

8.3 The Simplex Algorithm



The **simplex algorithm** (or “method”) is an iterative technique for finding an optimal solution of the linear programming problem of section 8.2. The details of the algorithm are somewhat involved, but the basic idea is to start at a vertex of the feasible solution space S and traverse an edge of the polytope that leads to another vertex with a greater value of f . Then, repeat this process until there is no neighboring vertex with a greater value of f , at which point the solution is guaranteed to be optimal.

Rather than present the details of the algorithm, we choose to show an example using Python. There have been some improvements on the original algorithm that have been implemented into many standard software packages, including Python’s `scipy` package (Virtanen et al. 2019) module `scipy.optimize`.¹

Example 8.2

Maximize the objective function

$$f(x) = c \cdot x \quad (8.6)$$

for $x \in \mathbb{R}^2$ and

$$c = [5 \quad 2]^\top \quad (8.7)$$

subject to constraints

$$0 \leq x_1 \leq 10 \quad (8.8)$$

$$-5 \leq x_2 \leq 15 \quad (8.9)$$

$$4x_1 + x_2 \leq 40 \quad (8.10)$$

$$x_1 + 3x_2 \leq 35 \quad (8.11)$$

$$-8x_1 - x_2 \geq -75. \quad (8.12)$$

First, load some Python packages.

```
from scipy.optimize import linprog
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Encoding the Problem Before we can use `linprog`, we must first encode equations (8.6) and (8.8) into a form `linprog` will recognize. We begin with f , which we can write as $c \cdot x$ with the coefficients of c as follows.

1. Another Python package `pulp` (PuLP) is probably more popular for linear programming; however, we choose `scipy.optimize` because it has applications beyond linear programming.

```
| c = [-5, -2] # negative to find max
```

We've negated each constant because `linprog` *minimizes* f and we want to *maximize* f . Now let's encode the inequality constraints. We will write the left-hand side coefficients in the matrix A and the right-hand-side values in vector a such that

$$Ax \leq a. \quad (8.13)$$

Notice that one of our constraint inequalities is \geq instead of \leq . We can flip this by multiplying the inequality by -1 . We use simple lists to encode A and a .

```
| A = [
|     [4, 1],
|     [1, 3],
|     [8, 1]
| ]
| a = [40, 35, 75]
```

Now we need to define the lower l and upper u bounds of x . The function `linprog` expects these to be in a single list of lower- and upper-bounds of each x_i .

```
| lu = [
|     (0, 10),
|     (-5, 15),
| ]
```

We want to keep track of each step `linprog` takes. We can access these by defining a function callback, to be passed to `linprog`.

```
| x = [] # for storing the steps
| def callback(res): # called at each step
|     global x
|     print(f"nit = {res.nit}, x_k = {res.x}")
|     x.append(res.x.copy()) # store
```

Now we need to call `linprog`. We don't have any equality constraints, so we need only use the keyword arguments `A_ub=A` and `b_ub=a`. For demonstration purposes, we tell it to use the 'simplex' method, which is not as good as its other methods, which use better algorithms based on the simplex.

```
res = linprog(
    c,
    A_ub=A,
    b_ub=a,
    bounds=lu,
    method='simplex',
    callback=callback
)
x = np.array(x)
```

```
nit = 0, x_k = [ 0. -5.]
nit = 1, x_k = [10. -5.]
nit = 2, x_k = [8.75  5. ]
nit = 3, x_k = [7.72727273  9.09090909]
nit = 4, x_k = [7.72727273  9.09090909]
nit = 5, x_k = [7.72727273  9.09090909]
nit = 5, x_k = [7.72727273  9.09090909]
```

So the optimal solution $(x, f(x))$ is as follows.

```
print(f"optimum x: {res.x}")
print(f"optimum f(x): {-res.fun}")

optimum x: [7.72727273  9.09090909]
optimum f(x): 56.81818181818182
```

The last point was repeated

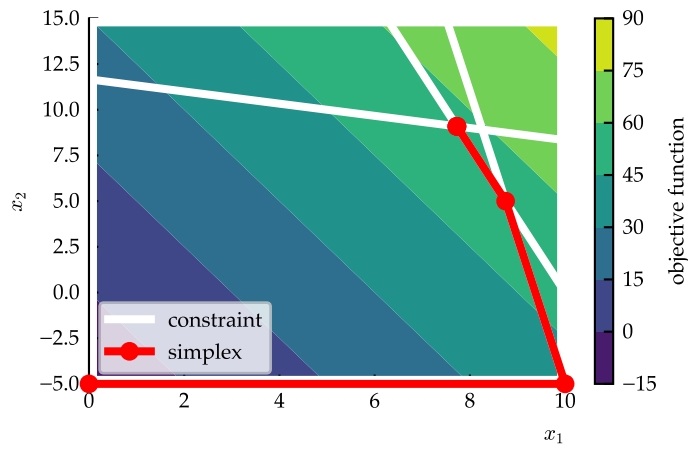
- Once because there was no adjacent vertex with greater $f(x)$ and
- Twice because the algorithm calls `callback` twice on the last step.

Plotting When the solution space is in \mathbb{R}^2 , it is helpful to graphically represent the solution space, constraints, and the progression of the algorithm. We begin by defining the inequality lines from A and a over the bounds of x_1 .

