Linear Programs

The Simplex Algorithm II

Bashar Dudin

March 19, 2019

EPITA

Où on en est, à quoi on fait face.

La procédure adoptée à ce stade s'est toujours terminée. Notre conjecture de base est qu'elle nous renvoie un point optimal du programme linéaire de départ. Pour l'instant on ne peut pas en être certains. Par bien des égards ce qu'on a fait jusqu'à présent est loin d'être satisfaisant. Étant donné un programme linéaire L, voici une liste de points qu'ils nous restent à éclaircir:

- Comment savoir si L est admissible?

Où on en est, à quoi on fait face.

La procédure adoptée à ce stade s'est toujours terminée. Notre conjecture de base est qu'elle nous renvoie un point optimal du programme linéaire de départ. Pour l'instant on ne peut pas en être certains. Par bien des égards ce qu'on a fait jusqu'à présent est loin d'être satisfaisant. Étant donné un programme linéaire L, voici une liste de points qu'ils nous restent à éclaircir:

- Comment savoir si L est admissible?
- Que faire lorsque la solution de base n'est pas admissible?

Où on en est, à quoi on fait face.

La procédure adoptée à ce stade s'est toujours terminée. Notre conjecture de base est qu'elle nous renvoie un point optimal du programme linéaire de départ. Pour l'instant on ne peut pas en être certains. Par bien des égards ce qu'on a fait jusqu'à présent est loin d'être satisfaisant. Étant donné un programme linéaire L, voici une liste de points qu'ils nous restent à éclaircir:

- Comment savoir si L est admissible?
- Que faire lorsque la solution de base n'est pas admissible?
- Comment vérifier que L n'est pas majoré?

Où on en est, à quoi on fait face.

La procédure adoptée à ce stade s'est toujours terminée. Notre conjecture de base est qu'elle nous renvoie un point optimal du programme linéaire de départ. Pour l'instant on ne peut pas en être certains. Par bien des égards ce qu'on a fait jusqu'à présent est loin d'être satisfaisant. Étant donné un programme linéaire L, voici une liste de points qu'ils nous restent à éclaircir:

- Comment savoir si L est admissible?
- Que faire lorsque la solution de base n'est pas admissible?
- Comment vérifier que L n'est pas majoré?
- Est-ce que la procédure qu'on a pu tester jusqu'à présent se termine en général?

Où on en est, à quoi on fait face.

La procédure adoptée à ce stade s'est toujours terminée. Notre conjecture de base est qu'elle nous renvoie un point optimal du programme linéaire de départ. Pour l'instant on ne peut pas en être certains. Par bien des égards ce qu'on a fait jusqu'à présent est loin d'être satisfaisant. Étant donné un programme linéaire L, voici une liste de points qu'ils nous restent à éclaircir:

- Comment savoir si L est admissible?
- Que faire lorsque la solution de base n'est pas admissible?
- Comment vérifier que L n'est pas majoré?
- Est-ce que la procédure qu'on a pu tester jusqu'à présent se termine en général?
- Si la procédure se termine, est-ce qu'on en déduit toujours une valeur optimale?

Où on en est, à quoi on fait face.

La procédure adoptée à ce stade s'est toujours terminée. Notre conjecture de base est qu'elle nous renvoie un point optimal du programme linéaire de départ. Pour l'instant on ne peut pas en être certains. Par bien des égards ce qu'on a fait jusqu'à présent est loin d'être satisfaisant. Étant donné un programme linéaire L, voici une liste de points qu'ils nous restent à éclaircir:

- Comment savoir si L est admissible?
- Que faire lorsque la solution de base n'est pas admissible?
- Comment vérifier que L n'est pas majoré?
- Est-ce que la procédure qu'on a pu tester jusqu'à présent se termine en général?
- Si la procédure se termine, est-ce qu'on en déduit toujours une valeur optimale?

