Independent Study 2018

Ziyan (Jessica) Feng, Nhu Do

December 20, 2018

1 Introduction

2 Mathematical background

2.1 The Lagrange dual function

2.1.1 The Lagrangian

Consider an optimization in standard form:

$$\begin{array}{lll} \text{minimize: } f_0(x) & & \\ \text{subject to: } f_i(x) \leq 0, & i = 1, \dots, m & \\ \qquad\qquad\quad h_i(x) = 0, & i = 1, \dots, p, & \end{array} \tag{1}$$

with variable $x \in \mathbb{R}^n$. Assume its domain $\mathcal{D}$ is nonempty, and denote by p^* the optimal value of problem (1), which is not necessarily convex.

The basic idea in Lagrangian duality is to augment the objective function with a weighted sum of the constraint functions. We defined the *Lagrangian* $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ associated with this problem as:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$$

with domain $L = \mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$. We refer to λ_i as the *Lagrange multiplier* associated with the ith inequality constraint, and ν_i the Lagrange multiplier associated with the ith equality constraint. Vectors λ, ν are the *dual variables* or *Lagrange multiplier vectors.*

2.1.2 The Lagrange dual function

We define the *Lagrange dual function* $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ as the minimum value of the Lagrangian over x:

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu) = \inf_{x \in \mathcal{D}} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right)$$

As g is the pointwise infimum of a family of affine functions of (λ, ν), it is concave.

2.1.3 Lower bounds on optimal value

Let's denote by $x \succeq y$ the componentwise inequality between vectors x and y.
The dual function yields lower bounds on the optimal value p^* of problem (1): For any $\lambda \succeq 0$ and any ν we have:

$$g(\lambda, \nu) \leq p^*, \tag{2}$$

which can be easily verified. The dual function gives a nontrivial lower bound on p^* only when $\lambda \succeq 0$ and $g(\lambda, \nu) > -\infty$. Such a pair (λ, ν) is called *dual feasible.*

2.1.4 Linear approximation interpretation

We can rewrite the original problem (1) as an unconstraine problem,

$$\text{minimize } f_0(x) + \sum_{i=1}^{m} I_f(f_i(x)) + \sum_{i=1}^{p} I_0(h_i(x)), \tag{3}$$

where:

$$I_f(u) = \begin{cases} 0, & u \leq 0 \\ +\infty, & u > 0 \end{cases} \quad , \quad I_g(u) = \begin{cases} 0, & u = 0 \\ +\infty, & u \neq 0 \end{cases}$$

The function $I_f(u), I_g(u)$ can be interpreted as expressing our displeasure associated with the violation of the corresponding constraint.
We can approximate problem (3) as minimizing the Lagrangian $L(x, \lambda, \nu)$, and the dual function value $g(\lambda, \nu)$ is the optimal value of the problem

$$\text{minimize } L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x), \text{ assuming all } \lambda_i > 0 \tag{4}$$

Since $\lambda_i u \leq I_f(u)$ and $\nu_i u \leq I_0(u)$ for all u, it's clear that the dual function yields a lower bound on the optimal value of the original problem.

2.1.5 The Lagrange dual problem

For each pair (λ, ν) with $\lambda \succeq 0$, the Lagrange dual function gives us a lower bound on the optimal value p^* of the optimization problem (1). The best lower bound can be found by solving the optimization problem:

$$\begin{aligned} &\text{maximize: } g(\lambda, \nu) \\ &\text{subject to: } \lambda \succeq 0. \end{aligned} \tag{5}$$

This problem is called the *Lagrange dual problem* associated with problem (1). The original problem (1) is called the *primal problem.* We refer to (λ^*, ν^*) as *dual optimal* or *optimal Lagrange multipliers.*
The Lagrange dual problem (5) is a convex optimization problem, since the objective function is concave and the constraint is convex. This is the case regardless of whether the primal problem is convex.

2.1.6 Weak duality and Strong duality

The optimal value of the Lagrange dual problem, which we denote d^*, is, by definition, the best lower bound on p^* that can be obtained from the Lagrange dual function. In particular, we have:

$$d^* \leq p^*, \tag{6}$$

This property is called *weak duality.* The inequality (6) holds when d^* and p^* are infinite. The bound (6) is sometimes used to find a lower bound on the optimal value of a problem that is difficult to solve, since the dual problem is always convex, and in many cases can be solved efficiently.