```
n = len(c) # number of variables x
m = np.shape(A)[0] # number of inequality constraints
x2 = np.empty([m,2])
for i in range(0,m):
    x2[i,:] = -A[i][0]/A[i][1]*np.array(lu[0]) + a[i]/A[i][1]
```


Now we plot a contour plot of f over the bounds of x_1 and x_2 and overlay the inequality constraints and the steps of the algorithm stored in `x`.

```
lu_array = np.array(lu)
fig, ax = plt.subplots()
mpl.rcParams['lines.linewidth'] = 3
# contour plot
X1 = np.linspace(*lu_array[0],100)
X2 = np.linspace(*lu_array[1],100)
X1, X2 = np.meshgrid(X1,X2)
F2 = -c[0]*X1 + -c[1]*X2 # negative because max hack
con = ax.contourf(X1,X2,F2)
cbar = fig.colorbar(con,ax=ax)
cbar.ax.set_ylabel('objective function')
# bounds on x
un = np.array([1,1])
opts = {'c':'w','label':None,'linewidth':6}
plt.plot(lu_array[0],lu_array[1,0]*un,**opts)
plt.plot(lu_array[0],lu_array[1,1]*un,**opts)
plt.plot(lu_array[0,0]*un,lu_array[1]**opts)
plt.plot(lu_array[0,1]*un,lu_array[1]**opts)
# inequality constraints
for i in range(0,m):
    p, = plt.plot(lu[0],x2[i,:],c='w')
    p.set_label('constraint')
# steps
plt.plot(
    x[:,0], x[:,1], '-o', c='r',
    clip_on=False, zorder=20, label='simplex'
)
plt.ylim(lu_array[1])
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.show()
```

Figure 8.3. Simplex method on f .

8.4 Problems



Problem 8.1  **CHORTLE** Consider the function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, defined as

$$f(\mathbf{x}) = \cos(x_1 - e^{x_2} + 2) \sin(x_1^2/4 - x_2^2/3 + 4) \quad (8.14)$$

Use the method of Barzilai and Borwein (1988) starting at $\mathbf{x}_0 = (1, 1)$ to find a minimum of the function.

Problem 8.2  **CUMMERBUND** Consider the functions (a) $f_1 : \mathbb{R}^2 \rightarrow \mathbb{R}$ and (b) $f_2 : \mathbb{R}^2 \rightarrow \mathbb{R}$ defined as

$$f_1(\mathbf{x}) = 4(x_1 - 16)^2 + (x_2 + 64)^2 + x_1 \sin^2 x_1 \quad (8.15)$$

$$f_2(\mathbf{x}) = \frac{1}{2} \mathbf{x} \cdot A \mathbf{x} - \mathbf{b} \cdot \mathbf{x} \quad (8.16)$$

where

$$A = \begin{bmatrix} 5 & 0 \\ 0 & 15 \end{bmatrix} \quad \text{and} \quad (8.17a)$$

$$\mathbf{b} = [-2 \quad 1]^\top. \quad (8.17b)$$

Use the method of Barzilai and Borwein (1988) starting at some \mathbf{x}_0 to find a minimum of each function.

Problem 8.3  Consider the function $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ defined as

$$f(\mathbf{x}) = \sin x_1 + \cos x_2 + \sqrt{(x_1 - 2)^2 + (x_2 + 1)^2}. \quad (8.18)$$

Use the gradient descent method of Barzilai and Borwein (1988) with a step size of $T = 10^{-8}$ starting at (a) $\mathbf{x}_0 = [0 \quad 0]^\top$ and (b) $\mathbf{x}'_0 = [2 \quad 0]^\top$ to find minima of f . (c) Explain why the two minima are different.

Problem 8.4  **MELTY** Maximize the objective function

$$f(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x} \quad (8.19a)$$

for $\mathbf{x} \in \mathbb{R}^3$ and

$$\mathbf{c} = [3 \quad -8 \quad 1]^\top \quad (8.19b)$$