Pivoter

Pivoter | Fixer la notation

Dans le but de répondre aux questions qu'ils nous restent à aborder il va nous falloir écrire au propre les algorithmes qu'on a testés à la main. On considère le programme linéaire L

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

On écrit

- $\boldsymbol{A}$ pour la matrice de taille (m, n) des coefficients $\left(a_{ij}\right)_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}}$
- $\boldsymbol{b}$ pour le m-tuple $(b_1, \ldots, b_m)$
- $\boldsymbol{c}$ pour le n-tuple $(c_1, \ldots, c_n)$

Pivoter | Fixer la notation

Dans le but de répondre aux questions qu'ils nous restent à aborder il va nous falloir écrire au propre les algorithmes qu'on a testés à la main. On considère le programme linéaire L

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

Le programme linéaire L dans cette forme est décrit par la donnée $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$. Le programme initial a en général $v = 0$.

Pivoter | Fixer la notation

Dans le but de répondre aux questions qu'ils nous restent à aborder il va nous falloir écrire au propre les algorithmes qu'on a testés à la main. On considère le programme linéaire L

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

Pour encoder la forme slack on inclus également les données N, B des indices de variables hors-base et de base. Une forme slack est donc donnée par $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$. L'ensemble N est initialisé à $\{1, \dots, n\}$ et B à $\{n+1, \dots, n+m\}$.

Pivoter | Fixer la notation

Dans le but de répondre aux questions qu'ils nous restent à aborder il va nous falloir écrire au propre les algorithmes qu'on a testés à la main. On considère le programme linéaire L

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

Pour encoder la forme slack on inclus également les données N, B des indices de variables hors-base et de base. Une forme slack est donc donnée par $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$. L'ensemble N est initialisé à $\{1, \ldots, n\}$ et B à $\{n+1, \ldots, n+m\}$. Pour se rapprocher de l'écriture d'un système linéaire on modifie légèrement l'écriture de la forme slack.

Pivoter | Le tableau d'un programme linéaire

Les contraintes linéaires de L dans la forme slack représente une matrice $(m, n+m)$ notée $\underline{\boldsymbol{A}}$ et obtenue comme concaténation de $\boldsymbol{A}$ et $\boldsymbol{I}_m$ le long des lignes de $\boldsymbol{A}$. Le ***tableau*** T de L est obtenu en

Pivoter | Le tableau d'un programme linéaire

Les contraintes linéaires de L dans la forme slack représente une matrice $(m, n+m)$ notée $\underline{\boldsymbol{A}}$ et obtenue comme concaténation de $\boldsymbol{A}$ et $\boldsymbol{I}_m$ le long des lignes de $\boldsymbol{A}$. Le ***tableau*** T de L est obtenu en

- concaténant $\underline{\boldsymbol{A}}$ et b^T le long des lignes de A

Pivoter | Le tableau d'un programme linéaire

Les contraintes linéaires de L dans la forme slack représente une matrice $(m, n+m)$ notée $\underline{\boldsymbol{A}}$ et obtenue comme concaténation de $\boldsymbol{A}$ et $\boldsymbol{I}_m$ le long des lignes de $\boldsymbol{A}$. Le ***tableau*** T de L est obtenu en

- concaténant $\underline{\boldsymbol{A}}$ et b^T le long des lignes de A
- concaténant les résultats avec le vecteur $(-c_1, \ldots, -c_n, 0, \ldots 0, v)$ de taille $n+m+1$ le long des colonnes de $\boldsymbol{A}$.

Pivoter | Le tableau d'un programme linéaire

Les contraintes linéaires de L dans la forme slack représente une matrice $(m, n+m)$ notée $\underline{\boldsymbol{A}}$ et obtenue comme concaténation de $\boldsymbol{A}$ et $\boldsymbol{I}_m$ le long des lignes de $\boldsymbol{A}$. Le ***tableau*** T de L est obtenu en

- concaténant $\underline{\boldsymbol{A}}$ et b^T le long des lignes de A
- concaténant les résultats avec le vecteur $(-c_1, \ldots, -c_n, 0, \ldots 0, v)$ de taille $n+m+1$ le long des colonnes de $\boldsymbol{A}$.

	x_1	$\cdots$	x_n	x_{n+1}	x_{n+2}	$\cdots$	x_{n+m}	
	$-c_1$	$\cdots$	$-c_n$	0	0	$\cdots$	0	v
$n+1$	a_{11}	$\cdots$	a_{1n}	1	0	$\cdots$	0	b_1
$n+2$	a_{21}	$\cdots$	a_{2n}	0	1	$\cdots$	0	b_2
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$n+m$	a_{m1}	$\cdots$	a_{mn}	0	$\cdots$	$\cdots$	1	b_m

Pivoter

We are given input $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ and two indexes $e \in N$, $l \in B$ respectively corresponding to entering and leaving variables to and from the set of *basic* variables B.

