1 & & 3 & 1 & \end{bmatrix} \end{array}$$

- Both remaining rows have the same degree, hence, no need to swap rows. Swap columns 5 and 4:

$$\begin{array}{cc} & \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} & \begin{bmatrix} -1 & & & -6 & \\ & 1 & & 4 & \\ & -1 & -1 & 2 & \\ & 4 & 2 & -14 & \\ 1 & & 3 & & 1 \end{bmatrix} \end{array}$$

LU-factorization for the Sparse Matrix

- We have done the Minimum Degree Ordering Heuristic. Next we can extract the matrices L and U:

$$L = \begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -1 & 1 & & \\ & 4 & -2 & 1 & \\ -1 & & -3 & & 1 \end{bmatrix} \qquad U = \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} \\ \begin{bmatrix} -1 & & & -6 & \\ & 1 & & 4 & \\ & & -1 & 2 & \\ & & & -14 & \\ & & & & 1 \end{bmatrix} \end{array}$$

- Do matrices L and U look more sparse?
  - There are 5 off-diagonal nonzeros in L matrix and 3 off-diagonal nonzeros in U matrix. In contrast, the regular LU-factorization had a total of 12 off-diagonal nonzeros. The minimum-degree ordering heuristic reduced the number of nonzeros in 33%
- Can we solve the system of equations faster by using the new L and U?

Example

- To illustrate, let us solve the same system that we considered before:

$$B\Delta x_B = a_j$$

- Step 1. Solve for y in the system $Ly = a_j$:

$$\begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array}\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -1 & 1 & & \\ & 4 & -2 & 1 & \\ -1 & & -3 & & 1 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{array}{c} 4 \\ 5 \\ 3 \\ 1 \\ 2 \end{array}\begin{bmatrix} 3 \\ 0 \\ 0 \\ 7 \\ -2 \end{bmatrix}$$

- The result is:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \\ 7 \\ 1 \end{bmatrix}$$

- Step 2. Solve for Δx_B in the system $y = U\Delta x_B$:

$$
\begin{array}{c c}
\begin{array}{ccccc} 2 & 3 & 1 & 5 & 4 \end{array} & \\
\begin{bmatrix} -1 & & & -6 & \\ & 1 & & 4 & \\ & & -1 & 2 & \\ & & & -14 & \\ & & & & 1 \end{bmatrix} & \begin{array}{c} 2 \\ 3 \\ 1 \\ 5 \\ 4 \end{array}
\end{array}
\begin{bmatrix} \Delta x_2 \\ \Delta x_3 \\ \Delta x_1 \\ \Delta x_5 \\ \Delta x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 0 \\ 7 \\ 1 \end{bmatrix}
$$

- , and we got:

$$
\begin{array}{c} 2 \\ 3 \\ 1 \\ 5 \\ 4 \end{array}
\begin{bmatrix} \Delta x_2 \\ \Delta x_3 \\ \Delta x_1 \\ \Delta x_5 \\ \Delta x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ -1 \\ -\frac{1}{2} \\ 1 \end{bmatrix}
$$

- Did we perform less operations on using substitution to solve the systems of equations?

Reusing Factorization

Reusing the Factorization

- Still, LU-factorization is a computational bottleneck in the simplex algorithm
- If the matrix dimension is $m \times m$, the LU-factorization routine requires about m^3 operations, plus, substitutions for solving system equations and each requires m^2 operations
- Reusing a factorization is a approach to avoid the LU-factorization procedure to improve efficiency

- The current iteration:

$$B\Delta x_B = a_j$$

- Pick x_j entering the basis and x_i leaving the basis
- The next iteration basis, $\tilde{B}$, can be presented as:

$$\tilde{B} = B + (a_j - a_i)e^T{}_i$$

, where the column vector a_j associated with x_j, and a_i associated with x_i

- Matrix *B* is invertible, thus, we can rewrite as:

$$\tilde{B} = B(I + B^{-1}(a_j - a_i)e^T{}_i)$$

- Let $E = I + B^{-1}(a_j - a_i)e^T{}_i$, then $\tilde{B} = BE$

- (Continue the previous page) $E = I + B^{-1}(a_j - a_i)e^T{}_i$
- , which can be rewritten as:

$$E = I + (B^{-1}a_j - B^{-1}a_i)e^T{}_i$$

- Recall $a_j = Ne_j$, thus, $B^{-1}a_j = B^{-1}Ne_j = \Delta x_B$
- Also, $B^{-1}a_i = e_i$
- Therefore, $E = I + (\Delta x_B - e_i)e^T{}_i$