If the equality

$$d^* = p^* \tag{7}$$

holds, then we say that *strong duality* holds.

In general, strong duality does not hold. But if the primal (**??**) is convex, we usually (though not always) have strong duality.

A condition that guarantee strong duality is *Slater's condition*: there exists an $x \in$ **relint** $\mathcal{D}$ such that

$$f_i(x) < 0, i = 1, \dots, m, \qquad Ax = b. \tag{8}$$

where **relint** $\mathcal{D}$ denotes the relative interior of $\mathcal{D}$. Slater's theorem states that strong duality holds, if Slater's condition holds, and the problem is convex. Besides, the affine inequalities do not need to hold with strict inequality. Slater's condtion also implies that dual optimal value is attained when $d^* > -\infty$, i.e., there exists a dual feasible (λ^*, ν^*) with $g(\lambda^*, \nu^*) = d^* = p^*$.

2.2 Optimality conditions

2.2.1 Complementary slackness

Suppose that the primal and dual values are attained and equal (so strong duality holds). Let x^* be a primal optimal and (λ^*, ν^*) be a dual optimal point. There are some important observations:

- x^* minimizes $L(x, \lambda^*, \nu^*)$ over x (The Lagrangian $L(x, \lambda^*, \nu^*)$ can have other minimizers, x^* is simple a minimizer).
- *Complementary slackness:* $\lambda_i^* f_i(x^*) = 0$, for $i = 1, \dots, m$.

2.2.2 KKT optimality conditions

Assume that the functions $f_0, f_1, \dots, f_m, h_1, \dots, h_p$ are differentiable.

KKT conditions for nonconvex problem
Since x^* minimizes $L(x, \lambda^*, \nu^*)$ over x, by first-order necessary condition for a minimum, its gradient must vanish at x^*, i.e.,

$$\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x^*) = 0$$

.
Thus we have

$$
\begin{aligned}
f_i(x^*) &\leq 0, i = 1, \ldots, m \\
h_i(x^*) &= 0, i = 1, \ldots, p \\
\lambda_i^* &\geq 0, i = 1, \ldots, m \\
\lambda_i^* f_i(x^*) &= 0, i = 1, \ldots, m \\
\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x^*) &= 0
\end{aligned}
$$

which are called the *Karush-Kuhn-Tucker* (KKT) conditions.

For any optimization problem with **differentiable** objective and constraint functions for which **strong duality** obtains, any pair of primal and dual optimal points must satisfy the KKT conditions.

KKT conditions for convex problems
When the primal problem is convex, if f_i are convex and h_i are affine, and $\widetilde{x}, \widetilde{\lambda}, \widetilde{\nu}$ are points that satisfy the KKT conditions, then $\widetilde{x}$ and $(\widetilde{\lambda}, \widetilde{\nu})$ are primal and dual optimal.
If a convex optimization problem with differentiable objective and constraint functions satisfies Slater's condition, then the KKT conditions proves necessary and sufficient conditions for optimality.

2.3 Augmented Lagrangian and the Method of Multipliers

Consider the problem:

$$
\begin{array}{l} \text{minimize: } f(x) \\ \text{subject to: } Ax = b \end{array} \tag{9}
$$

The augmented Lagrangian for this problem is:

$$
L_\rho(x, \nu) = f(x) + \nu^T (Ax - b) + (\rho/2) \cdot ||Ax - b||^2
$$

where $\rho > 0$ is called the *penalty parameter*. Note that L_0 is the standard Lagrangian for the problem.
The algorithm:

$$
x^{k+1} = \text{argmin}_x L_\rho(x, \nu^k) \tag{10}
$$

$$
\nu^{k+1} = \nu^k + \rho(Ax^{k+1} - b) \tag{11}
$$

is the *Method of Multipliers* for solving (9). This is the same as standard dual ascent, but the $x-$minimization step uses the augmented Lagrangian, and the penalty parameter ρ is used as the step size. The method of multipliers converges under far more general conditions than dual ascent. By using the ρ as the step size in the dual update, the iterate (x^{k+1}, ν^{k+1}) is dual feasible. As the method of multipliers proceed, the primal residual $Ax^{k+1} - b$ converges to zero, yielding optimally.

Disadvantage: When f is separable, the augmented Lagrangian L_ρ is not separable, so the $x-$minimization step (10) cannot be carried out separately in parallel for each x_i.

