Notes for the Optimal Transport class

Quentin Bertrand, Mathurin Massias, Titouan Vayer

Last updated: December 10, 2025

Contents

1 Linear programming and the simplex algorithm

Resources: Nocedal and Wright (1999, Chapters 12 and 13)

1.1 Linear programming

Definition 1.1. *Let $m, n \in \mathbb{N}^*$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$. The following optimization problem is called a Linear Program (LP):*

$$\min_{x \in \mathbb{R}^n} \; c^\top x \quad \textit{s.t.} \quad Ax = b, \, x \geq 0 \, . \tag{1.1}$$

We consider only the setting where $m < n$: otherwise, there is much likely at most one solution to $Ax = b$, so the optimization problem is not very interesting. We also consider that A has rank m; if the rank is strictly smaller, then some constraints are redundant, and can be removed from the problem without affecting it.

Proposition 1.2 (Lagrangian and KKT)**.** *The Lagrangian writes*[a]*:*

$$\mathcal{L}(x, \lambda, \mu) = c^\top x + \lambda^\top (b - Ax) - \mu^\top x \,. \tag{1.2}$$

The KKT conditions, crucial to solve the LP, are:

$$c - A^\top \lambda - \mu = 0 \tag{1.3}$$
$$Ax = b \tag{1.4}$$
$$x \geq 0 \tag{1.5}$$
$$\mu \geq 0 \tag{1.6}$$
$$\mu_i x_i = 0 \quad \forall i \in [m] \tag{1.7}$$

[a]we use this formulation for consistency with existing textbooks, but beware in the case where the constraint is $Ax \leq b$ the second term will be $+\lambda^\top (Ax - b)$

Exercise 1.1. *Write down the derivation of the above KKT conditions for the LP* (1.2)*.*

We will not need the dual, but we derive it because we like duality. The reader in a rush can skip the remarks about duality. The dual of Problem (1.2) is obtained as:

$$\max_{\lambda, \mu \geq 0} \min_{x} \lambda^\top b + x^\top (c - A^\top \lambda - \mu) \tag{1.8}$$
$$\Leftrightarrow \max_{\lambda, \mu \geq 0} \lambda^\top b \quad \text{s.t.} \quad c - A^\top \lambda - \mu = 0 \tag{1.9}$$
$$\Leftrightarrow \max_{\lambda} \lambda^\top b \quad \text{s.t.} \quad c - A^\top \lambda \geq 0 \tag{1.10}$$
$$\Leftrightarrow \max_{\lambda \in \mathbb{R}^m} \lambda^\top b \quad \text{s.t.} \quad c \geq A^\top \lambda \tag{1.11}$$

Remark 1.3. *It may be convenient to keep the variable μ explicit, and write the dual as:*

$$\max \lambda^\top b \quad s.t. \quad A^\top \lambda + \mu = c \tag{1.12}$$

The reason is that by complementary slackness, we must have for a KKT point $\mu_i x_i = 0$, a property that is exploited by the simplex algorithm.

Remark 1.4 (Alternate forms)**.** *There is an alternate formulation of* (1.2)*, in which one uses the constraint $Ax \leq b$:*

$$\min_{x \in \mathbb{R}^n} c^\top x \quad s.t. \quad Ax \leq b,\, x \geq 0\,, \tag{1.13}$$

but it turns out (Exercise 1.2) that this form is equivalent to form (1.2)*. The dual for* (1.13) *is the same as before, except for a positivity constraint on λ:*

$$\max_{\lambda \in \mathbb{R}^m} \lambda^\top b \quad s.t. \quad c + A^\top \lambda \geq 0,\, \lambda \geq 0 \tag{1.14}$$

Also, some people write the problem as a maximization problem: $\max_{x \in \mathbb{R}^n} c^\top x$ s.t. $Ax = b$, $x \geq 0$. The dual is then $\min_y b^\top y$ s.t. $A^\top y \geq c, y \geq 0$, which can be obtained easily with the above results, using that the dual of the dual is the primal together with the substitution $(b, A, c) \to (c, -A^\top, -b)$.

