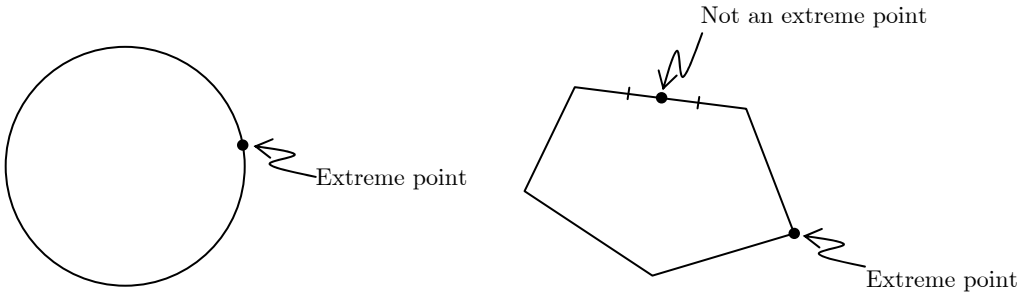
The set $\left\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = A\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0, A \in \mathbb{R}^{n \times m}, \boldsymbol{\alpha} \in \mathbb{R}^m\right\}$ is the *cone generated by the columns of A*.

- ***Definition (extreme point)***: An *extreme point* of the convex set $\mathcal{C}$ is a point $\boldsymbol{x} \in \mathcal{C}$ that cannot be written as a convex combination of other points in $\mathcal{C}$.



- ***Definition (Convex Combination)***: A *convex combination* of points $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k$ is a point $\boldsymbol{x} = \sum_{i=1}^k \lambda_i \boldsymbol{x}_i$, such that $\boldsymbol{\lambda} \geq 0$ and $\sum_{i=1}^k \lambda_i = 1$. The set of convex combinations of a set points is the smallest convex set containing all the points; it is called the *convex hull* of these points.
- ***Definition (Hyperplane)***: The set $\mathcal{H} = \left\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a} \cdot \boldsymbol{x} = \beta, \boldsymbol{a} \in \mathbb{R}^n, \beta \in \mathbb{R}\right\}$ is called a *hyperplane* with *normal* $\boldsymbol{a}$. The set $\bar{\mathcal{H}} = \left\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a} \cdot \boldsymbol{x} \leq \beta\right\}$ is a *closed halfspace*, and $\mathcal{H}$ is its *bounding hyperplane*.
- ***Definition (Afine set)***: A set $\mathcal{C}_a \in \mathbb{R}^n$ is an *affine set* if for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{C}_a$ and $\lambda \in (-\infty, \infty)$, $\boldsymbol{x}(\lambda) = \lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2 \in \mathcal{C}_a$. A hyperplane is an example of an affine set. Roughly speaking, an affine set is a subspace that need not contain the original.
- ***Definition (Polyhedron)***: A *polyhedron* is a set which is the intersection of a finite number of closed hyperplanes. It is necessarily convex. If the polyhedron is non-empty and bounded (ie: there exists a large ball it lies inside of), it is called a *polytope*.
- ***Definition (Dimension)***: The *dimension* of an affine set $\mathcal{C}_a$ is the maximum number of linearly independent vectors in $\mathcal{C}_a$.
- ***Definition (Supporting hyperplane)***: A *supporting hyperplane* of a closed, convex set $\mathcal{C}$ is a hyperplane $\mathcal{H}$ such that $\mathcal{H} \cap \mathcal{C} \neq \varnothing$ and $\mathcal{C} \subseteq \bar{\mathcal{H}}$:

turns out that an appropriate starting point is simply the point in the polyhedron with the smallest number of strictly positive components. This *must* be a vertex, because if it was not, we could carry out the steps outlined above and find a point with fewer strictly positive components – this is a contradiction.

[Note that it is *not* always the case a polyhedron must have vertices – for example, the polyhedron $\left\{\boldsymbol{x} \in \mathbb{R}^2 : x_1 \geq 0, x_1 \leq 1\right\}$ has no vertices. However, the non-negativity constrains of the standard-form polyhedron ensure there is at least one.

- ***The Fundamental Theorem***
  - ***Theorem (Fundamental Theorem of Linear Programming)***: If $\mathcal{P} \neq \varnothing$, then the minimum $\min_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{c} \cdot \boldsymbol{x}$ is either attained at a vertex of $\mathcal{P}$ or unbounded.

