Programmes Linéaires

Initialisation de l'algorithme du simplex

Bashar Dudin

March 31, 2019

EPITA

Où on en est, à quoi on fait face?

On a un algorithme du SIMPLEXE (*restreint*), qu'on prétend résoudre les programmes linéaires. Pour l'instant on a fait de montrer que l'algorithme en question se termine, que cela soit en retournant une valeur finie ou en renvoyant l'information *non-borné*.

Jusqu'à présent, on a travailler sous l'hypothèse:

- nos programmes linéaire sont ***admissible***, c'est-à-dire que le lieu admissible n'est pas vide ;
- la solution de base de programme linéaire initial est admissible.

Où on en est, à quoi on fait face?

On a un algorithme du SIMPLEXE (*restreint*), qu'on prétend résoudre les programmes linéaires. Pour l'instant on a fait de montrer que l'algorithme en question se termine, que cela soit en retournant une valeur finie ou en renvoyant l'information *non-borné*.

Jusqu'à présent, on a travailler sous l'hypothèse:

- nos programmes linéaire sont ***admissible***, c'est-à-dire que le lieu admissible n'est pas vide ;
- la solution de base de programme linéaire initial est admissible.

Dans la suite, on va construire une fonction `init_simplex` qui ayant un entrée $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ retourne ou bien le fait que le programme linéaire correspondant est ***non***-admissible ou alors un programme linéaire $(N, B, \underline{\boldsymbol{A}}, \underline{\boldsymbol{b}}, \underline{\boldsymbol{c}}, v)$ sous forme *slack* qui a une solution de base admissible.

Admissibilité

Décider de l’admissibilité

Soit L le programme linéaire sous forme standard:

$$
\begin{array}{lc}
\text{maximiser} & z = v + \sum_{j=1}^{n} c_j x_j \\
\text{sujet á} & \\
& \forall i \in \{1, \ldots, m\}, \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \\
\text{avec} & \forall j \in N, \quad x_j \geq 0
\end{array}
$$

Décider de l'admissibilité

Soit L le programme linéaire sous forme standard:

$$\begin{aligned}
&\text{maximiser} && z = v + \sum_{j=1}^{n} c_j x_j \\
&\text{sujet á} && \\
& && \forall i \in \{1, \ldots, m\}, \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \\
&\text{avec} && \forall j \in N, \quad x_j \geq 0
\end{aligned}$$

À partir de L on construit le programme linéaire auxiliaire L_m[a]:

$$\begin{aligned}
&\text{maximiser} && z = -x_0 \\
&\text{sujet á} && \\
& && \forall i \in \{1, \ldots, m\}, \quad \sum_{j=1}^{n} a_{ij} x_j - x_0 \leq b_i \\
&\text{avec} && \forall j \in N \cup \{0\}, \quad x_j \geq 0
\end{aligned}$$

[a] Quelle en est la signification?

Décider de l'admissibilité

Proposition

L est admissible si et seulement si la valeur optimal de L_m est 0. Dans ce cas le point optimal associé est un point extrémal du lieu de admissible de L.

Décider de l'admissibilité

Proposition

L est admissible si et seulement si la valeur optimal de L_m est 0. Dans ce cas le point optimal associé est un point extrémal du lieu de admissible de L.

Preuve : On note en un premier temps que la valeur objectif de L_m est majorée par 0. Si on trouve que 0 est une valeur objectif de L_m elle est nécessairement optimale.

Décider de l'admissibilité

Proposition

L est admissible si et seulement si la valeur optimal de L_m est 0. Dans ce cas le point optimal associé est un point extrémal du lieu de admissible de L.

Preuve : On note en un premier temps que la valeur objectif de L_m est majorée par 0. Si on trouve que 0 est une valeur objectif de L_m elle est nécessairement optimale. Si L est admissible alors il existe un tuple $(t_1, \ldots, t_n)$ de nombre positifs qui satisfont les contraintes linéaires de L. Le tuple $(0, t_1, \ldots, t_n)$ satisfait donc celles de L_m et a pour valeur objectif 0.

Décider de l'admissibilité

Proposition

L est admissible si et seulement si la valeur optimal de L_m est 0. Dans ce cas le point optimal associé est un point extrémal du lieu de admissible de L.