Exercise 1.2. *By introducing a slack variable $z \geq 0$ such that $Ax + z = b$, check that problem (1.13) (with inequality constraint) is equivalent to a problem of the form (1.2) (with equality constraint) with the choice $\tilde{A} = (A, \mathrm{Id}_m) \in \mathbb{R}^{m\times(m+n)}$, $\tilde{x} = (x, z) \in \mathbb{R}^{n+m}$, $\tilde{b} = b$ and $\tilde{c} = (c, 0_m)$.*

By introducing two additional positive variables x_+ and x_- such that $x = x_+ - x_-$, show that (1.13) without the positivity constraint on x *is equivalent to a problem of the form (1.2) with another choice of $\tilde{A}$, $\tilde{x}$, $\tilde{b}$ and $\tilde{c}$.*

The reason why we focus on the KKT condition (1.3) – (1.7) is that they will equip us with useful tools to find solutions.

Proposition 1.5 (KKT conditions are necessary and sufficient)**.** *In Problem (1.2), the objective function is convex, the inequality constraints are convex and the equality constraints are affine, so the KKT conditions are sufficient (this is a generic result, see Prop. 8.36 in* `https://mathurinm.github.io/assets/2022_ens/class.pdf`*). Since all the constraints are linear, the KKT are also necessary*[a]*.*

[a]for generic constraints, one usually needs additional qualification conditions such as LICQ or Mangasarian-Fromowitz, the wikipedia page on KKT conditions has a nice list

Proof. Proof of sufficiency: If (x^*, λ^*, μ^*) satisfies the KKT conditions, then observe that the affine function $\mathcal{L}(\cdot, \lambda^*, \mu^*)$ is in fact constant as it has zero slope. From this, for any feasible $x \in \mathbb{R}^n$ and using the KKT conditions for x^*,

$$\mathcal{L}(x, \lambda^*, \mu^*) = \mathcal{L}(x^*, \lambda^*, \mu^*) \tag{1.15}$$

$$c^\top x + {\lambda^*}^\top (Ax - b) - {\mu^*}^\top x = c^\top x + {\lambda^*}^\top (Ax^* - b) - {\mu^*}^\top x^* \tag{1.16}$$

$$c^\top x - {\mu^*}^\top x = c^\top x^* \tag{1.17}$$

but the LHS is smaller than $c^\top x$ because both x and μ^* are positive, so $c^\top x \geq c^\top x^*$.

Therefore any feasible point has a higher objective value than x^*, which is feasible, which shows that x^* is optimal. □

We now introduce, as exercise, basic properties of the linear program. They are not needed in the rest of the section, but the proofs are short, elegant, and useful to get familiar with the problem and its KKT conditions.

Exercise 1.3. *Show that the primal value of any feasible point is greater than the dual value of any dual feasible point (weak duality holds). Show that the primal and the dual have the same set of KKT conditions.*
Show that if one of them has finite optimal value, so does the other, and their optimal values are equal (strong duality).
Show that if one is unbounded, the other is unfeasible.

1.2 The (revised) simplex algorithm

We now introduce the most famous algorithm to solve linear programs: the simplex algorithm. We first present it under its *revised* version, that relies on linear algebra notation. An equivalent formulation under a different form, the *tableau* simplex, will also be presented in **??**.

The simplex relies on special feasible points x, called basic feasible points.

Definition 1.6 (Basic feasible point)**.** *A basic feasible point (BFP) for the LP* (1.2) *is a point such that $Ax = b$, $x \geq 0$, and their exists a superset $\mathcal{B}$ of the support of x, such that $|\mathcal{B}| = m$ and $A_{:\mathcal{B}}$ is invertible. The set $\mathcal{B}$ is called a* basis.
We say that a BFP is nondegenerated if its support is exactly *of size m.*

Note that, following Nocedal and Wright (1999), we depart from the classical denomination "basic feasible *solution*", because we do not want to call "solution" a point that is not optimal.

Proposition 1.7 (Nocedal and Wright (1999, Thm. 13. 3))**.** *The basic feasible points are exactly the vertices of the dual polytope $\{x \in \mathbb{R}^m : Ax = b, x \geq 0\}$.*

The following exercise is not needed to understand the simplex algorithm and can be skipped. It shows that under reasonable conditions the problem has solutions, and basic feasible points, and solutions being basic feasible points.

Exercise 1.4 (Nocedal and Wright (1999, Thm 13.2))**.** *Show that if* (1.2) *has a nonempty feasible region, then there is at least one basic feasible point.*
Show that if (1.2) *has solutions, then at least one such solution is a basic feasible point.*
Hint: introduce p, the minimal number of non zero coordinates of any feasible point (resp. solution).