(We say that f is *separable* (with respect to a partition or splitting of the variable into subvectors), meaning that $f(x) = \sum_{i=1}^{N} f_i(x_i)$ where $x = (x_1, \ldots, x_N)$ and the variables $x_i \in \mathbb{R}^{n_i}$ are subvectors of x).

2.4 Alternating Direction Method of Multipliers (ADMM)

2.4.1 Algorithm

ADMM solves problems in the form

$$\begin{array}{l} \text{minimize: } f(x) + g(z) \\ \text{subject to: } Ax + Bz = c \end{array} \tag{12}$$

with variables $x \in \mathbb{R}^n, z \in \mathbb{R}^m$, where $A \in \mathbb{R}^{p \times n}, B \in \mathbb{R}^{p \times m}$, and $c \in \mathbb{R}^p$. The optimal value of this problem will be denoted by

$$p^* = \inf\{f(x) + g(z) \mid Ax + Bz = c\}$$

We form the augmented Lagrangian

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}||Ax + Bz - c||^2$$

ADMM consists of the iterations

$$\begin{aligned} x^{k+1} &:= \arg\min_x L_\rho(x, z^k, y^k) \\ z^{k+1} &:= \arg\min_z L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &:= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \end{aligned} \tag{13}$$

where $\rho > 0$.

2.4.2 Convergence

Assumption 1. The (extended-real-valued) functions $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ and $g : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$ are closed, proper, and convex.
In other words, the function f satisfies assumption 1 iff its epigraph

$$\text{epi } f = \{(x, t) \in \mathbb{R}^n \times \mathbb{R} \mid f(x) \leq t\}$$

is a closed nonempty convex set. Also note that f, g could be nondifferentiable and assume the value $+\infty$.

Assumption 2. The unaugmented Lagrangian L_0 has a saddle point. Explicitly, there exist (x^*, z^*, y^*), not necessarily unique, such that

$$L_0(x^*, z^*, y) \leq L_0(x^*, z^*, y^*) \leq L_0(x, z, y^*)$$

holds for all x, z, y.
This implies that (x^*, z^*) is a solution to (12), y^* is dual optimal, and the optimal values of the primal and dual problems are equal.

Under assumptions 1 and 2, the ADMM iterates satisfy the following:

- *Residual convergence.* $r^k \to 0$ as $k \to \infty$, where $r^k = Ax^k + Bz^k - c$, i.e., the iterates approach feasibility.
- *Objective convergence.* $f(x^k) + g(z^k) \to p^*$ as $k \to \infty$, i.e., the objective function of the iterates approaches the optimal value.
- *Dual variable convergence.* $y^k \to y^*$ as $k \to \infty$, where y^* is a dual optimal point.

Note that x^k and z^k need not converge to optimal values.

3 Problem formulation

Variables:

- b_i: habitat target for species i
- x_j: amount of money invested for watershed j
- $f_j(x_j)$: total habitat in watershed j if x_j money is invested
- a_{ij}: whether species i is available in watershed j
  $a_{ij} = \begin{cases} 1, & \text{if species i is in watershed j} \\ 0, & \text{otherwise} \end{cases}$

Optimization problem

$$\begin{aligned} \min \ & \sum_j x_j \\ \text{subject to } & x_j \geq 0, \forall j \\ & \sum_j a_{ij} f_j(x_j) \geq b_i, \forall i \end{aligned}$$

- Lagrangian:
$$L(x, \lambda) = \sum_j x_j + \sum_i \lambda_i (b_i - \sum_j a_{ij} f_j(x_j)) \tag{14}$$

- Dual function:
$$g(\lambda) = \min_{x \geq 0} L(x, \lambda) \tag{15}$$

- Dual problem:
$$\max_{\lambda \geq 0} g(\lambda) \tag{16}$$

3.1 Gradient Ascent Approach

The dual problem (16) can be solved by using gradient ascent. **Gradient ascent** maximizes a function by moving in the positive gradient direction, or minimizes a function by moving in the negative gradient direction.

$$\max_{\lambda \geq 0} g(\lambda)$$

$$\text{Gradient Ascent} : \lambda_{t+1} = \lambda_t + \alpha_t \nabla g(\lambda)$$

$$\nabla g(\lambda) = \nabla_\lambda L(x^*_\lambda, \lambda) = \sum_i b_i - \sum_j a_{ij} f_j(x_j)$$

where x^*_λ is the minimizer of $L(x, \lambda)$.
Gradient ascent produces $\lambda, \lambda^2 \dots \lambda^*$ that converges to λ^*. This is optimal for dual but not necessarily optimal for primal problem, since we get $x, x^2 \dots x^*$ from λ.