Pivoter

We are given input $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ and two indexes $e \in N$, $l \in B$ respectively corresponding to entering and leaving variables to and from the set of *basic* variables B.

- Express x_e in terms of other variables in equation l

```
T[l, :] = (1/T[l, e])*T[l, :]
```

Pivoter

We are given input $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ and two indexes $e \in N$, $l \in B$ respectively corresponding to entering and leaving variables to and from the set of *basic* variables B.

- Express x_e in terms of other variables in equation l
- Replace x_e by previously obtained expression in linear constraints

```
T[l, :] = (1/T[l, e])*T[l, :]
```

```
if i != l:
  T[i, :] -= T[i, e]*T[l, :]
```

Pivoter

We are given input $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ and two indexes $e \in N$, $l \in B$ respectively corresponding to entering and leaving variables to and from the set of *basic* variables B.

- Express x_e in terms of other variables in equation l
- Replace x_e by previously obtained expression in linear constraints
- Replace x_e by corresponding expression in the value function

```
T[l, :] = (1/T[l, e])*T[l, :]
```

```
if i != l:
  T[i, :] -= T[i, e]*T[l, :]
```

Pivoter

We are given input $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ and two indexes $e \in N$, $l \in B$ respectively corresponding to entering and leaving variables to and from the set of *basic* variables B.

- Express x_e in terms of other variables in equation l
- Replace x_e by previously obtained expression in linear constraints
- Replace x_e by corresponding expression in the value function
- Update basic and none basic sets of variables.

```
T[l, :] = (1/T[l, e])*T[l, :]
```

```
if i != l:
  T[i, :] -= T[i, e]*T[l, :]
```

```
N.insert(N.index(e), l).remove(e)
B.insert(B.index(l), e).remove(l)
```

Pivoter | Full Function

```python
def pivot(N, B, T, e, l):
    """Pivoting in linear programs.

    Pivots entering and leaving variables in linear
    program given as tableau. Done in place.
    """
    T[l, :] = (1/T[l, e])*T[l, :]
    for i in range(m+1):
        if i != l:  # ugly
            T[i, :] -= T[i, e]*T[i, :]
    N.insert(N.index(e), l).remove(e)
    B.remove(B.index(l), e).remove(l)
```

Pivoter | Full Function

- Index e is an element of N and l one of B.

Pivoter | Full Function

- Index e is an element of N and l one of B.
- There better be no entries e, l such that `T[l, e] == 0`. We shall ensure this is never the case when `pivot` is used.

Pivoter | Full Function

- Index e is an element of N and l one of B.
- There better be no entries e, l such that `T[l, e] == 0`. We shall ensure this is never the case when `pivot` is used.
- The objective value and basic solution of obtained linear program can be read on last column to the right.

Détecter le fait de ne pas être borné

Tester la non *bornitude*

On note L le programme linéaire en forme standard

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

Fait

S'il y a un indice j tel que $c_j > 0$ et tous les coefficients de x_j dans les contraintes linéaires sont négatifs alors L est non majoré.

Tester la non *bornitude*

On note L le programme linéaire en forme standard

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

Fait

De manière équivalente, s'il y a une colonne dans le tableau de L ayant une première entrée négative non nulle et toutes les autres négatives alors L est non majoré.

Tester la non *bornitude*

En utilisant le fait précédent, à chaque appel de `pivot`, on teste si on ne satisfait pas la propriété:

> *Il existe un indice j tel que* $c_j > 0$ *et tous les autres coefficients de* x_j *sont négatifs ou nuls.*

[1] Il l'est mais on a encore aucune certitude.