Description of the simplex algorithm The simplex algorithm aims at constructing a triplet satisfying the KKT conditions (1.3) – (1.7). By Proposition 1.5, this will yield a solution of Problem (1.2).
The algorithms starts with a point x that is a BFP (for now we take it for granted; we will see how to obtain one in Section 1.3). At each iteration, it modifies x and constructs μ and λ ensuring three things:

- x remains a BFP, so a fortiori $Ax = b$ and $x \geq 0$
- $c - A^\top \lambda - \mu = 0$
- $\forall i \in [n]$, $\mu_i x_i = 0$

Therefore, four out of the five KKT conditions will be satisfied at each iteration by design. The algorithm proceeds as long as the last KKT condition, $\mu \geq 0$, is not satisfied.

The simplex is presented in Algorithm 1 for completeness, but it is advised to not read it first, and only refer to it punctually when reading the explanations below.

Algorithm 1 Simplex (Phase II)

```
input  : $A$, $c$, $x$ a BFP
for $t = 0, \dots$ do
    $\mathcal{B} = \{j \in [n], x_j \neq 0\}$
    $\mathcal{N} = [n] \setminus \mathcal{B}$
    $B = A_{:\mathcal{B}} \in \mathbb{R}^{m \times m}$
    $N = A_{:\mathcal{N}} \in \mathbb{R}^{m \times (n-m)}$
    $\lambda = B^{\top^{-1}} c_{\mathcal{B}}$
    $\mu_{\mathcal{B}} = 0, \mu_{\mathcal{N}} = c_{\mathcal{N}} - N^\top \lambda$ (all conditions now satisfied except $\mu \geq 0$)
    if $\min_i \mu_i < 0$ then
        $q = \mathrm{argmin}_{i \in [n]} \mu_i$ (entering variable)
        $d = B^{-1} A_{:q}$
        $\delta = \min_{i, d_i > 0} \frac{(x_{\mathcal{B}})_i}{d_i}$ (if $d \leq 0$, problem is unbounded, stop)
        $x_{\mathcal{B}} = x_{\mathcal{B}} - \delta d$ (one variable leaves the support of $x$)
        $x_q = \delta$ (one variable enters the support of $x$)
    else
        stop
output: $x, \lambda, \mu$
```

We now explain the principle of the simplex algorithm. Given our current BFP x, we would like to construct λ and μ so that all KKT conditions are satisfied[1].

Steps to construct λ and μ from x, aiming for KKT conditions to hold Because x is a BFP, $Ax = b$ and $x \geq 0$ are satisfied. If we want complementary slackness (1.7) to hold, we must have $\mu_{\mathcal{B}} = 0$. So we first set $\mu_{\mathcal{B}} = 0$. Then, if we want the condition $c - A^\top \lambda - \mu = 0$ to hold, by restricting this system of n equations to rows in $\mathcal{B}$, we must have $c_{\mathcal{B}} - A_{:\mathcal{B}}^\top \lambda = 0$ (because $\mu_{\mathcal{B}}$ is 0). Hence, by writing $B := A_{:\mathcal{B}}$, which is invertible since x is a BFP by assumption, we have

$$\lambda = B^{-1^\top} x_{\mathcal{B}} \tag{1.18}$$

Now we can plug this back in $c - A^\top \lambda - \mu = 0$ to get the value of μ outside of $\mathcal{B}$: $\mu_{\mathcal{N}} = c_{\mathcal{N}} - N^\top \lambda$, where $\mathcal{N}$ is the complement of $\mathcal{B}$ in $[n]$ and $N := A_{:\mathcal{N}}$.

With these little algebraic manipulations, we were therefore able to construct, from a BFP x, a triplet x, μ, λ that satisfies all KKT conditions except maybe $\mu \geq 0$.

If our value of μ is positive, then we have found a KKT point, thus a solution. Otherwise, there exists a q such that $\mu_q < 0$; because of the way μ was constructed (namely we set $\mu_{\mathcal{B}} = 0$), we must have $q \notin \mathcal{B}$.

[1]we only try, it is not necessarily possible, as this would imply that x is optimal

The idea of the simplex algorithm is to try to fix this by increasing x_q (which currently is 0). While doing this, we aim at maintaining three properties: Ax must remain equal to b, x must remain positive, and x must remain a BFP, i.e. have m non zero entries. Let's find a direction and an update length under these constraints.