    Proof: We consider two cases:

    - ***Case 1 – $\mathcal{P}$ has a recession direction $\boldsymbol{d}$ such that $\boldsymbol{c} \cdot \boldsymbol{d} < 0$***: in that case, the problem is unbounded because for any $\bar{\boldsymbol{x}} \in \mathcal{P}$, $\boldsymbol{c} \cdot g(\theta) = \boldsymbol{c} \cdot \left(\bar{\boldsymbol{x}} + \theta \boldsymbol{d}\right) = \boldsymbol{c} \cdot \bar{\boldsymbol{x}} + \theta \boldsymbol{c} \cdot \boldsymbol{d} \downarrow -\infty$ as $\theta \to \infty$.
    - ***Case 2 – $\mathcal{P}$ has no such recession direction***: in that case, consider any point $\bar{\boldsymbol{x}} \in \mathcal{P}$. By our Representation Theorem, we can write $\bar{\boldsymbol{x}} = \sum \lambda_i \boldsymbol{v}^i + \alpha \boldsymbol{d}$, where $\lambda_+ = 1, \lambda_i \geq 0, \alpha \geq 0$. We then have

$$\begin{aligned} \boldsymbol{c} \cdot \bar{\boldsymbol{x}} &= \sum \lambda_i \boldsymbol{c} \cdot \boldsymbol{v}^i + \alpha\left(\boldsymbol{c} \cdot \boldsymbol{d}\right) \\ &\leq \sum \lambda_i \boldsymbol{c} \cdot \boldsymbol{v}^i \\ &\leq \sum \lambda_i \min_{i \in v}\left(\boldsymbol{c} \cdot \boldsymbol{v}^i\right) \\ &= \min_{i \in v}\left(\boldsymbol{c} \cdot \boldsymbol{v}^i\right) \end{aligned}$$

    Thus, the minimum is indeed attained at a vertex.

Simplex

- We have thus far established that the optimum of a linear program occurs at one of the vertices of the feasible region. We now consider the *simplex algorithm*, an efficient method of jumping for vertex to vertex while constantly improving the objective function.

- ***Representation in terms of emanating directions***
  - Consider a polyhedron $A\boldsymbol{x} = \boldsymbol{b}$, where $A = \left[B, N\right] \in \mathbb{R}^{m\times n}$, and a non-degenerate basic solution $\hat{\boldsymbol{x}} = \left(\hat{\boldsymbol{x}}_B^\top, \hat{\boldsymbol{x}}_N^\top\right)^\top$, where $\hat{\boldsymbol{x}}_B = B^{-1}\boldsymbol{b} > \boldsymbol{0}$ and $\hat{\boldsymbol{x}}_N = \boldsymbol{0}$.
  - ***Claim***: Consider the matrix

$$M = \begin{pmatrix} B & N \\ 0 & I \end{pmatrix} \qquad\qquad M\hat{\boldsymbol{x}} = \begin{pmatrix} B & N \\ 0 & I \end{pmatrix}\begin{pmatrix} \hat{\boldsymbol{x}}_B \\ \hat{\boldsymbol{x}}_N \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ 0 \end{pmatrix}$$

The last $n - m$ columns of M^{-1} (ie: from column $m + 1$ onwards) are the directions of the edges of P emanating from the basic feasible solution $\hat{\boldsymbol{x}}$.

Proof: Let $\boldsymbol{\eta}^j$ be the j^{th} column of M^{-1}. Using the fact that since there is no degeneracy, $\boldsymbol{x}_B$ has M nonzero components, and so the row is clearly from the second half of the matrix above, we can write, for $j > m$:

$$\boldsymbol{\eta}^j = M^{-1}e_j = \begin{pmatrix} B^{-1} & -B^{-1}N \\ 0 & I \end{pmatrix} e_j = \begin{pmatrix} -B^{-1}N \\ I \end{pmatrix} e_{j-b} = \begin{pmatrix} -B^{-1}\boldsymbol{A}^{\text{col } j} \\ \vdots \\ 1 \\ \vdots \end{pmatrix} \leftarrow j^{\text{th row}}$$

Now, consider moving in the direction $\boldsymbol{\eta}^j$ by an amount θ; $\boldsymbol{x}(\theta) = \hat{\boldsymbol{x}} + \theta\boldsymbol{\eta}^j$. This point is still on the polyhedron, because

$$\begin{aligned} A\boldsymbol{x}_B(\theta) &= B\boldsymbol{x}_B(\theta) + N\boldsymbol{x}_N(\theta) \\ &= B\left(\hat{\boldsymbol{x}} - \theta B^{-1}\boldsymbol{A}^{\text{col } j}\right) + \theta\boldsymbol{A}^{\text{col } j} \\ &= \boldsymbol{b} - \theta\boldsymbol{A}^{\text{col } j} + \theta\boldsymbol{A}^{\text{col } j} = \boldsymbol{b} \end{aligned}$$

Thus, η_j is indeed an edge of P, and it clearly results from increasing only *one* of the $\boldsymbol{x}_N$.