Project gradient ascent minimize subject to a constraint by moving in the direction of negative gradient, then projecting onto feasible set.

$$\max_{\lambda \geq 0} g(\lambda)$$

$$y_{t+1} = \lambda_t + \alpha_t \nabla g(\lambda)$$

$$\lambda_{t+1} = \arg\max_{\lambda \geq 0} ||y_{t+1} + \lambda_t||$$

Dual Ascent produces (both gradient ascent and project gradient ascent):

$$\lim_{t \to \infty} g(\lambda^t) = g(\lambda^*)$$

We also want:

$$\lim_{t \to \infty} f(x^t) = f(x^*)$$

We can get x^*_λ by using each λ_t, but can not prove primal is optimal.
Thus gradient ascent can not solve our problem since we can only get optima for dual problem, and we can not prove strong duality. We will introduce another method, ADMM, in next section, which gives both λ^* and x^*.

3.2 ADMM approach

Assume that there are m species and n watersheds. We introduce slack variables s_i's and rewrite the primal as follows:

$$\begin{aligned}
\text{minimize } & \sum_{j=1}^{n} x_j \\
\text{subject to } & \sum_{j=1}^{n} a_{ij} f_j(x_j) = b_i + s_i, \forall i = 1, \dots, m \\
& x_j \geq 0, \forall j = 1, \dots, n \\
& s_i \geq 0, \forall i = 1, \dots, m
\end{aligned}$$

Let $A \in \mathbb{R}^{m \times n}$ such that $(A)_{ij} = a_{ij}$, $f(x) = \begin{bmatrix} f_1(x_1) \\ \vdots \\ f_n(x_n) \end{bmatrix}$, $b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$, $s = \begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix}$.

- The primal now becomes:

$$\begin{aligned} \text{minimize } & \sum_{j=1}^{n} x_j \\ \text{subject to } & Af(x) = b + s \\ & x_j \geq 0, \forall j = 1, \dots, n \\ & s_i \geq 0, \forall i = 1, \dots, m \end{aligned}$$

- Augmented Lagrangian:

$$\begin{aligned} L_\rho(x,s,y) &= \sum_{j=1}^{n} x_j + \sum_{i=1}^{m} y_i(b_i + s_i - \sum_{j=1}^{n} a_{ij} f_j(x_j)) + \frac{\rho}{2} \sum_{i=1}^{m} \left(b_i + s_i - \sum_{j=1}^{n} a_{ij} f_j(x_j) \right)^2 \\ &= \sum_{j=1}^{n} x_j + y^T(b + s - Af(x)) + \frac{\rho}{2} ||b + s - Af(x)||^2 \end{aligned}$$

  note: x and s are primal variables, y is dual variable. y is unconstraint.

- Updating rule:
  We first introduce some notations. In the $(k+1)$-th iteration that is updating vector s, let's say it is going to update component s_i next. We denote by $s_{<i}^{k+1}$ values of the first $i-1$ components of vector s that have been updated in the $(k+1)$-th iteration. Similarly, denote by $s_{>i}^{k}$ the last $m-i$ components of vector s that have been updated in the k-th iteration, but not yet updated in the $(k+1)$-th iteration. Then we can write $s = \begin{bmatrix} s_{<i}^{k+1} \\ -- \\ s_i \\ -- \\ s_{>i}^{k} \end{bmatrix}$. We use similar notations for x with $x_{<h}, x_h$ and $x_{>h}$. The update rules are as follows:

+ Update x: (indexing match with for_species function in Python code)