Preuve : On note en un premier temps que la valeur objectif de L_m est majorée par 0. Si on trouve que 0 est une valeur objectif de L_m elle est nécessairement optimale. Si L est admissible alors il existe un tuple $(t_1, \ldots, t_n)$ de nombre positifs qui satisfont les contraintes linéaires de L. Le tuple $(0, t_1, \ldots, t_n)$ satisfait donc celles de L_m et a pour valeur objectif 0.

Inversement, si L_m a 0 pour valeur objectif (donc optimale) le point optimal associé est de la forme $(0, t_1, \ldots, t_n)$. En réinjectant ce tuple dans les contraintes de L_m on trouve que $(t_1, \ldots, t_n)$ satisfait les contraintes de L. ∎

Décider de l'admissibilité

Proposition

L est admissible si et seulement si la valeur optimal de L_m est 0. Dans ce cas le point optimal associé est un point extrémal du lieu de admissible de L.

La différence entre L et L_m réside dans le fait que L_m est toujours admissible.

Si b_{min} est le plus petit b_i négatif pour $i \in B$, le tuple $(-b_{min}, 0, \ldots, 0)$ est une solution admissible de L_m.

En admettant (temporairement) la validité de l'algorithme du simplexe *restreint*, si l'on trouve un programme équivalent à L_m qui a une solution de base admissible on peut donc décider de l'admissibilité de L.

Décider de l'admissibilité

On considère la forme *slack* de L_m

$$
\begin{array}{ll}
\text{maximiser} & z = -x_0 \\
\text{sujet á} & \\
& \forall i \quad x_{i+m} + \sum_{j=1}^{n} a_{ij} x_j - x_0 = b_i \\
\text{avec} & \forall j \in N \cup B \cup \{0\}, \quad x_j \geq 0
\end{array}
$$

Décider de l'admissibilité

On considère la forme *slack* de L_m

$$\begin{array}{ll} \text{maximiser} & z = -x_0 \\ \text{sujet á} & \\ & \forall i \quad x_{i+m} + \sum_{j=1}^{n} a_{ij} x_j - x_0 = b_i \\ \text{avec} & \forall j \in N \cup B \cup \{0\}, \quad x_j \geq 0 \end{array}$$

La solution de base de L_m n'est pas admissible dès que c'est le case de L_m ; un des b_i est strictement négatif. **On suppose que c'est le cas**.

Décider de l'admissibilité

On considère la forme *slack* de L_m

$$\begin{array}{ll} \text{maximiser} & z = -x_0 \\ \text{sujet á} & \\ & \forall i \quad x_{i+m} + \sum_{j=1}^{n} a_{ij} x_j - x_0 = b_i \\ \text{avec} & \forall j \in N \cup B \cup \{0\}, \quad x_j \geq 0 \end{array}$$

La solution de base de L_m n'est pas admissible dès que c'est le case de L_m ; un des b_i est strictement négatif. **On suppose que c'est le cas**.

Soit b_{min} le plus petit des b_i pour $i \in B$. On sait déjà que $(-b_{min}, 0, \ldots, 0, \boldsymbol{b} - b_{min})$ est une solution admissible de L_m.

Décider de l'admissibilité

On considère la forme *slack* de L_m

$$\begin{array}{ll} \text{maximiser} & z = -x_0 \\ \text{sujet á} & \\ & \forall i \quad x_{i+m} + \sum_{j=1}^{n} a_{ij} x_j - x_0 = b_i \\ \text{avec} & \forall j \in N \cup B \cup \{0\}, \quad x_j \geq 0 \end{array}$$

La solution de base de L_m n'est pas admissible dès que c'est le case de L_m ; un des b_i est strictement négatif. **On suppose que c'est le cas**.

Soit b_{min} le plus petit des b_i pour $i \in B$. On sait déjà que $(-b_{min}, 0, \ldots, 0, \boldsymbol{b} - b_{min})$ est une solution admissible de L_m.

Il suffit désormais d'utiliser `pivot` avec variable entrante 0 et sortante *min*. La **même** solution admissible de L_m est maintenant une solution **de base** admissible.