- Now, we can solve the system of equations:

$$\tilde{B}\Delta\tilde{x}_B = \tilde{a}_j$$

- Since $\tilde{B} = BE$, thus:

$$BE\Delta\tilde{x}_B = \tilde{a}_j$$

- Let $u = E\Delta\tilde{x}_B$. The $\Delta\tilde{x}_B$ can be solved in two steps:

$$Bu = \tilde{a}_j$$

$$E\Delta\tilde{x}_B = u$$

- Step 1. B and $\tilde{a}_j$ are given. Solve for u in the system of equations $Bu = \tilde{a}_j$
- Step 2. Next, we need to solve for $\Delta \tilde{x}_B$ in the system of equations $E\Delta \tilde{x}_B = u$
  - , which is equal to $\Delta \tilde{x}_B = E^{-1}u$
  - How to find the inverse of matrix E?

    $E = I + (\Delta x_B - e_i)e^T{}_i \qquad , E^{-1} = (I + (\Delta x_B - e_i)e^T{}_i)^{-1} = ?$
  - **Proposition 8.1** Given two column vectors u and v for which $1 + v^T u \neq 0$,

$$(I + uv^T)^{-1} = I - \frac{uv^T}{1 + v^T u}$$

(Proof)

$$(I + uv^T)(I - \frac{uv^T}{1 + v^T u}) = I + uv^T - \frac{uv^T}{1 + v^T u} - \frac{uv^T uv^T}{1 + v^T u} = I + uv^T(1 - \frac{1}{1 + v^T u} - \frac{v^T u}{1 + v^T u}) = I$$

- (Continue step 2) $E = I + (\Delta x_B - e_i) e^T{}_i$
- By Proposition 8.1: $(I + uv^T)^{-1} = I - \dfrac{uv^T}{1 + v^T u}$
- Let $\Delta x_B - e_i = u$ and $e_i = v$
- Thus, $E^{-1} = (I + (\Delta x_B - e_i) e^T{}_i)^{-1} = I - \dfrac{(\Delta x_B - e_i) e^T{}_i}{1 + e^T{}_i (\Delta x_B - e_i)} = I - \dfrac{(\Delta x_B - e_i) e^T{}_i}{1 + e^T{}_i \Delta x_B - e^T{}_i e_i} = I - \dfrac{(\Delta x_B - e_i) e^T{}_i}{\Delta x_i}$
- Recall $\Delta \tilde{x}_B = E^{-1} u$
- Therefore, $\Delta \tilde{x}_B = (I - \dfrac{(\Delta x_B - e_i) e^T{}_i}{\Delta x_i}) u = u - \dfrac{u_i}{\Delta x_i} (\Delta x_B - e_i)$

Example

- Suppose we select the third variable entering basis, and the new basis matrix is denoted as $\tilde{B}$. The column vector corresponding to the entering variable is:

$$\tilde{a}_j = [5 \quad 0 \quad 0 \quad 0 \quad -1]^T$$

- Solve step direction vector $\Delta \tilde{x}_B$ in two steps :
  - Step 1. Solve for $Bu = \tilde{a}_j$, we obtain $u = [0 \quad 3 \quad 1 \quad -3 \quad -\frac{1}{2}]^T$
  - Step 2. Solve for $\Delta \tilde{x}_B = E^{-1}u$

$$\Delta\tilde{x}_B = u - \frac{u_3}{\Delta x_3}(\Delta x_B - e_3) = \begin{bmatrix} 0 \\ 3 \\ 1 \\ -3 \\ -\frac{1}{2} \end{bmatrix} - \frac{1}{2}\left(\begin{bmatrix} -1 \\ 0 \\ 2 \\ 1 \\ -\frac{1}{2} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} \frac{1}{2} \\ 3 \\ \frac{1}{2} \\ -\frac{7}{2} \\ -\frac{1}{4} \end{bmatrix}$$

Homework

- Exercise 3.6 (degeneracy)
- Exercise 4.9 (efficiency)
- Exercise 6.1 (matrix notation)
- Exercise 8.1 (a) (b) (c) only
- Implement the reusing factorization algorithm
  - Input: A and B matrices, the current basic variable solution, and entering variable x_j and leaving variable x_i
  - Outputs: The new basic variable solution

Self-Practice

- **Part I.** Using CPLEX to obtain the optimal tableau of exercise 6.2

  (Hint) you may use C++ or python if the C# callable library is not existed

- **Part II.** Write down the following matrixes or values corresponding to the optimal tableau:

  (a) B

  (b) N

  (c) b

  (d) c_B

  (e) c_N

  (f) $B^{-1}N$

  (g) $x^*_B = B^{-1}b$

  (h) ζ^*