Linear Programs

Initializing The Simplex Algorithm

Bashar Dudin

November 20, 2018

EPITA

Where We Stand, What We Face?

We now have an algorithm SIMPLEX (*Restricted*), which is conjectured to solve linear programs. For the time being we've shown that it terminates either by specifying that we have an unbounded linear program, or by returning a finite value *expected* to be the maximum we are looking for.

Uptill now, we've been working under the following two assumptions :

- Our linear program is ***feasible*** ; meaning that it has at least one feasible solution
- The basic solution of the initial slack form is feasible.

Where We Stand, What We Face?

We now have an algorithm SIMPLEX (*Restricted*), which is conjectured to solve linear programs. For the time being we've shown that it terminates either by specifying that we have an unbounded linear program, or by returning a finite value *expected* to be the maximum we are looking for.

Uptill now, we've been working under the following two assumptions :

- Our linear program is ***feasible*** ; meaning that it has at least one feasible solution
- The basic solution of the initial slack form is feasible.

During this lecture, we are going to build a function `init_simplex` taking in a linear program $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$ and either returning back the fact it is ***not*** feasible or a linear program $(N, B, \underline{\boldsymbol{A}}, \underline{\boldsymbol{b}}, \underline{\boldsymbol{c}}, v)$ in slack form having feasible basic solution.

Feasibility

Deciding On Feasibility

Let L be the linear program, given in standard form:

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

Deciding On Feasibility

Let L be the linear program, given in standard form:

$$
\begin{array}{ll}
\text{maximize} & z = v + \sum_{j=1}^{n} c_j x_j \\
\text{subject to} & \\
& \forall i \in \{1, \dots, m\}, \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \\
\text{with} & \forall j \in N, \quad x_j \geq 0
\end{array}
$$

Out of L we build the following auxiliary linear program L_m:

$$
\begin{array}{ll}
\text{maximize} & z = -x_0 \\
\text{subject to} & \\
& \forall i \in \{1, \dots, m\}, \quad \sum_{j=1}^{n} a_{ij} x_j - x_0 \leq b_i \\
\text{with} & \forall j \in N \cup \{0\}, \quad x_j \geq 0
\end{array}
$$

Deciding On Feasibility

Proposition

L is feasible if, and only if, the optimal objective value of L_m is 0.

Deciding On Feasibility

Proposition

L is feasible if, and only if, the optimal objective value of L_m is 0.

Proof : Notice first that the optimal objective value of L_m is 0. Therefore, if we show that 0 is an objective value of L_m it is necessarily the optimal one.

Deciding On Feasibility

Proposition

L is feasible if, and only if, the optimal objective value of L_m is 0.

Proof : Notice first that the optimal objective value of L_m is 0. Therefore, if we show that 0 is an objective value of L_m it is necessarily the optimal one.
If L is feasible then there is a tuple $(t_1, \ldots, t_n)$ of non-negative real numbers satisfying all linear constraints of L. The tuple $(0, t_1, \ldots, t_n)$ does thus satisfy L_m and 0 is then an objective value of L_m, i.e. the optimal one.

Deciding On Feasibility

Proposition

L is feasible if, and only if, the optimal objective value of L_m is 0.

Proof : Notice first that the optimal objective value of L_m is 0. Therefore, if we show that 0 is an objective value of L_m it is necessarily the optimal one.
If L is feasible then there is a tuple $(t_1, \ldots, t_n)$ of non-negative real numbers satisfying all linear constraints of L. The tuple $(0, t_1, \ldots, t_n)$ does thus satisfy L_m and 0 is then an objective value of L_m, i.e. the optimal one.
Conversely, if L_m has objective value 0 (thus optimal) it is of the form $(0, t_1, \ldots, t_n)$. Plugging this tuple in the linear constraints of L_m it implies $(t_1, \ldots, t_n)$ is a solution of L. ■

Deciding On Feasibility

Proposition

L is feasible if, and only if, the optimal objective value of L_m is 0.

The difference between L and L_m is that L_m is always feasible :

If b_{min} is the smallest negative b_i for $i \in B$, the tuple $(-b_{min}, 0, \ldots, 0)$ is a feasible solution of L_m.

Thus, temporarily admitting validity of the *restricted* simplex algorithm, if we're able to find a linear program equivalent to L_m which has feasible basic solution then we can decide on the feasibilty of L.

Deciding On Feasibility

Consider the slack form of L_m

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

Deciding On Feasibility

Consider the slack form of L_m

$$\begin{aligned} &\text{maximize} && z = -x_0 \\ &\text{subject to} && \\ & && \forall i \quad x_{i+m} + \sum_{j=1}^{n} a_{ij} x_j - x_0 = b_i \\ &\text{with} && \forall j \in N \cup B \cup \{0\}, \quad x_j \geq 0 \end{aligned}$$

Basic solution of L_m is not feasible as soon as L is not, i.e. as soon as a b_i is negative.
We assume this is the case.