$$
\begin{aligned}
x_j^{(k+1)} &= \arg\min_{x_j} L_\rho(x_{<j}, x_j, x_{>j}, s^k, y^k) \\
&= \arg\min_{x_j} \left\{ x_j - \sum_{i=1}^{m} y_i^k \cdot a_{ij} \cdot f_j(x_j) + \frac{\rho}{2} \sum_{i=1}^{m} \left(b_i + s_i^k - \sum_{h \neq j} a_{ih} f_h(x_h) - a_{ij} f_j(x_j) \right)^2 \right\} \\
&= \arg\min_{x_j} \Bigg\{ x_j - f_j(x_j) \sum_{i=1}^{m} y_i^k a_{ij} + \\
&\qquad \frac{\rho}{2} \sum_{i=1}^{m} \left[(a_{ij} f_j(x_j))^2 - 2 a_{ij} f_j(x_j) \left(b_i + s_i^k - \sum_{h<j} a_{ih} f_h(x_h^{k+1}) - \sum_{h>j} a_{ih} f_h(x_h^k) \right) \right] \Bigg\} \\
&= \arg\min_{x_j} \Bigg\{ x_j - f_j(x_j) \sum_{i=1}^{m} y_i^k \cdot a_{ij} + \\
&\qquad \frac{\rho}{2} f_j(x_j)^2 \sum_{i=1}^{m} a_{ij}^2 - \rho \cdot f_j(x_j) \sum_{i=1}^{m} a_{ij} \left(b_i + s_i^k - \sum_{h<j} a_{ih} f_h(x_h^{k+1}) - \sum_{h>j} a_{ih} f_h(x_h^k) \right) \Bigg\} \\
&= \arg\min_{x_j} \{ x_j - \alpha \cdot f_j(x_j) + \beta \cdot f_j^2(x_j) \}, \text{ where } \alpha \text{ is unconstrained, } \beta > 0 \\
&= \arg\min_{x_j} (F_{x_j})
\end{aligned}
$$

+ Update s:

$$
\begin{aligned}
s_i^{(k+1)} &= \arg\min_{s_i} L_\rho(x^{k+1}, s_{<i}, s_i, s_{>i}, y^k) \\
&= \arg\min_{s_i} \left[y_i^k s_i + \frac{\rho}{2} (b_i + s_i - (Af(x^{k+1}))_i)^2 \right] \\
&= \arg\min_{s_i} \left[y_i^k s_i + \frac{\rho}{2} (s_i^2 + 2 b_i s_i - 2 s_i (Af(x^{k+1}))_i) \right] \\
&= \arg\min_{s_i} \left[y_i^k s_i + \frac{\rho}{2} \left(s_i^2 + 2 b_i s_i - 2 s_i \sum_{j=1}^{n} a_{ij} f_j(x_j^{k+1}) \right) \right] \\
&= \arg\min_{s_i} (F_{s_i})
\end{aligned}
$$

+ Update y:

$$
y^{k+1} = y^k + \rho \sum_{i=1}^{m} \left(b_i + s_i^{k+1} - \sum_{j=1}^{n} a_{ij} f_j(x_j^{k+1}) \right)
$$

- Square of convex function is also convex under certain condition
  Let $g(x)$ be a convex function and $g(x) \geq 0 \ \forall x$.
  $\dfrac{d^2}{dx^2} \geq 0$ because $g(x)$ is convex.

Then

$$\begin{aligned}
&\frac{d^2}{dx^2}(g(x))^2 \\
&= \frac{d}{dx}\left(2g(x)\frac{d}{dx}g(x)\right) \\
&= 2\frac{d}{dx}g(x)\frac{d}{dx}g(x) + 2g(x)\frac{d^2}{dx^2}g(x) \\
&= 2\left(\frac{d}{dx}g(x)\right)^2 + 2g(x)\frac{d^2}{dx^2}g(x)
\end{aligned}$$

The square part is non-negative, $g(x) \geq 0$, and $\frac{d^2}{dx^2}g(x) \geq 0$.

Thus $\frac{d^2}{dx^2}(g(x))^2$. The square of a convex function is also convex.

- Convexity of the functions used to update components of x, s:
  - + Because $f_h(x_h)$ is concave, bijective, and presumably strictly increasing, its inverse $f_h^{-1}(x_h)$ is convex. Besides, the function $-\alpha \cdot f_h(x_h) + \beta \cdot f_h^2(x_h)$ (view this as a function of $f_h(x_h)$) is convex (since $\beta > 0$). Thus, $F_{x_h} = f_h^{-1}(x_h) + (-\alpha \cdot f_h(x_h) + \beta \cdot f_h^2(x_h))$ is convex.
  - + F_{s_i} is a quadratic function of s_i. Since $\frac{\rho}{2} > 0$, the function F_{s_i} is convex.
- Checking KKT conditions for the formulation with slack variables:
  (1) (We don't have inequality constraints)
  (2) Checking $Af(x^*) \overset{?}{=} b + s^*$
  (3) (We don't have dual variables that correspond to the inequality constraints)
  (4) (No need to check complementary slackness since we don't have inequality constraints)
  (5) Checking $\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x^*) \overset{?}{=} 0$
  For our formulation, and this condition becomes:

$$\nabla(x_1 + \ldots + x_n + 0 \cdot s_1 + \ldots + 0 \cdot s_m) - \sum_{i=1}^{m} y_i^* \cdot \nabla\left(\sum_{h=1}^{n} b_i + s_i - a_{ih} f_h(x_h)\right)$$

$$= \begin{bmatrix}1\\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0\end{bmatrix} - \left(y_1^* \cdot \begin{bmatrix} a_{11} \cdot f_1'(x_1) \\ \vdots \\ a_{1n} \cdot f_n'(x_n) \\ 1 \\ \vdots \\ 0 \end{bmatrix} + \ldots + y_m^* \cdot \begin{bmatrix} a_{m1} \cdot f_1'(x_1) \\ \vdots \\ a_{mn} \cdot f_n'(x_n) \\ 0 \\ \vdots \\ 1 \end{bmatrix}\right)$$

$$= \begin{bmatrix}1\\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0\end{bmatrix} - \begin{bmatrix} f_1'(x_1) \sum_{i=1}^{m} y_i^* \cdot a_{i1} \\ \vdots \\ f_n'(x_n) \sum_{i=1}^{m} y_i^* \cdot a_{in} \\ y_1^* \\ \vdots \\ y_m^* \end{bmatrix} \overset{?}{\neq} \begin{bmatrix}0\\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0\end{bmatrix}$$

3.3 Deriving KKT conditions for our problem

First, let's write a version of the problem that include Lagrange multipliers, i.e. dual variables, for the non-negativity constraints on x and s.

- **Primal**

$$\begin{aligned} &\text{minimize} \sum_{j=1}^{n} x_j \\ &\text{subject to} \sum_{j=1}^{n} a_{ij} f_j(x_j) = b_i + s_i, \forall i = 1, \ldots, m && \text{(duals: } y_1, \ldots, y_m : \text{ free)} \\ &x_j \geq 0, \forall j = 1, \ldots, n && \text{(duals: } w_1, \ldots, w_n \leq 0) \\ &s_i \geq 0, \forall i = 1, \ldots, m && \text{(duals: } z_1, \ldots, z_m \leq 0) \end{aligned}$$

- **Lagrangian**

$$L(x, s, y, w, z) = \sum_{j=1}^{n} x_j + \sum_{i=1}^{m} y_i \left(b_i + s_i - \sum_{j=1}^{n} a_{ij} f_j(x_j)\right) + \sum_{j=1}^{n} w_j x_j + \sum_{i=1}^{m} z_i s_i$$

- **KKT conditions for this version of the problem**

(1)

$$x_j \overset{?}{\geq} 0, \forall j = 1, \ldots, n$$

$$s_i \overset{?}{\geq} 0, \forall i = 1, \ldots, m$$

(2) Checking $Af(x^*) \overset{?}{=} b + s$
(3) Checking:

$$w_j \overset{?}{\leq} 0, \forall j = 1, \ldots, n$$
$$z_i \overset{?}{\leq} 0, \forall i = 1, \ldots, m$$

(4) (Complementary slackness)

$$w_j x_j \overset{?}{=} 0, \forall j = 1, \ldots, n$$
$$z_i s_i \overset{?}{=} 0, \forall i = 1, \ldots, m$$

(5) (Stationary) (Is it correct that there are no stationary conditions for duals corresponding to inequality constraint, based on B&V 5.5.3?)

$$\left(\frac{\partial L}{\partial x} \overset{?}{=} 0\right) \qquad \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} f_1'(x_1) \sum_{i=1}^{m} y_i^* \cdot a_{i1} \\ \vdots \\ f_n'(x_n) \sum_{i=1}^{m} y_i^* \cdot a_{in} \end{bmatrix} + \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \overset{?}{=} \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\left(\frac{\partial L}{\partial s} \overset{?}{=} 0\right) \qquad \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} + \begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix} \overset{?}{=} \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

Note: y is free, $w \leq 0, z \leq 0$

3.3.1 KKT conditions for our problem

(1)

$$x_j \overset{?}{\geq} 0, \forall j = 1, \ldots, n$$
$$s_i \overset{?}{\geq} 0, \forall i = 1, \ldots, m$$

(2) Checking $Af(x^*) \overset{?}{=} b + s$
(3) (Combining complementary slackness and Stationary conditions):

- If $x_j > 0$ for some j, we need to have: $1 = f_j'(x_j) \sum_{i=1}^{m} y_i^* \cdot a_{ij}$.