Tester la non *bornitude*

En utilisant le fait précédent, à chaque appel de `pivot`, on teste si on ne satisfait pas la propriété:

> *Il existe un indice* j *tel que* $c_j > 0$ *et tous les autres coefficients de* x_j *sont négatifs ou nuls.*

C'est une condition nécessaire, pour l'instant on a aucune garantie qu'on tombe dans une telle situation quand L n'est pas majoré. Le fait que ce soit le cas vient des résultats de dualité.

[1] Il l'est mais on a encore aucune certitude.

Tester la non *bornitude*

En utilisant le fait précédent, à chaque appel de `pivot`, on teste si on ne satisfait pas la propriété:

> *Il existe un indice j tel que* $c_j > 0$ *et tous les autres coefficients de* x_j *sont négatifs ou nuls.*

C'est une condition nécessaire, pour l'instant on a aucune garantie qu'on tombe dans une telle situation quand L n'est pas majoré. Le fait que ce soit le cas vient des résultats de dualité.

Pour l'instant on va devoir accepter que ce test naïf sera suffisant[1].

[1] Il l'est mais on a encore aucune certitude.

The Simplex Algorithm : Second Try

The Simplex Algorithm *Restricted*

```python
# Under construction function!
def _simplex(N, B, T):
    """Restricted simplex algorithm

    Runs simplex algorithm on basic feasible
    linear program in slack form.

    Args:
      N, B (list[int]): lists of non-basic
      and basic variables.
      T (ndarray[float]): numpy array for
      tableau of linear program.
    Output:
      (ndarray[float]) vector tail of which
      is maximal objective value, rest is
      optimal point.
    """
    m, margins = len(B), dict()
    aug_var = [i for i in N if T[0, i] < 0]
    while aug_var:
      e = random.choice(augmenting_var)
      for i in range(m):
        if T[i, e] > 0:
          margins[B[i]] = T[i, -1]/T[i, e]
      if not margins:
        raise Exception("Unbounded LP")
      min_margin = min(margins.values())
      minima = [i for i in margins\
                if margins[i] == min_margin]
      l = random.choice(minima)
      pivot(N, B, T, e, l)
      aug_var = [i for i in N if T[0, i] < 0]
    return T[:, -1]
```

Boucler

À chaque fois qu'on entre dans la boucle `while` de `_simplex` ou bien on *augmente la valeur objective* ou alors on découvre que le PL est non-majoré. A priori, `_simplex` pourrait s'exécuter indéfiniment, en itérant entre des formes slack équivalentes ayant une valeur objective constante. On va chercher à étudier ce phénomène.

Boucler

À chaque fois qu'on entre dans la boucle `while` de `_simplex` ou bien on *augmente la valeur objective* ou alors on découvre que le PL est non-majoré. A priori, `_simplex` pourrait s'exécuter indéfiniment, en itérant entre des formes slack équivalentes ayant une valeur objective constante. On va chercher à étudier ce phénomène.

Proposition C

Soit L un PL $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ où $\boldsymbol{A}$ est une matrice (m, n). Si la boucle de `_simplex` s'exécute plus de $\binom{n+m}{m}$ fois, alors l'algorithme boucle, i.e. la boucle s'exécute indéfiniment en alternant entre un nombre fini de formes slack ayant la même valeur objective.

Remarque : Cela signifie que dès que `_simplex` entre dans sa boucle principale plus $\binom{n+m}{m}$ fois alors on peut retourner la valeur objective et la solution de base courante.

Boucler

La preuve de la proposition ci-dessus se base sur deux faits:

- si on tombe sur une forme slack déjà vue au cours de la résolution des précédentes itérations, alors on retrouvera par la suite les mêmes itérations qui s'ensuivaient, on a un comportement périodique ;
- il n'y a qu'un nombre fini de formes slacks possibles pour un même programme linéaire.

Boucler

La preuve de la proposition ci-dessus se base sur deux faits:

- si on tombe sur une forme slack déjà vue au cours de la résolution des précédentes itérations, alors on retrouvera par la suite les mêmes itérations qui s'ensuivaient, on a un comportement périodique ;
- il n'y a qu'un nombre fini de formes slacks possibles pour un même programme linéaire.

