# week04-02–dual-prices-foo

February 9, 2025

## 1 George McNinch Math 87 - Spring 2025

## 2 Week 4

## 3 Dual prices

## 4 Recollections

Let's briefly recall the notion of *slack variables* and *complementary slackness* from our duality discussion.

To this end, consider a linear program $\mathcal{L}$ given by $(\mathbf{c} \in \mathbb{R}^{1\times n}, A \in \mathbb{R}^{r\times n}, \mathbf{b} \in \mathbb{R}^r)$ which seeks to `maximize` its objective function. We write $\mathbf{0} \leq \mathbf{x} \in \mathbb{R}^n$ for the variable vector of our linear program, and we recall that it satisfies $A\mathbf{x} \leq \mathbf{b}$.

As usual we'll write $\mathcal{L}'$ for the dual linear program – it is determined by the triple $(\mathbf{b}^T, A^T, c^T)$; it seeks to `minimize` its objective function; the dual varible vector is written $\mathbf{y} \in \mathbf{R}^r$ and it satisfies $\mathbf{y} \geq \mathbf{0}$ and $A^T\mathbf{y} \geq \mathbf{c}^T$.

*Complementary slackness* is the assertion that for feasible points $\mathbf{x}$ for $\mathcal{L}$ and $\mathbf{y}$ for $\mathcal{L}'$, $\mathbf{x}$ is optimal for $\mathcal{L}$ and $\mathbf{y}$ is optimal for $\mathcal{L}'$ if and only if

$$(\clubsuit) \quad (\mathbf{b} - A\mathbf{x})^T \cdot \mathbf{y} = 0 \quad \text{and} \quad (\mathbf{y}^T A - \mathbf{c}) \cdot \mathbf{x} = 0.$$

For optimal vectors $\mathbf{x}^*$ and $\mathbf{y}^*$ we refer to the slack vectors:

$$(\mathbf{b} - A\mathbf{x}^*) \quad \text{and} \quad ((\mathbf{y}^*)^T A - \mathbf{c})$$

**Remark**: Recall that if $\mathbf{x}$ is a feasible point for $\mathcal{L}$, then $A\mathbf{x} \leq \mathbf{b}$ or put another way, the slack vector $\mathbf{b} - A\mathbf{x} \geq \mathbf{0}$ is non-negative. Now, if also $\mathbf{y} \geq \mathbf{0}$, it is easy to see that the product – a scalar quantity – satisfies

$$(\mathbf{b} - A\mathbf{x})^T \cdot \mathbf{y} \geq 0$$

and that in order to have $(\mathbf{b} - A\mathbf{x})^T \cdot \mathbf{y} = 0$ for a non-zero vector $\mathbf{y}$, some of the coefficients of $\mathbf{b} - A\mathbf{x}$ must be zero; in the discussion below we say that those coefficients - or the corresponding constraints – are *binding*.

## 5 Example

A company has acquired 100 lots on which to build homes of two styles: Cape Cod and Ranch. They will build these homes over a year, during which they will have available 13,000 hours of bricklayer labor and 12,000 hours of carpenter labor. Each Cape Cod house requires 200 hours of carpentry labor and 50 hours of bricklayer labor. Each Ranch house requires 120 hours of bricklayer labor and 100 hours of carpentry. The profit for building a Cape Cod home is projected to be \$5,100 and each Ranch home is projected to be \$5,000. How many of each type of house would you recommend building?

Variables:

- $C = \#$ Cape Cod homes built
- $R = \#$ Ranch homes built

Constraints:

- $l = \#$ lots
- $b =$ hours of bricklayer labor
- $c =$ hours of carpentry labor

Let's assemble the input for the linear program using some code:

[1]:
```
from scipy.optimize import linprog
import numpy as np

home_specs = { "cape": { "carpentry": 200.0,
                         "bricklayer": 50.0,
                         "profit": 5100.0,
                         "lots": 1
                       },
               "ranch": { "carpentry": 100.0,
                          "bricklayer": 120.0,
                          "profit": 5000.0,
                          "lots": 1
                        }
             }

resources_avail = { "carpentry": 12000,
                    "bricklayer": 13000,
                    "lots": 100
                  }
```

We are using python *dictionaries* to store the parameters. You don't *have* to do this, but it can be a nice organizing tool.

