Unit 2

1 Linear Programming

In a Linear Program (LP) all the functions defining the objective and the constraints are linear and the variables move in $\mathbb{R}^n$. A general form for an LP is the following:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \leq b_j, \quad j = 1, \ldots, m \\ & x \in \mathbb{R}^n. \end{aligned}$$

The feasible set of an LP, being defined by linear constraints, is called **polyhedron**.

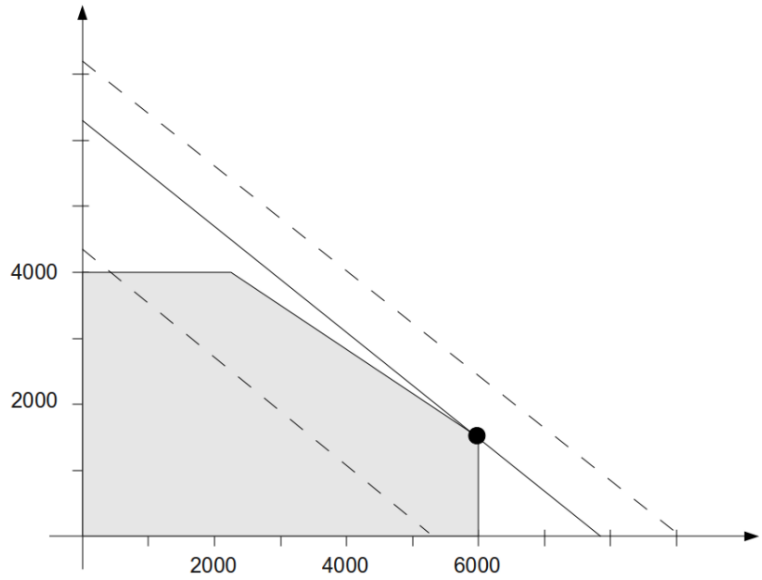
LPs are widely used in practical applications because they are easy to understand and there are many efficient and effective tools to compute their optimal points (e.g. GLPK).

1.1 Graphical solution

LPs up to two variables can be solved graphically. For example the LP

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & -25x_1 - 30x_2 \\ \text{subject to} \quad & 7x_1 + 10x_2 \leq 56000 \\ & 0 \leq x_1 \leq 6000 \\ & 0 \leq x_2 \leq 4000 \end{aligned}$$

has this graphical representation



The optimal point is $x = (6000\ 1400)^T$. [Check it numerically with GLPK]

Practical rules:

- the x-axis on the graph shows the range of values of x_1, while the y-axis the range of values of x_2;
- the box constraints define a rectangle on the graph, the other constraints can only further cut this rectangle;
- a generic constraint $a_{1j}x_1 + a_{2j}x_2 \leq b_j$ can be drawn following these steps:
  1) draw the line passing through the points $\begin{pmatrix} 0 & \frac{b_j}{a_{2j}} \end{pmatrix}^T$ and $\begin{pmatrix} \frac{b_j}{a_{1j}} & 0 \end{pmatrix}^T$, this line defines two half-planes
  2) consider any point $\overline{x}$ not on the line (the origin is the first choice if available): if $a_{1j}\overline{x}_1 + a_{2j}\overline{x}_2 < b_j$ then cut the half-plane not containing $\overline{x}$, otherwise, if $a_{1j}\overline{x}_1 + a_{2j}\overline{x}_2 > b_j$, then cut the half-plane containing $\overline{x}$;
- after having drawn the box and all the constraints, the feasible set (the feasible polyhedron) is ready;
- the objective function $c_1x_1 + c_2x_2$ is represented in the graph with its level sets:
  1) consider a suitable value C for the objective function: draw the line passing through the points $\begin{pmatrix} 0 & \frac{C}{c_2} \end{pmatrix}^T$ and $\begin{pmatrix} \frac{C}{c_1} & 0 \end{pmatrix}^T$, all the points on this line have an

objective value equal to C (we say that the points on the line constitute the level set of value C)

2) any other line parallel to this one is a different level set of the objective function;

- the direction $(0 \quad 0)^T \to (c_1 \quad c_2)^T$ on the graph goes towards greater values of the level sets: if the objective must be maximized, then the best level set is the one with maximum value on the feasible set, vice versa, if the objective must be minimized, then the best level set is the one with minimum value on the feasible set;

- all the points in the graph that are on the line representing the best level set and inside the feasible set are optimal points of the LP.