Seul le second point nécessite d'être clarifié. On se contente de le montrer dans le cas de PL ayant un lieu admissible d'intérieur non vide.

Boucler

Lemme *B*

Soit L un PL donné par $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$. Une forme slack de L qui apparaît dans `_simplex` est déterminée par le choix d'un sous-ensemble B de variables de base.

Boucler

Lemme B

Soit L un PL donné par $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$. Une forme slack de L qui apparaît dans `_simplex` est déterminée par le choix d'un sous-ensemble B de variables de base.

Preuve : Deux formes slacks $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$
et $(N, B, \boldsymbol{A}', \boldsymbol{b}', \boldsymbol{c}', v')$ sont équivalentes.

Lemme *B*

Soit L un PL donné par $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$. Une forme slack de L qui apparaît dans `_simplex` est déterminée par le choix d'un sous-ensemble B de variables de base.

Preuve : Deux formes slacks $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ et $(N, B, \boldsymbol{A}', \boldsymbol{b}', \boldsymbol{c}', v')$ sont équivalentes.

En étudiant la différence des contraintes linéaires, on obtient les relations:

$$0 = (v - v') + \sum_{j \in N} (c_j - c'_j) x_j$$

et pour chaque $i \in B$

$$0 = (b_i - b'_i) - \sum_{j \in N} (a_{ij} - a'_{ij}) x_j.$$

Lemme B

Soit L un PL donné par $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$. Une forme slack de L qui apparaît dans `_simplex` est déterminée par le choix d'un sous-ensemble B de variables de base.

Preuve : Deux formes slacks $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ et $(N, B, \boldsymbol{A}', \boldsymbol{b}', \boldsymbol{c}', v')$ sont équivalentes.

En étudiant la différence des contraintes linéaires, on obtient les relations:

$$0 = (v - v') + \sum_{j \in N} (c_j - c_j') x_j$$

et pour chaque $i \in B$

$$0 = (b_i - b_i') - \sum_{j \in N} (a_{ij} - a_{ij}') x_j.$$

Ces relations étant vraie pour tout vecteur $(x_1, \ldots, x_n)$ dans le lieu admissible de LP, si le lieu admissible de LP est d'intérieur non vide, il est clair que pour tout $i \in B$ et $j \in N$ on a $v = v'$, $b_i = b_i'$, $c_j = c_j'$, $a_{ij} = a_{ij}'$. ∎

Boucler

Rajouter un compteur de boucle à l'algo du `_simplex` garanti la terminaison de `_simplex`.

Boucler

Rajouter un compteur de boucle à l'algo du `_simplex` garanti la terminaison de `_simplex`. **Remarque :** Cette solution n'est pas intelligente. Il y a plusieurs solutions à ce problème en pratique. Dans notre cas on implémente la règle de ***Bland***: à chaque choix d'indice aux lignes 21 et 30 de `_simplex`, on choisit le plus petit possible.

The Simplex Algorithm *Restricted* | No Cycling Version

```python
def _simplex(N, B, T):
    """Restricted simplex algorithm

    Runs simplex algorithm on basic feasible
    linear program in slack form.

    Args:
      N, B (list[int]): lists of non-basic
      and basic variables.
      T (ndarray[float]): numpy array for
      tableau of linear program.
    Output:
      (ndarray[float]) vector tail of which
      is maximal objective value, rest is
      optimal point.
    """
    m = len(B)
    l, margin = None, float('inf')
    aug_var = [i for i in N if T[0, i] < 0]
    while aug_var:
      e = min(augmenting_var)
      for i in range(m):
        if T[i, e] > 0:
           if T[i, -1]/T[i, e] < margin:
             margin = T[i, -1]/T[i,e]
             l = i
      if not l:
        raise Exception("Unbounded LP")
      pivot(N, B, T, e, l)
      aug_var = [i for i in N if T[0, i] < 0]
    return T[:, -1]
```

C’est tout pour aujourd’hui!