Finding an update direction d: To modify x while preserving $Ax = b$, we must thus move along a direction that is in the kernel of A. Wlog consider that $\mathcal{B} = \{1, \dots, m\}$, and so we have that the variable that will enter the support/basis q, is greater than $m + 1$. Since A has rank m, the $m + 1$ columns $(A_{:1}, \dots, A_{:m-1}, A_{:m}, A_{:q})$ are linearly dependent. There exist (d_1, d_{m-1}, d_m, d_q) not all equal to 0 such that $\sum_{i=1}^m d_i A_{:i} + d_q A_{:q} = 0$. Since $A_{:\mathcal{B}}$ is invertible, we cannot have $d_q = 0$ (this would imply that all other d_i's are zero). Hence, dividing by $-d_q$ and renaming the other d_i's to d_i, we can assume than $\sum_{i=1}^m d_i A_{:i} = A_{:q}$. With $d = (d_1, \dots, d_m) \in \mathbb{R}^m$, this rewrites as $Bd = A_{:q}$ (recall that $B := A_{:\mathcal{B}}$).

By construction of d, we see that increasing x_q from 0 to $\delta > 0$ and decreasing $x_{\mathcal{B}}$ by $-\delta d$ will keep Ax constant, since the variation induced in Ax by this change is $-\delta Bd + \delta A_{:q} = 0$.

How should we choose the update stepsize δ? We want to make a step as large as possible, until one coordinate in $\mathcal{B}$ vanishes, so that exactly one variable enters the basis (namely q), and another leaves[2]. It is easy to check that this is achieved by

$$\delta = \min_{i \in [m] : d_i > 0} \frac{(x_{\mathcal{B}})_i}{d_i} \,. \tag{1.19}$$

If all coordinates of d are negative, the we can pick δ arbitrarily large, and this means that the problem is unbounded (this is implied by the objective decrease property that we will prove in Equation (1.25)). Otherwise, since $x_i > 0$ on the basis, we have $\delta > 0$, so the algorithm moves, and by construction x is still a BFP. We can then proceed to the next iteration of the simplex algorithm.

Does the algorithm ever finish? One nice observations is that the objective strictly decreases at each iteration: denoting x^+ the value of the primal iterate after the update, since only the values at indices in $\mathcal{B} \cup \{q\}$ change, we have that the variation of the

[2]the careful reader will notice that two variables may leave at the same time, leading to x no longer being a nondegenrated BFP, as its support will be of size $m - 1$ isntead of m. Such situations can be covered but they require an extra level of technicality. We pretend that they don't happen – which is not crazy either, as such situations occur in very specific cases only.

objective is:

$$\begin{aligned}
c^\top(x^+ - x) &= -\delta c_{\mathcal{B}}^\top d + \delta c_q && && (1.20)\\
&= \delta(-c_{\mathcal{B}}^\top B^{-1} A_{:q} + c_q) && (d = B^{-1}A_{:q}) && (1.21)\\
&= \delta(-\lambda^\top A_{:q} + c_q) && (\lambda = {B^{-1}}^\top c_{\mathcal{B}}) && (1.22)\\
&= \delta(-(A^\top \lambda)_q + c_q) && && (1.23)\\
&= \delta(\mu_q - c_q + c_q) && (c = A^\top\lambda + \mu) && (1.24)\\
&= \delta\mu_q < 0 && && (1.25)
\end{aligned}$$

because $\mu_q < 0$ by construction of the algorithm, and $\delta > 0$. Thus, the algorithm can only visit each BFP/vertex at most once; since all iterates are BFP/vertices and there are a finite number of BFP/vertices, the simplex algorithm terminates.

1.3 How to find a basic feasible solution to initialize the algorithm?

The trick is very clever and elegant: in the so-called *Phase I* of the simplex algorithm[3], we will find a BFP by solving another LP, for which it is this time easy to find a BFP as initialization.

This auxiliary LP uses a lifted variable $(x, z) \in \mathbb{R}^{n+m}$:

$$\min_{x,z} 1_n^\top z \quad \text{s.t.} \quad Ax + Dz = b,\ x \geq 0,\ z \geq 0 \tag{1.26}$$

where $D = \operatorname{diag}(\operatorname{sign}(b))$.