Remarks:

- If after all the cuts given by the constraints the feasible set is empty, then the LP is an infeasible problem (see Unit 1).

- The LP could not have a full box. In this case some sides of the rectangle are not present in the graph and the feasible set could be unbounded. If this happens then the LP could not admit any optimal point because it is impossible to indentify the best level set. Specifically, it is a case in which the LP is an unbounded problem (see Unit 1).

- The set of optimal points could consist of a single point (in this case it is a vertex of the polyhedron, see the next sections), a segment (whose end points are vertices of the polyhedron, see the next sections), an half line (whose initial point is a vertex of the polyhedron, see the next sections), a line, or the entire feasible polyhedron.

[Give practical simple examples in which all these cases occurs.]

1.2 Vertices

Roughly speaking the vertices of a polyhedron $P \triangleq \{x \in \mathbb{R}^n : \sum_{i=1}^n a_{ij}x_i \leq b_j, \quad j = 1, \ldots, m\}$ are its "corners". Formally we give both a geometric and an algebraic characterization of vertex of a polyhedron.

Definition 1.1 *(Geometric characterization of vertex of a polyhedron P)*
$\overline{x}$ is a vertex of P if

(i) $\overline{x} \in P$,

(ii) $\nexists\, y, z \in P$ *such that* $y \neq \overline{x}$, $z \neq \overline{x}$, $y \neq z$, *and* $\overline{x} \in [y, z]$.

Remark: given $y, z \in \mathbb{R}^n$, the set $[y, z] \triangleq \{x \in \mathbb{R}^n : x = (1 - \lambda)y + \lambda z, 0 \leq \lambda \leq 1\}$ is said segment with end points y and z.

Theorem 1.2 *(Algebraic characterization of vertex of a polyhedron P)*
$\overline{x}$ is a vertex of P if and only if

(i) $\overline{x} \in P$,

(ii) *the set of indices* $J_{\overline{x}} \triangleq \{j \in \{1, \ldots, m\} : \sum_{i=1}^{n} a_{ij} x_i = b_j\}$ *of the active constraints at* $\overline{x}$ *contains at least* n *elements,*

(iii) *the vectors* a_{*j}, *with* j *in the set of indices* $J_{\overline{x}}$ *of the active constraints at* $\overline{x}$, *are linearly independent.*

Remarks:

- Considered together, conditions (ii) and (iii) in Theorem 1.2 are equivalent to require that $\text{rank}(a_{*j})_{j \in J_{\overline{x}}} = n$.
- P contains **at most** $\begin{pmatrix} m \\ n \end{pmatrix} = \frac{m!}{n!(m-n)!}$ vertices.

  [Explain why this is a direct consequence of Theorem 1.2.]
- P contains **at least one** vertex if and only if it does not contain any line. Any LP can be equivalently modified making its feasible set not containing any line.

1.3 Theoretical foundations

In an LP, if optimal points and vertices exist, then at least one optimal point is a vertex of the polyhedron. This is the most important property of an LP, because it is possible to examine only the vertices of the polyhedron in order to compute an optimal point of the problem. And the set of the vertices is finite! These theoretical basis are given by the fundamental theorem of linear programming.

Theorem 1.3 *(Fundamental theorem of linear programming)*
Assume that P does not contain any line. **One and only one** *of the following statements is true:*

(i) *the LP is infeasible,*

(ii) *the LP is unbounded,*

(iii) *at least one of the vertices of P is an optimal point of the LP.*

Remarks:

- Assuming P not containing any line is equivalent to assuming the existence of the vertices. If this assumption is omitted then statement (iii) in Theorem 1.3 can be rephrased: "the LP admits optimal points".