For example, the `bricklayer` requirements for `ranch` houses can be found as follows:

```
home_specs["ranch"]["bricklayer"]
```

And you can get a list of all `bricklayer` requirements using a loop:

```
htypes = ["cape", "ranch"]
```

```
for ht in htypes:
    print(home_specs[ht]["bricklayer"])
```

We could even make an `array` of all the `bricklayer` requirements using a so-called *list comprehension*:

```
np.array([ home_specs[t]["bricklayer"] for t in htypes ])
```

[2]:
```
home_specs["ranch"]["bricklayer"]
```

[2]:
```
120.0
```

[3]:
```
htypes = ["cape", "ranch"]
for ht in htypes:
    print(home_specs[ht]["bricklayer"])
```

```
50.0
120.0
```

[4]:
```
np.array([ home_specs[t]["bricklayer"] for t in htypes ])
```

[4]:
```
array([ 50., 120.])
```

[14]:
```
list(home_specs.values())
```

[14]:
```
[{'carpentry': 200.0, 'bricklayer': 50.0, 'profit': 5100.0, 'lots': 1},
 {'carpentry': 100.0, 'bricklayer': 120.0, 'profit': 5000.0, 'lots': 1}]
```

Now we can use some python code to create the matrices/vectors required for our linear program from the data stored in `home_specs` and `resources_avail`, as follows:

[6]:
```
# objective vector; entries correspond to the profit for each home_type.
c = np.array([ home_specs[t]["profit"] for t in home_specs.keys() ])

# constraint matrix; one row for each resource type.
A = np.array([ [ home_specs[t][r] for t in home_specs.keys() ]
                for r in resources_avail.keys() ])

b = np.array([ resources_avail[r] for r in resources_avail.keys() ])

print(f"objective function is given by \nc=\n{c}\n")
print(f"the inequality Ax <= b is given by \n\n A=\n{A} \n\n b=\n{b}")
```

```
objective function is given by
c=
[5100. 5000.]

the inequality Ax <= b is given by

 A=
[[200. 100.]
```

```
 [ 50. 120.]
 [  1.   1.]]

 b=
[12000 13000   100]
```

[16]:
```
np.array(resources_avail.values())
```

[16]:
```
array(dict_values([12000, 13000, 100]), dtype=object)
```

Primal linear program: *maximize* for the data $(\mathbf{c}, A, \mathbf{b})$.

i.e. the objective function is given by $\mathbf{c} \cdot \begin{bmatrix} C \\ R \end{bmatrix} = 5100C + 5000R$ where $\begin{bmatrix} C \\ R \end{bmatrix} \geq \mathbf{0}$.

And $A \cdot \begin{bmatrix} C \\ R \end{bmatrix} \leq \mathbf{b}$.

---

Thus the dual linear program is given by the data $(\mathbf{b}^T, A^T, \mathbf{c}^T)$.

Let's look first at the data determining the dual:

[7]:
```
print(f"objective function for the dual is given by \nc=\n{b}\n")
print(f"the inequality Ax >= b for the dual is given by \n\n A=\n{A.T} \n\n␣
  ↪b=\n{c}")
```

```
objective function for the dual is given by
c=
[12000 13000   100]

the inequality Ax >= b for the dual is given by

 A=
[[200.  50.   1.]
 [100. 120.   1.]]

 b=
[5100. 5000.]
```

We label the variables of the dual linear program using the two resource constraints: $\mathbf{y} = \begin{bmatrix} y_c \\ y_b \\ y_l \end{bmatrix}$ where $y_l$ denotes the unit price of a lot, $y_c$ denotes the unit price of carpentry labor, and $y_b$ denotes the unit price of bricklayer labor.

So the objective function for the dual system is given by

$$\mathbf{b}^T \cdot \begin{bmatrix} y_l \\ y_b \\ y_c \end{bmatrix} = 100y_l + 12000y_b + 13000y_c$$

and the inequality constraints are given by

$$A^T \cdot \begin{bmatrix} y_l \\ y_b \\ y_c \end{bmatrix} \geq \mathbf{c}^T = \begin{bmatrix} 5100 \\ 5000 \end{bmatrix}$$