For a geometric interpretation, consider that the rows of M contain the vectors normal to every active constraint at the BFS, and that $MM^{-1} = I \Rightarrow \boldsymbol{m}_{\text{row } i}\boldsymbol{M}^{-1}_{\text{col } j} = 0\ \forall i \neq j$. This means that our emanating edges (columns of M^{-1}) are perpendicular to every normal vector save one (along which we're trying to move):

- We can also ascribe an economic interpretation to the dual program itself. Two examples:
  - In a transportation problem where each constraint corresponds to supply at a source or sink, the dual variables can be interpreted as the cost an external contractor would charge to handle the transportation of one unit away from a source or towards a sink. The constraint requires the total cost of transportation of unit material from a given source to a given sink be less than or equal to *our* cost for transportation along that path.
  - In a diet problem, where each problem corresponds to a given nutrient, the dual variables can be interpreted as the cost of a pill containing a unit amount of the said nutrient.

- ***The Dual Simplex Algorithm***
  - The primal simplex algorithm effectively involved journeying from solution to solution while maintaining primal feasibility and complementary slackness and looking for dual feasibility. The dual simplex algorithm does the opposite – it keeps dual feasibility (ie: primal optimality) and complementary slackness and looks for primal feasibility.
  - Consider a basis consisting of m linearly independent columns of A, with the following tableau:

$B^{-1}A$	$B^{-1}\boldsymbol{b}$
$\overline{\boldsymbol{c}}^T$	$-\boldsymbol{c}_B^T B^{-1}\boldsymbol{b}$

Or, in more detail:

$\vdots$		$\vdots$	$x_{B,1}$
$B^{-1}\boldsymbol{A}^{\text{col } 1}$	$\cdots$	$B^{-1}\boldsymbol{A}^{\text{col } n}$	$\vdots$
$\vdots$		$\vdots$	$x_{B,m}$
$\overline{c}_1$	$\cdots$	$\overline{c}_n$	$-\boldsymbol{c}_B^T\boldsymbol{x}_B$

We consider a solution which might be primal infeasible (ie: some of the x_B may be negative) but primal optimal (ie: all the reduced costs are positive).

- Otherwise, it is a simple matter to add an extra column to the simplex tableau and perform a few more iterations.

- ***Adding a new inequality constraint***
  - Consider adding a new constraint $\boldsymbol{a}^{\text{row } m+1} \cdot \boldsymbol{x} \geq b_{m+1}$. If $\boldsymbol{x}^*$ satisfies this constraint, then it is an optimal solution to the new problem.
  - If not, we introduce a new nonnegative slack variable, and re-write $\boldsymbol{a}^{\text{row } m+1} \cdot \boldsymbol{x} - x_{n+1} = b_{m+1}$. The matrix A is then replaced by

$$\bar{A} = \begin{bmatrix} A & \boldsymbol{0} \\ \boldsymbol{a}^{\text{row } m+1} & -1 \end{bmatrix}$$

We introduce a new basis that includes all the variables in B *plus* our new slack variable. This gives

$$\bar{B} = \begin{bmatrix} B & \boldsymbol{0} \\ \boldsymbol{a}_B^{\text{row } m+1} & -1 \end{bmatrix} \qquad \bar{B}^{-1} = \begin{bmatrix} B^{-1} & \boldsymbol{0} \\ \boldsymbol{a}_B^{\text{row } m+1} B^{-1} & -1 \end{bmatrix}$$

The basic solution is $\left(\boldsymbol{x}^*, \boldsymbol{a}_B^{\text{row } m+1} \boldsymbol{x}^* - x_{m+1}\right)$ – and it is not feasible, since the original constraint was not satisfied.