  If $x_j = 0$ for some j, we only need: $1 \geq f_j'(x_j) \sum_{i=1}^{m} y_i^* \cdot a_{ij}$.
- If $s_i > 0$ for some i, we need $y_i = 0$.
  If $s_i = 0$ for some i, y_i can be anything.

3.4 Experiments and Notes

3.4.1 Adjusting ρ

In the original paper, there seems to be not enough information on how to set the penalty parameter ρ, which is used as the step size. We did several experiments on setting ρ and recorded the number of iterations it takes to converge as well as performance on getting optimal solution.
We used the simple 3 species and 3 watersheds problem with the following a-matrix and b-vector $[1.5e+06, 3.5e+06, 3e+06]$

```python
# a[i][j] = 1 if species i is available in watershed j, 0 otherwise
a = np.array([[1, 0, 0],
              [1, 1, 1],
              [1, 0, 1]])

iters = 5000
```

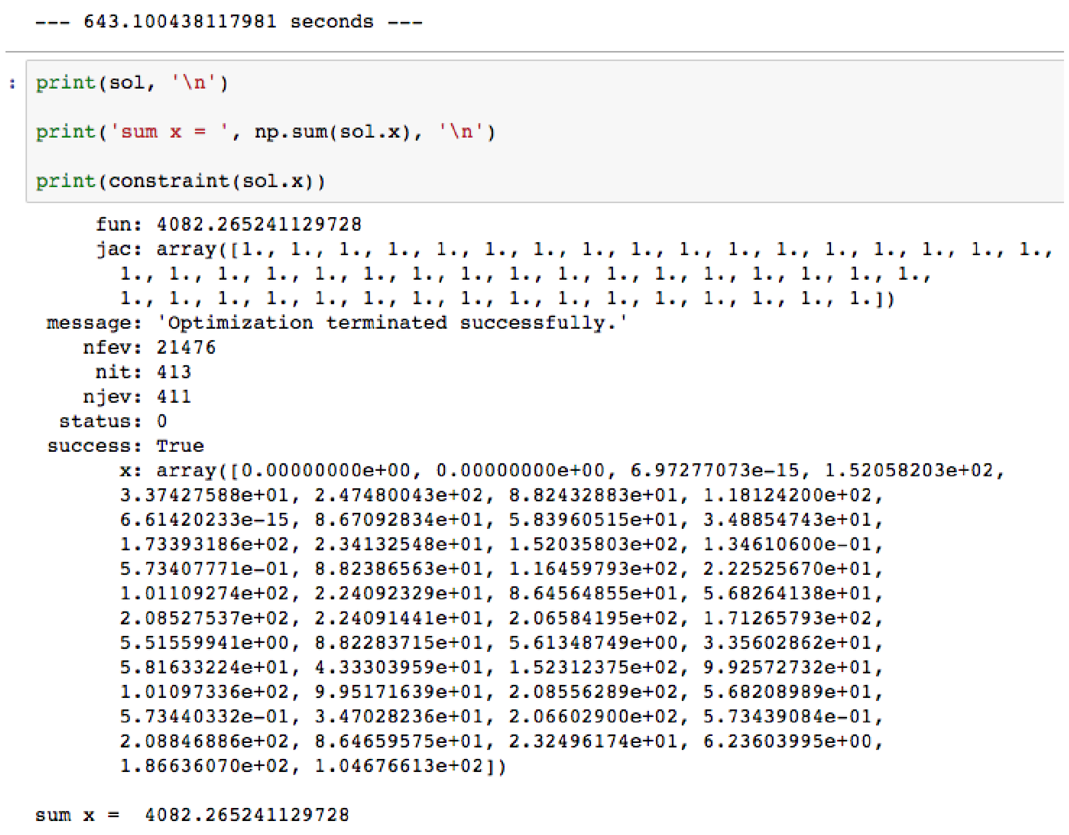
	number of iterations to converge	performance
optimal solution by scipy	19	
$\rho = 0.1$	did not converge in 20000 iterations	
$\rho = 1$	did not converge in 20000 iterations	
$\rho = 10$	3130	x=[12995, 165, 531] specie target=[1.45, 3.5, 3.0]
$\rho = 100$	316	x=[13937, 162, 500] specie target=[1.495, 3.5, 3.0]
$\rho = 1000$	60	x=[14000, 165, 495] specie target=[1.4976, 3.5, 3.0]
$\rho = 10000$	31	x=[14000, 165, 495] specie target=[1.4976, 3.5, 3.0]
$\rho = 100000$	169	x=[14065, 165, 494]specie target=[1.5, 3.5, 3.0]
$\rho = 1000000$	1650	x=[14065, 165, 494] specie target=[1.5, 3.5, 3.0]