Deciding On Feasibility

Consider the slack form of L_m

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

Basic solution of L_m is not feasible as soon as L is not, i.e. as soon as a b_i is negative. **We assume this is the case**.

Let b_{min} be the minimal b_i coefficient. We already know that $(-b_{min}, 0, \ldots, 0, \boldsymbol{b} - b_{min})$ is a feasible solution of L_m.

Deciding On Feasibility

Consider the slack form of L_m

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

Basic solution of L_m is not feasible as soon as L is not, i.e. as soon as a b_i is negative. **We assume this is the case**.

Let b_{min} be the minimal b_i coefficient. We already know that $(-b_{min}, 0, \dots, 0, \boldsymbol{b} - b_{min})$ is a feasible solution of L_m.

Let's now use `pivot` with entering variable 0 and leaving one *min*. The **same** previous feasible solution of L_m is now the basic solution of the linear program we got after pivoting.

Deciding On Feasibility

Consider the slack form of L_m

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

Basic solution of L_m is not feasible as soon as L is not, i.e. as soon as a b_i is negative. **We assume this is the case**.

Let b_{min} be the minimal b_i coefficient. We already know that $(-b_{min}, 0, \ldots, 0, \boldsymbol{b} - b_{min})$ is a feasible solution of L_m.

Let's now use `pivot` with entering variable 0 and leaving one *min*. The **same** previous feasible solution of L_m is now the basic solution of the linear program we got after pivoting.

We got an equivalent linear program to L_m having feasible basic solution !

Tableau of Auxiliary Linear Program

The tableau `T_m` of L_m is obtained out of the one for L (`T`) by adding column $(1, -1, \ldots, -1)^T$ to tableau before column $\boldsymbol{b}^T$ and putting old coefficients of first row to 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

```
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 ...
```

Not an equivalent LP to L there! Need to understand boundary effects.

Get A Feasible Basic Solution

Getting a Feasible Basic Solution

Assumption

We assume `is_feasible(N, B, T)` returns `True`.

Getting a Feasible Basic Solution

Assumption

We assume `is_feasible(N, B, T)` returns `True`.

Under the previous assumption `is_feasible` transforms L_m into an equivalent linear program P which has feasible basic solution with objective value 0.

Getting a Feasible Basic Solution

Assumption

We assume `is_feasible(N, B, T)` returns `True`.

- Replace the objective value of P with the original one of L.

Getting a Feasible Basic Solution

Assumption

We assume `is_feasible(N, B, T)` returns `True`.

- Replace the objective value of P with the original one of L.
- Make of it a proper LP Q in slack form by replacing possible basic variables of the objective value with their expressions in terms of non-basic variables of P.

Getting a Feasible Basic Solution

Assumption

We assume `is_feasible(N, B, T)` returns `True`.

- Replace the objective value of P with the original one of L.
- Make of it a proper LP Q in slack form by replacing possible basic variables of the objective value with their expressions in terms of non-basic variables of P.
- Ensure x_0 is no more a basic variable Q by pivoting with leaving variable 0 and entering one any variable having non-zero coefficient.

Getting a Feasible Basic Solution

Assumption

We assume `is_feasible(N, B, T)` returns `True`.

- Replace the objective value of P with the original one of L.
- Make of it a proper LP Q in slack form by replacing possible basic variables of the objective value with their expressions in terms of non-basic variables of P.
- Ensure x_0 is no more a basic variable Q by pivoting with leaving variable 0 and entering one any variable having non-zero coefficient.

Fact

Putting x_0 to 0 in Q gives back an equivalent LP to L having feasible basic solution.

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 ...
```

Where We Stand?

We are now able to

- know whether a linear program is feasible or not using `is_feasible`

Where We Stand?

We are now able to

- know whether a linear program is feasible or not using `is_feasible`
- if it is feasible we can build up an equivalent linear program which has feasible basic solution using boundary effect of `is_feasible`

Where We Stand?

We are now able to

- know whether a linear program is feasible or not using `is_feasible`
- if it is feasible we can build up an equivalent linear program which has feasible basic solution using boundary effect of `is_feasible`
- once we get a linear program having feasible basic solution we can check whether our linear program is unbounded or has a *hopefully* optimal finite solution using the *restricted* `_simplex`.

Where We Stand?

We are now able to

- know whether a linear program is feasible or not using `is_feasible`
- if it is feasible we can build up an equivalent linear program which has feasible basic solution using boundary effect of `is_feasible`
- once we get a linear program having feasible basic solution we can check whether our linear program is unbounded or has a *hopefully* optimal finite solution using the *restricted* `_simplex`.

We are left with showing validity of *restricted* `_simplex`!

We call `simplex` the algorithm globally going through all previous steps.

That’s it for today !