For this, we can check that $(0_n, |b|)$ is a basic feasible point[4], that we can use a initialization. Now the objective function of this LP is clearly positive, and if the initial problem is feasible, for any feasible x one has that $(x, 0_m)$ gives an objective value of 0 for the auxiliary problem. Then the optimal value of the auxiliary LP (1.26) is 0, which means that for any solution[5] (x^*, z^*), we must have $z^* = 0$, and so $Ax^* = b$. In addition since there are m constraints in the auxiliary LP, (x^*, z^*) has m non zero entries, and so so does x^*.

Hence, solving the auxiliary LP provides us with a BFP for the original LP: a feasible point with m non zero entries.

Remark 1.8. *For a problem with the constraint $Ax \leq b$, one can use as auxiliary problem:*

$$\min_{x_0,x} x_0 \quad s.t. \quad Ax \leq b + x_0 1_m,\ x_0 \geq 0,\ x \geq 0 \tag{1.27}$$

[3] *Phase II* is the procedure described in the previous section, which requires a BFP for initialization

[4] that's why we needed the signs in D, because of the positivity constraint

[5] a degenerated case with non unique solution is possible, but addressable

we can create a BFP py picking $x = 0$, x_0 large enough ($-\min_i b_i$), which sets exactly one slack variable to 0. We are left with $1 + (m - 1)$ nonzero variables, the point is feasible: all good.

If instead we want can only use a solver working with the equality constraint, we use the lifted formulation using slack variables $z \in \mathbb{R}^n$ together with an initialization $x = 0_m$, $x_0 = -\min_i b_i$ so that all z_i are non zero except one.

2 Graph background

Definition 2.1.

- *A tree is a connected and acyclic subgraph. Equivalently, it is acyclic and such that, if any edge is added, a cycle is formed. Equivalently still, it is connected but would become disconnected if one single edge is removed from it.*
- *A spanning tree of a graph is a subgraph that is a tree and contains all nodes of the original graph (remember it must be connected, so all nodes have a related edge).*

Proposition 2.2. *(i) A connected graph has an Euler cycle (a cycle that uses each edge exactly once) if and only if every node has an even degree. The graph is then said to be Eulerian.*

(ii) A tree must have a leaf (in fact, at least 2 if it has more than 2 vertices).

(iii) A spanning tree on m nodes has $m - 1$ edges

(iv) Deleting a leaf from a tree (edge and node) leads to a tree (or an empty graph).

Proof. (ii): Assume that the tree T has no leaf. No tree has all even nodes, otherwise it would be an Eulerian graph and thus have a cycle. By the handshaking lemma, it must have two odd nodes, u and v. Since there are no leaves, those two have degree at least three. Pick a path P from u to v, containing no odd nodes (if it contains odd nodes, change u or v to one such node to end up with a path with only odd nodes). Remove the edges in P form T. Now u and v have even degrees, and the parity of the other degrees does not change.

Repeat as long as there are odd degree nodes: we will end up with a graph where all degrees are even, thus Eulerian, meaning that there is a cycle, contradicting the fact that T is a tree. To prove that there are 2 leaves, if it has only one: connect it to an odd degree node (thus of degree more than 3), remove the path as above. We now have a tree with one less leaf, if we repeat enough times we'll end up with a tree without leaf, contradiction.

(iii) A tree on m vertices must have $m - 1$ edges (proof by induction). When it is a spanning tree, it has the same number of vertices as the original graph, which concludes the proof.

(iv) seems trivial □

3 The network simplex

Resource: (Vanderbei, 1998, Chap. 14)

It is a special version of the simplex, in a case where the constrain matrix A has a very particular form, stemming from a problem in graph theory called the *minimum-cost network flow problem.*

Definition 3.1 (Minimum-cost network flow problem)**.** *Consider a directed graph* (E, V) *where each node* $k \in V$ *is equipped with a supply* $b_k \in \mathbb{R}$ *(called a demand if it is negative), such that* $\sum_{k\in V} b_k = 0$*. The problem is:*

$$\min \sum_{(i,j)\in E} x_{ij}c_{ij} \quad s.t. \quad \sum_{(i,k)\in E} x_{ik} - \sum_{(k,j)\in E} x_{kj} = -b_k,\ \forall k \in V \tag{3.1}$$

The first term in the LHS of the constraint in the incoming flow of node k, while the second is the outgoing one. Summing the constraints over k, we see indeed that $\sum b_k = 0$ is a necessary condition for the problem to be feasible.