- Computing one single optimal point of an LP is sufficient to characterize its **entire set of optimal points**. Specifically, the set of optimal points of an LP is the following polyhedron:

$$\begin{cases} \sum_{i=1}^{n} c_i x_i = \sum_{i=1}^{n} c_i \overline{x}_i \\ \sum_{i=1}^{n} a_{ij} x_i \leq b_j, \quad j = 1, \ldots, m, \end{cases}$$

  where $\overline{x}$ is any optimal point of the LP.

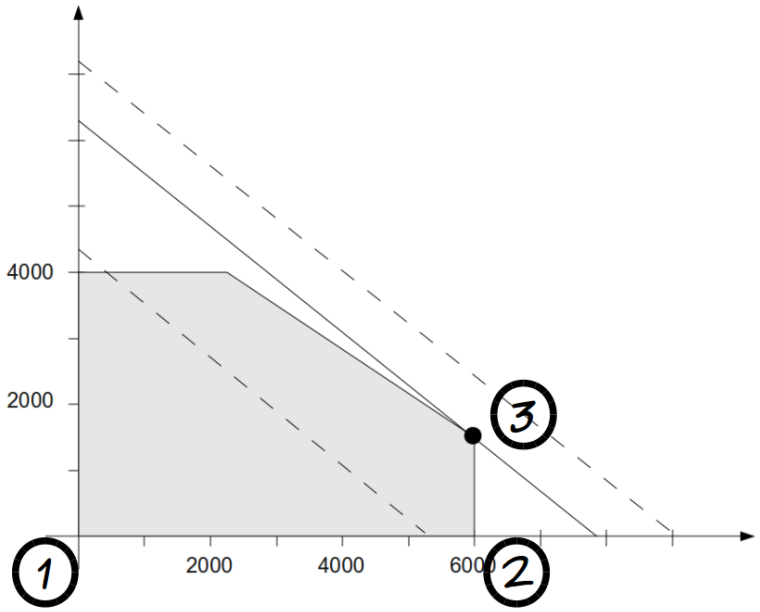
1.4 The simplex algorithm in brief

Almost all software to compute optimal points of LPs are based on the simplex algorithm. It is a very effective and efficient method to check if an LP either is infeasible, is unbounded, or admits optimal points. In the latter case it returns a vertex that is an optimal point for the LP. The power of the simplex algorithm lies in the fact that it takes always a finite number of computations to successfully stop. And this is a consequence of the fact that it jumps only from one vertex to another without backtracking.

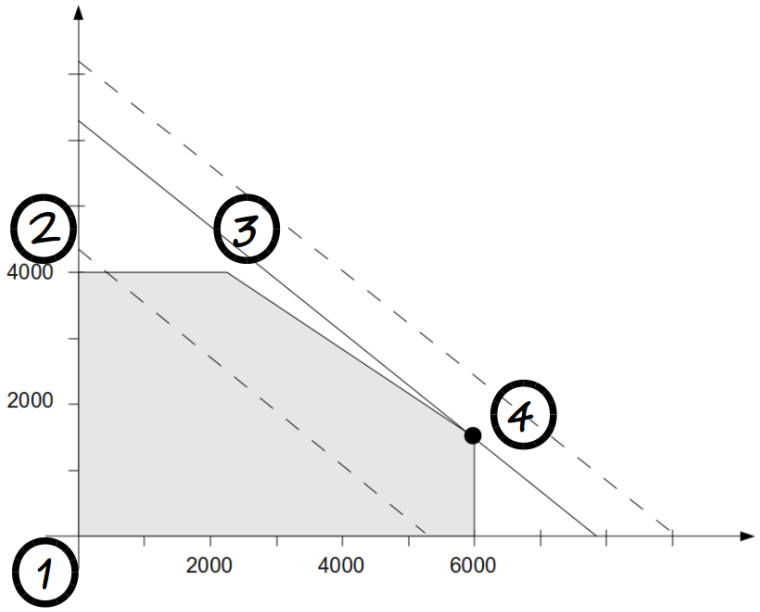
In brief the simplex algorithm performs the following steps that require a finite number of computations:

1. Modify the problem making the feasible set not containing any line.
2. Check if the feasible set is empty: in this case stop, the problem is **infeasible**.
3. Compute a first vertex.
4. Check if moving along an adjacent edge the objective function gets better levels: in this case, if such edge ends in a vertex then move on it and repeat step 4, otherwise stop, the problem is **unbounded**.
5. Stop, the current vertex is an **optimal point** of the LP. (The optimality is guaranteed by the convexity of the problem, see the forthcoming section about convexity.)

The simplex algorithm, applied on the example in section 1.1 and starting from the vertex $(0 \;\; 0)^T$, jumps on the vertices depicted in the following picture



but also this different way is correct



it depends on the specific implementation.

2 Product mix with alternative resources

Consider again the product mix problem. Assume now that the resources are not concurrent, but they are **alternative resources** $(R1, \ldots, Rm)$. Namely, any given quantity of product Pi will consume only a proportional amount of one single resource Rj (instead of consuming all the resources as in the case of concurrent resources).

This decision problem is an LP:

$$
\begin{aligned}
\underset{q}{\text{maximize}} \quad & \sum_{i=1}^{n} p_i \sum_{j=1}^{m} q_{ij} \\
\text{subject to} \quad & \sum_{i=1}^{n} a_{ij} q_{ij} \leq b_j, \quad j = 1, \ldots, m \\
& l_i \leq \sum_{j=1}^{m} q_{ij} \leq u_i, \quad i = 1, \ldots, n \\
& 0 \leq q_{ij}, \quad i = 1, \ldots, n, \; j = 1, \ldots, m \\
& q \in \mathbb{R}^n \times \mathbb{R}^m.
\end{aligned}
$$

The GNU MathProg model file named `product_mix3.mod`:

```
set P;
set R;

param p{P};
param a{P,R};
param b{R};
param l{P};
param u{P};

var q{i in P, j in R} >= 0;

maximize utility:
  sum{i in P, j in R} p[i]*q[i,j];

subject to alternative_resources {j in R}:
  sum{i in P} a[i,j]*q[i,j] <= b[j];

subject to box {i in P}:
  l[i] <= sum{j in R} q[i,j] <= u[i];

end;
```

Now, we run `glpsol` in our console:

```
glpsol -m product_mix3.mod -d product_mix1.dat -o output.txt
```