  - We want to figure out a way to add this new constraint to our tableau (which, recall, contains $B^{-1}A$). First, consider the problem algebraically. We have that

$$\bar{B}^{-1}\bar{A} = \begin{bmatrix} B^{-1}A & 0 \\ \boldsymbol{a}_B^{\text{row } m+1} B^{-1}A - \boldsymbol{a}^{\text{row } m+1} & 1 \end{bmatrix}$$

And also that the reduced cost do not change, because the objective coefficient of the new slack variable is 0:

$$\begin{bmatrix} \boldsymbol{c}^\top & 0 \end{bmatrix} - \begin{bmatrix} \boldsymbol{c}_B^\top & 0 \end{bmatrix} \bar{B}^{-1}\bar{A} = \begin{bmatrix} \boldsymbol{c}^\top - \boldsymbol{c}^\top B^{-1}A & 0 \end{bmatrix}$$

  - The above is hardly useful in terms of practically writing the new tableau. More informatively, we be describe the above in terms of row operations:
    - Add a new row to the tableau simply consisting of $\begin{bmatrix} \boldsymbol{a}^{\text{row } m+1} & -1 \end{bmatrix}$
    - Perform the row operations necessary to ensure that the columns of $\bar{B}^{-1}\bar{A}$ that correspond to basic variables form the identity matrix. In particular, this involves:
      - Multiplying the row by –1, to obtain a 1 in the last column.

$$\begin{aligned} z(\theta_3) &= \left(\boldsymbol{c} + \lambda\theta_1\boldsymbol{d} + (1-\lambda)\theta_2\boldsymbol{d}\right)^\top \boldsymbol{x}^*(\theta_3) \\ &= \lambda\left(\boldsymbol{c} + \theta_1\boldsymbol{d}\right)^\top \boldsymbol{x}^*(\theta_3) + (1-\lambda)\left(\boldsymbol{c} + \theta_2\boldsymbol{d}\right)^\top \boldsymbol{x}^*(\theta_3) \\ &\geq \lambda z(\theta_1) + (1-\lambda) z(\theta_2) \end{aligned}$$

And so z is concave.

Network flow problems

- ***The network flow problem***
  - Let $I(j) = \{(i,j) : (i,j) \in \mathcal{A}\}$ be the set of edges *incoming* into i, and $O(i) = \{(i,j) : (i,j) \in \mathcal{A}\}$ be the set of edges *leaving* node i. Let b_i the *supply* at node i, that enters from the outside. Then the network flow problem is
    $\min \sum_{(i,j)\in\mathcal{A}} c_{ij} f_{ij}$ s.t. $\sum_{(i,j)\in I(j)} f_{ij} - \sum_{(i,j)\in O(i)} f_{ij} = b_j \;\; \forall j \in \mathcal{N}, 0 \leq f_{ij} \leq u_{ij}$
  - The first constraint can concisely be expressed as $A\boldsymbol{f} = \boldsymbol{b}$ where each *row* of A represents a node and each column represents an arc. a_{ij} contains 1 if arc j leads to node i, a –1 if arc j leaves from node i, and a 0 otherwise.
  - To deal with lower bound m_{ij} on flows, simply define $\overline{f}_{ij} = f_{ij} - m_{ij}$, $\overline{u}_{ij} = u_{ij} - m_{ij}$ and $\overline{\boldsymbol{b}} = \boldsymbol{b} - A\boldsymbol{m}$
  - The dual of the un-capacitated problem is max $\boldsymbol{b}\cdot\boldsymbol{p}$ s.t. $A^\top \boldsymbol{p} \leq \boldsymbol{c}$.
    - Due to the structure of A, we in fact have $p_i - p_j \leq c_{ij} \;\; \forall (i,j) \in \mathcal{A}$. Note that adding or removing a constant from each p_i keeps the solution feasible, and has to effect on the objective (because $\boldsymbol{1}\cdot\boldsymbol{b} = 0$). As such, we can assume $p_n = 0$
    - Complementary slackness requires that $p_i - p_j = c$ for all arcs on which something flows (ie: on which $f_{ij} \neq 0$). These can therefore easily be calculated by setting $p_n = 0$ and backtracking through arcs with flow.
    - The p_i represent shadow prices of increasing b_i by a certain amount.
- ***Network flow algorithms***
  - A *circulation* is a flow vector $\boldsymbol{h}$ such that $A\boldsymbol{h} = \boldsymbol{0}$. The cost of such a circulation is $\boldsymbol{c}\cdot\boldsymbol{h}$, and "pushing" θ units of the flow means setting $\boldsymbol{f} \leftarrow \boldsymbol{f} + \theta\boldsymbol{h}$.