Décider de l'admissibilité

On considère la forme *slack* de L_m

$$\begin{array}{lr} \text{maximiser} & z = -x_0 \\ \text{sujet á} & \\ & \forall i \quad x_{i+m} + \sum_{j=1}^{n} a_{ij}x_j - x_0 = b_i \\ \text{avec} & \forall j \in N \cup B \cup \{0\}, \quad x_j \geq 0 \end{array}$$

La solution de base de L_m n'est pas admissible dès que c'est le case de L_m ; un des b_i est strictement négatif. **On suppose que c'est le cas**.

Soit b_{min} le plus petit des b_i pour $i \in B$. On sait déjà que $(-b_{min}, 0, \ldots, 0, \boldsymbol{b} - b_{min})$ est une solution admissible de L_m.

Il suffit désormais d'utiliser `pivot` avec variable entrante 0 et sortante *min*. La **même** solution admissible de L_m est maintenant une solution **de base** admissible.

On vient d'obtenir un programme linéaire équivalent à L_m ayant une solution de base admissible !

Tableau du programme auxiliaire

Le tableau `T_m` de L_m est obtenu à partir de L (`T`) en ajoutant la colonne $(-1, 1, \ldots, 1)^T$ avant la colonne $\boldsymbol{b}^T$ et en mettant tous les autres coefficients de la première ligne à 0.

	x_1	$\cdots$	x_n	x_{n+1}	x_{n+2}	$\cdots$	x_{n+m}	x_0	
	0	$\cdots$	0	0	0	$\cdots$	0	1	0
$n+1$	a_{11}	$\cdots$	a_{1n}	1	0	$\cdots$	0	-1	b_1
$n+2$	a_{21}	$\cdots$	a_{2n}	0	1	$\cdots$	0	-1	b_2
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$n+m$	a_{m1}	$\cdots$	a_{mn}	0	$\cdots$	$\cdots$	1	-1	b_m

Testing Feasibility

```python
def is_feasible(N, B, T):
"""Testing feasibility of linear program.

Args:
  N, B (list[int]): lists of non-basic and
  basic variables.
  T (ndarray[float]): numpy array for
  tableau of linear program.
Output:
  (bool) True if program is feasible
  False otherwise.
"""
p, q = T.shape[0], T.shape[1]
c = T[0, :]
T[0, :] = [0]*q
new_c = np.array([1] + [-1]*(p-1), dtype=float)
T = np.insert(T, -1, new_c, axis=1)
N.append(0)
# pivoting to be basic feasible
i_min = np.argmin(T[1: ,-1])
pivot(N, B, T, i_min, 0)
V = _simplex(N, B, T)
return math.isclose(V[0], 0)
# Possibly slightly modifying the
# optimisation problem ...
```

Testing Feasibility

```python
def is_feasible(N, B, T):
    """Testing feasibility of linear program.

    Args:
      N, B (list[int]): lists of non-basic and
      basic variables.
      T (ndarray[float]): numpy array for
      tableau of linear program.
    Output:
      (bool) True if program is feasible
      False otherwise.
    """
    p, q = T.shape[0], T.shape[1]
    c = T[0, :]
    T[0, :] = [0]*q
    new_c = np.array([1] + [-1]*(p-1), dtype=float)
    T = np.insert(T, -1, new_c, axis=1)
    N.append(0)
    # pivoting to be basic feasible
    i_min = np.argmin(T[1: ,-1])
    pivot(N, B, T, i_min, 0)
    V = _simplex(N, B, T)
    return math.isclose(V[0], 0)
    # Possibly slightly modifying the
    # optimisation problem ...
```

Pas encore de programme équivalent à L ici. Il faut étudier l'effet de bord.

Deuxième phase: retomber sur ses pieds

Retrouver le programme de départ

Hypothèse

On suppose `is_feasible(N, B, T)` renvoie `True`.

Retrouver le programme de départ

Hypothèse

On suppose `is_feasible(N, B, T)` renvoie `True`.

Sous l'hypothèse précédente `is_feasible` transforme L_m en un programme linéaire P équivalent qui a une solution de base admissible ayant pour valeur objectif 0. Le point optimal associé est un point extrémal du lieu admissible de L. Pour retrouver L:

Retrouver le programme de départ

Hypothèse

On suppose `is_feasible(N, B, T)` renvoie `True`.