[8]:
```
primal = linprog((-1)*c,A_ub = A,b_ub = b)

dual = linprog(b,A_ub = (-1)*A.T,b_ub = (-1)*c)

print("** primal:\n",primal,"\n\n----------\n\n")
print("** dual:\n",dual)
```

```
** primal:
         message: Optimization terminated successfully. (HiGHS Status 7:
Optimal)
        success: True
         status: 0
            fun: -502000.0
              x: [ 2.000e+01  8.000e+01]
            nit: 2
          lower:  residual: [ 2.000e+01  8.000e+01]
                 marginals: [ 0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf]
                 marginals: [ 0.000e+00  0.000e+00]
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 0.000e+00  2.400e+03  0.000e+00]
                 marginals: [-1.000e+00 -0.000e+00 -4.900e+03]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0

----------


** dual:
         message: Optimization terminated successfully. (HiGHS Status 7:
Optimal)
        success: True
         status: 0
            fun: 502000.0
              x: [ 1.000e+00  0.000e+00  4.900e+03]
            nit: 2
          lower:  residual: [ 1.000e+00  0.000e+00  4.900e+03]
                 marginals: [ 0.000e+00  2.400e+03  0.000e+00]
          upper:  residual: [       inf        inf        inf]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00]
```

```
          eqlin:  residual: []
                 marginals: []
        ineqlin:  residual: [ 0.000e+00  0.000e+00]
                 marginals: [-2.000e+01 -8.000e+01]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

So `scipy` confirms that an optimal solution to the primal linear system is

`primal.x = [ 2.000e+01  8.000e+01]`; i.e.

$$\mathbf{x}^* = \begin{bmatrix} C \\ R \end{bmatrix} = \begin{bmatrix} 20 \\ 80 \end{bmatrix}$$

Note that $\mathbf{c} \cdot \begin{bmatrix} 20 \\ 80 \end{bmatrix} = \$502,000$

And an optimal solution to the dual linear system is

`dual.x = [ 1.000e+00  0.000e+00  4.900e+03]`; i.e.

$$\mathbf{y}^* = \begin{bmatrix} y_c \\ y_b \\ y_l \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 4900 \end{bmatrix}$$

---

Let's try to understand what the *slack vectors* are telling us.

Let's compute

$$(\mathbf{b} - A\mathbf{x}^*) \quad \text{and} \quad ((\mathbf{y}^*)^T A - \mathbf{c})$$

[9]:
```
xstar = primal.x    # we get the vector from the .x member of the
                    # class returned by linprog

ystar = dual.x

slack_primal = b - A @ xstar
slack_dual = ystar @ A - c

# Note that you can actually get the slack vectors from the result given by␣
 ↪`linprog`
print("primal slack agrees?", slack_primal == primal.slack)
print("dual slack agrees?", slack_dual == dual.slack)
print(slack_primal)
print(slack_dual)
```

```
primal slack agrees? [ True  True  True]
dual slack agrees? [ True  True]
[   0. 2400.    0.]
[0. 0.]
```

We focus on `slack_primal = b - A @ xstar = [   0. 2400.    0.]`; i.e.

$$(\mathbf{b} - A\mathbf{x}^*) \approx \begin{bmatrix} 0 \\ 2400 \\ 0 \end{bmatrix}$$

First, this confirms (at least part of) the complementary slackness result; indeed,

$$(\mathbf{b} - A\mathbf{x}^*)^T \cdot \mathbf{y}^* = \begin{bmatrix} 0 & 2400 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 4900 \end{bmatrix} = 0.$$

[10]:
```
slack_primal @ ystar
```

[10]:
```
np.float64(0.0)
```

We can easily check the other assertion; i.e. `slack_dual @ xstar == 0`.

(But in this case, this is an easy assertion as `slack_dual` is the zero vector.)

In general, *one says that the constraints – or the dual variables – corresponding to zero entries of the slack vector are binding.*

In this case, the first and second entries of the slack variable $\mathbf{b} - A\mathbf{x}^*$ are zero, and hence the "lots" and "carpentry" constraints are binding; the resulting dual prices are 4900 for lots and 1 for carpentry. But we have an oversupply of available bricklaying, and thus the dual price for bricklaying is 0.

As some heuristic evidence for why we see the result we do, note that in our model, Cape Cod houses are more profitable, but require more carpentry.

## 6 Understanding the dual prices

Let's try to understand the meanining of the dual prices in this case. The *dual price lemma* – see the slide below – shows that – roughly speaking – the dual price predicts the change in the objective function if the right-hand side of the constraint inequality changes by 1.

Imagine that the owner of 15 lots adjacent to the development described above offers to sell them for \$60,000 total. Should you buy them?

Well, this amounts to changing the inequality $C + R \leq 100$ to $C + R \leq 115$. Since the dual price of lots is $\$4900, we predict a gain in profit of
$4900 \times 15$. For total price \$60,000 for the 15 lots amounts to a per-lot price of \$4000 per lot.

So, if the prediction is correct, we would make a profit of \$900 per lot!

Re-reading the fine print on the lemma below, however, we actually see that it isn't quite true "on the nose" that changes in the constraint values cause the objective function to increases by the corresponding multiple of the dual price – in fact, that increase is only an "upper bound" (however, see the remark below the lemma for some justification for why it is not an unreasonable estimate to use).

In fact, re-running the linear program after replacing the inequality constraint by $C+R \leq 115$, our profits increase from \$502,000 to \$563,895 i.e. by roughly \$62,000. After we spend the \$60,000 on the new lots, we only net $\$2,000$.*Sowemakeconsiderablylessthantheestimated* $900 \times 15 = 13,500$.

[11]:
```
def key_index(k,dict):
    keys = list(dict.keys())
    return keys.index(k)