In this problem, each variable x_{ij} represents the quantity that travels in edge $(i, j) \in E$; each x_{ij} is present exactly three times in the constraints: one for the positivity, one as outgoing edge of i and one as incoming edge for j. We can write (3.1) as a linear programme, introducing the constraint matrix $A \in \mathbb{R}^{|V|\times|E|}$, as the node-edge adjacency matrix: for each $e = (i, j) \in V$, we set $A_{ie} = -1$ and $A_{je} = 1$. For consistency with Section 1, we will write $|V| = m$ and $|E| = n$; it is reasonable to consider $n > m$ here too.

Here comes a caveat: contrary to what we have used in Section 1, the columns of A are not independent, as summing all columns yields the 0 vector. This will require additional precautions at some steps.

The dual of (3.1) is $\max_y -b^\top y$ s.t. $A^\top y \le c$, for which we introduce a slack variable and use the explicit form of A:

$$\max -b^\top y \quad \text{s.t.} \quad y_i - y_j + z_{ij} = c_{ij},\ z \ge 0 \tag{3.2}$$

The complementary conditions are $x_{ij}z_{ij} = 0$ (see Remark 1.3, z being μ).

We see that, since $b^\top 1_m = 0$ and $A^\top 1_m = 0_n$, the dual is invariant by translation of y, so we will regularly impose $y_r = 0$ for some peculiar node r, called the root node, in order to have well-defined unique solutions.

Steps of the primal network simplex algorithm Key property: a spanning tree of the graph corresponds exactly to a basis of A. With the small change that bases are of size $m - 1$ for A, as A does not have full row rank, but rank $m - 1$. Pick a node, delete the flow constraint here. Denote $\tilde{A}$ and $\tilde{b}$ the corresponding matrix and vector. Then a square submatrix of $\tilde{A}$ is a basis iff the arcs its columns correspond to form a spanning tree (see that finding a solution to $Bx = u$ is the same as filling assignemnt, and this can be done in a unique manner starting from leaf nodes (it somehow amounts to finding a permutation of the rows and columns of B to make it lower triangular)).

Dual Corresponding to any basis there is also a dual solution: first we work on the spanning tree, where since $z_{ij} = 0$ by complementary slackness, the dual feasability condition is simply $y_i - y_j = c_{ij}$. Since the spanning tree is, well, a tree, we can compute the values of the y_i starting from the root node, for which we have assigned $y_r = 0$ arbitrarily, and then we travel along the tree: easy!

Then we assign the z_{ij} using the values of y_i found (all y_i are computed because we have a spanning tree). Notice that those are the classical simplex steps: compute primal solution on basis, then λ (y here), then μ (z here) outside of the primal support.

z is not necessarily positive: we find an edge for which is is not; this is necessarily not in the spanning tree; we add it to the spanning tree. This must create a cycle (by characterization of a spanning tree). We will remove one edge from this cycle, by adding/subtracting (depending on orientation) the same quantity to every edge in the cycle. We do so until one edge has 0 flow (easy to see that it is the edge pointing in a direction opposite to the added edge, with minimal current flow). See (Vanderbei, 1998, Fig. 14.7) for a clear picture. So with this modification we have added a new edge, removed one (we still have a spanning tree), and the primal flow still satisfies the constraints.

Now it remains to update the dual variable y. When we remove the edge (not adding the other), we partition the graph into two subtree. The dual variables on the subtree containing the root node should not change (their values is already determined by the choice we made for the root node). All other variables (those in the subtree not connected to the root) must all be shifted by the same amount; we determine which amount by looking at the condition $\tilde{y}_i - \tilde{y}_j = c_{ij}$ for the newly added node (i, j); while before we add $y_i - y_j + z_{ij} = c_{ij}$, so it tells us that the shift amount is z_{ij} if vertex i is in subtree 2, or $-z_{ij}$ otherwise.

Balanced flow: satisfies the balance equation at every node (not necessarily while being positive). If in addition it is positive, it is a feasible flow. A tree solution is a balanced flow supported by a spanning tree.

References

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

Robert J Vanderbei. Linear programming: foundations and extensions. *Journal of the Operational Research Society*, 49(1):94–94, 1998.