- remplacer la fonction objectif de P par la fonction objectif originale L ;

Retrouver le programme de départ

Hypothèse

On suppose `is_feasible(N, B, T)` renvoie `True`.

- remplacer la fonction objectif de P par la fonction objectif originale L ;
- en faire un programme *slack* Q en remplaçant les variables de base dans la fonction objectif par leurs expressions en fonction des variables hors base de P ;

Retrouver le programme de départ

Hypothèse

On suppose `is_feasible(N, B, T)` renvoie `True`.

- remplacer la fonction objectif de P par la fonction objectif originale L ;
- en faire un programme *slack* Q en remplaçant les variables de base dans la fonction objectif par leurs expressions en fonction des variables hors base de P ;
- s'assurer que x_0 n'est plus une variable de base de Q en pivotant éventuellement avec une variable sortante n'ayant pas de coefficient nul.

Retrouver le programme de départ

Hypothèse

On suppose `is_feasible(N, B, T)` renvoie `True`.

- remplacer la fonction objectif de P par la fonction objectif originale L ;
- en faire un programme *slack* Q en remplaçant les variables de base dans la fonction objectif par leurs expressions en fonction des variables hors base de P ;
- s'assurer que x_0 n'est plus une variable de base de Q en pivotant éventuellement avec une variable sortante n'ayant pas de coefficient nul.

Fait

En posant $x_0 = 0$ dans la formulation de Q on retrouve un programme linéaire équivalent à L et ayant une solution de base admissible.

The `is_feasible` function

```python
def is_feasible(N, B, T):
  """Testing feasibility of linear program.

  Args:
    N, B (list[int]): lists of non-basic and
    basic variables.
    T (ndarray[float]): numpy array for
    tableau of linear program.
  Output:
    (bool) True if program is feasible
    False otherwise.
  Boundary effect:
    Transforms T into an equivalent LP having
    basic feasible solution if T is feasible.
  """
  p, q = T.shape[0], T.shape[1]
  c = T[0, :]
  T[0, :] = [0]*q
  new_c = np.array([1] + [-1]*(p-1), dtype=float)
  T = np.insert(T, -1, new_c, axis=1)
  N.append(0)
  # pivoting to be basic feasible
  i_min = np.argmin(T[1: ,-1])
  pivot(N, B, T, i_min, 0)
  V = _simplex(N, B, T)
  if 0 in B:
    l, e = B.index(0), 0
    while T[l, e] == 0 or e == l:
      e += 1
    pivot(N, B, T, l, e)
  for i in B:
    c -= c[i]*T[B.index(i), :]
  T[0, :] = c
  return math.isclose(V[0], 0)
  # Possibly slightly modifying the
  # optimisation problem ...
```

Où on en est?

On est désormais en mesure de

- savoir si un programme linéaire L est admissible via `is_feasible`

[1] On ne le sait pas encore.

Où on en est?

On est désormais en mesure de

- savoir si un programme linéaire L est admissible via `is_feasible`
- si L est admissible on peut extraire un programme équivalent par effet de bord de `is_feasible`

[1] On ne le sait pas encore.

Où on en est?

On est désormais en mesure de

- savoir si un programme linéaire L est admissible via `is_feasible`
- si L est admissible on peut extraire un programme équivalent par effet de bord de `is_feasible`
- une fois qu'on a un programme équivalent à L et ayant une solution de base admissible, on peut utiliser `_simplex` (*restreint*) pour *résoudre*[1] L.

[1] On ne le sait pas encore.

Où on en est?

On est désormais en mesure de

- savoir si un programme linéaire L est admissible via `is_feasible`
- si L est admissible on peut extraire un programme équivalent par effet de bord de `is_feasible`
- une fois qu'on a un programme équivalent à L et ayant une solution de base admissible, on peut utiliser `_simplex` (*restreint*) pour *résoudre*[1] L.

On appelle `simplex` (à deux phases) l'algorithme qui reproduit les étapes précédentes.

[1] On ne le sait pas encore.

That’s all for today !