def key_sbv(k,dict):
    return sbv(key_index(k,dict),len(list(dict.keys())))

def sbv(index,size):
    return np.array([1.0 if i == index else 0.0 for i in range(size)])


bprime = b + 15*key_sbv('lots',resources_avail)

primal_tweaked = linprog((-1)*c,A_ub = A,b_ub = bprime)

primal_tweaked
```

[11]:
```
         message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
         success: True
          status: 0
             fun: -563894.7368421053
               x: [ 7.368e+00  1.053e+02]
             nit: 3
           lower:  residual: [ 7.368e+00  1.053e+02]
                  marginals: [ 0.000e+00  0.000e+00]
           upper:  residual: [       inf        inf]
                  marginals: [ 0.000e+00  0.000e+00]
           eqlin:  residual: []
                  marginals: []
         ineqlin:  residual: [ 0.000e+00  0.000e+00  2.368e+00]
                  marginals: [-1.905e+01 -2.579e+01 -0.000e+00]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

As a final comment, one can recompute the slack vector for the "new" linear program (with $C+R \leq 115$) - one now finds no slack at all in either of the labor contraints (so they are both *binding*), but there is now slack in the *lot* constraint, which reflects an "oversupply" of lots.

[12]:
```
primal_tweaked.slack
```

[12]:
```
array([0.        , 0.        , 2.36842105])
```

### 6.1 Dual price lemma

Let's consider again a linear program $\mathcal{L}$ in standard form given by data $(\mathbf{c}, A, \mathbf{b})$.

Let $\Delta\mathbf{b} \in \mathbb{R}^r$ be a small perturbation of $\mathbf{b} \in \mathbb{R}^r$.

**Lemma:** Suppose that $\mathbf{x}^*$ is an optimal solution to the linear program $\mathcal{L}$ and that $\mathbf{x}'$ is an optimal solution to the linear program $\mathcal{L}_\Delta$ given by the data $(\mathbf{c}, A, \mathbf{b} + \Delta\mathbf{b})$. Then

$$\mathbf{c} \cdot \mathbf{x}' \leq \mathbf{c} \cdot \mathbf{x}^* + \Delta\mathbf{b}^T \cdot \mathbf{y}^*$$

where $\mathbf{y}^*$ is an optimal solution to the dual linear system $\mathcal{L}'$.

**Remark** One can actually prove equality in the lemma provided that the perturbation $\Delta\mathbf{b}$ vector is "small enough".

**Proof of Lemma:**

Note that the constraints of the unaltered dual $\mathcal{L}'$ are the same as those of the altered dual $\mathcal{L}'_a$. Thus, an optimal solution $\mathbf{y}^*$ for $\mathcal{L}'$ is at least feasible for $\mathcal{L}'_a$.

So we may apply the weak duality theorem to see that

$$(*) \quad \mathbf{c} \cdot \mathbf{x}' \leq (\mathbf{b} + \Delta\mathbf{b})^T \cdot \mathbf{y}^* = \mathbf{b}^T \cdot \mathbf{y}^* + \Delta\mathbf{b}^T \cdot \mathbf{y}^*.$$

However, by strong duality of the unaltered linear programs, we have

$$\mathbf{c} \cdot \mathbf{x}^* = \mathbf{b}^T \cdot \mathbf{y}^*;$$

substituting in $(*)$ we find

$$\mathbf{c} \cdot \mathbf{x}' \leq \mathbf{c} \cdot \mathbf{x}^* + \Delta\mathbf{b}^T \cdot \mathbf{y}^*$$

as required. **QED**