We get the output file named `output.txt`:

```
Problem:    product_mix3
Rows:       7
Columns:    9
Non-zeros:  27
Status:     OPTIMAL
Objective:  utility = 110705.8824 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 utility      B         110706
     2 alternative_resources[R1]
                    NU           480                         480            50
     3 alternative_resources[R2]
                    NU           480                         480       43.1373
     4 alternative_resources[R3]
                    NU           300                         300           220
     5 box[P1]      B             24             0           100
     6 box[P2]      B              0             0           100
     7 box[P3]      B        39.4118             0           100

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 q[P1,R1]     B             24             0
     2 q[P1,R2]     NL             0             0                  -337.255
     3 q[P1,R3]     NL             0             0                     -2520
     4 q[P2,R1]     NL             0             0                     < eps
     5 q[P2,R2]     NL             0             0                  -311.765
     6 q[P2,R3]     NL             0             0                    -16320
     7 q[P3,R1]     NL             0             0                      -900
     8 q[P3,R2]     B        9.41176             0
     9 q[P3,R3]     B             30             0

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 5.68e-14 on row 4
        max.rel.err = 9.46e-17 on row 4
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality
```

```
KKT.DE: max.abs.err = 4.55e-13 on column 8
        max.rel.err = 1.03e-16 on column 8
        High quality

KKT.DB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

The computed optimal value is 110705.8824.

Exercise: Modify the model by adding the same percentage constraints of Unit 1, and solve it with GLPK.
[Produce the model file `product_mix4.mod`.]

3 Blending

Blending or mixing ingredients to obtain a product with certain characteristics or properties is a decision problem that can be modeled as an LP. The problem consists in determining the optimum amounts of some ingredients $(I1, \ldots, In)$ to include in a mix. The final product must satisfy some **quality** restrictions on its quantitative characteristics $(Q1, \ldots, Qm)$ like amount of chemical elements or nutritive contents. Any unit of ingredient Ii has a certain cost c_i and a certain amount a_{ij} of quantitative characteristic Qj, for $j = 1, \ldots, m$.

Thus, the quantity q_i of any ingredient Ii must be decided in order to guarantee that any quantitative characteristic Qj is in the range $[d_j, b_j]$. Any quantity q_i must be in the box $[l_i, u_i]$, with $l_i \geq 0$.

$$\begin{aligned} \underset{q}{\text{minimize}} \quad & \sum_{i=1}^{n} c_i q_i \\ \text{subject to} \quad & d_j \leq \sum_{i=1}^{n} a_{ij} q_i \leq b_j, \quad j = 1, \ldots, m \\ & l_i \leq q_i \leq u_i, \quad i = 1, \ldots, n \\ & q \in \mathbb{R}^n. \end{aligned}$$

The GNU MathProg model file `blending1.mod`:

```
set I;
set Q;
```

```
param c{I};
param a{I,Q};
param d{Q};
param b{Q};
param l{I};
param u{I};

var q{i in I} >= l[i], <= u[i];

minimize cost:
  sum{i in I} c[i]*q[i];

subject to quality {j in Q}:
  d[j] <= sum{i in I} a[i,j]*q[i] <= b[j];

end;
```

and the data file `blending1.dat`:

```
set I := I1 I2 I3 I4;
set Q := Q1 Q2 Q3;

param: c l u :=
I1 2 0 6
I2 3 0 10
I3 4 0 6
I4 19 0 5;

param a: Q1 Q2 Q3 :=
I1 110 4 2
I2 160 8 285
I3 180 13 54
I4 260 14 80;

param: d b :=
Q1 2000 4000
Q2 50 100
Q3 700 1000;
```

constitute a particular instance of the blending problem that produce the following output:

```
Problem:    blending1
Rows:       4
Columns:    4
Non-zeros:  16
```

```
Status:     OPTIMAL
Objective:  cost = 73.79727461 (MINimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 cost         B        73.7973
     2 quality[Q1]  NL          2000          2000          4000      0.236669
     3 quality[Q2]  NU           100            50           100      -2.78592
     4 quality[Q3]  NU          1000           700          1000    -0.0441394

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------ ------------- ------------- -------------
     1 q[I1]        NU             6             0             6      -12.8017
     2 q[I2]        B        2.45127             0            10
     3 q[I3]        B        1.61884             0             6
     4 q[I4]        B        2.52464             0             5

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 3.41e-13 on row 4
        max.rel.err = 1.70e-16 on row 4
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err = 7.11e-15 on column 4
        max.rel.err = 5.73e-17 on column 4
        High quality

KKT.DB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

Exercise: Modify the model by adding some **gradation constraints**. Specifically, add to the model a characteristic $Q4$ that is quantified as a percentage like alcohol gradation, and add these constraints:

- the gradation of $Q4$ in the blend cannot be less than 5%
- the gradation of $Q4$ in the blend cannot be more than 10%.

Data: gradations of the ingredients $I1$: 8%, $I2$: 28%, $I3$: 33%, $I4$: 5%. Solve it with GLPK.

[Produce the model file `blending2.mod` and the data file `blending2.dat`.]