More detailed performance data on
https://sites.google.com/mtholyoke.edu/admm-18/jessicas-weekly-report/week-10-nov-20th

3.4.2 A specific example num_species = 50, num_watersheds_total = 50

In one example scipy.minimize solve the problem in 643.100438117981 seconds, 413 iterations, with message "Optimization terminated successfully".

(To replicate this problem, use the 11.28 python code with:
Make up roi curves: np.random.seed(2032)
Make up matrix A and vector b: np.random.seed(6873480)
and no other changes)

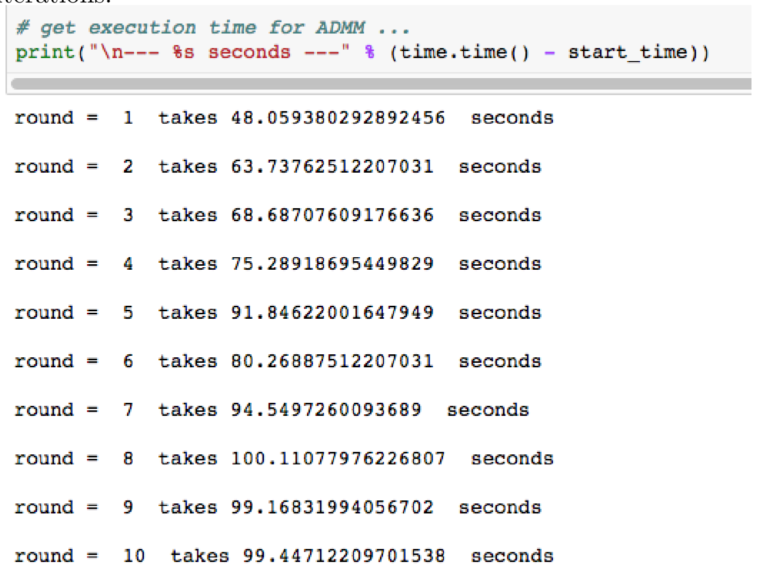


```
--- 643.100438117981 seconds ---
```

```
: print(sol, '\n')

  print('sum x = ', np.sum(sol.x), '\n')

  print(constraint(sol.x))
```

```
     fun: 4082.265241129728
     jac: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
 message: 'Optimization terminated successfully.'
    nfev: 21476
     nit: 413
    njev: 411
  status: 0
 success: True
       x: array([0.00000000e+00, 0.00000000e+00, 6.97277073e-15, 1.52058203e+02,
       3.37427588e+01, 2.47480043e+02, 8.82432883e+01, 1.18124200e+02,
       6.61420233e-15, 8.67092834e+01, 5.83960515e+01, 3.48854743e+01,
       1.73393186e+02, 2.34132548e+01, 1.52035803e+02, 1.34610600e-01,
       5.73407771e-01, 8.82386563e+01, 1.16459793e+02, 2.22525670e+01,
       1.01109274e+02, 2.24092329e+01, 8.64564855e+01, 5.68264138e+01,
       2.08527537e+02, 2.24091441e+01, 2.06584195e+02, 1.71265793e+02,
       5.51559941e+00, 8.82283715e+01, 5.61348749e+00, 3.35602862e+01,
       5.81633224e+01, 4.33303959e+01, 1.52312375e+02, 9.92572732e+01,
       1.01097336e+02, 9.95171639e+01, 2.08556289e+02, 5.68208989e+01,
       5.73440332e-01, 3.47028236e+01, 2.06602900e+02, 5.73439084e-01,
       2.08846886e+02, 8.64659575e+01, 2.32496174e+01, 6.23603995e+00,
       1.86636070e+02, 1.04676613e+02])

sum x =  4082.265241129728
```

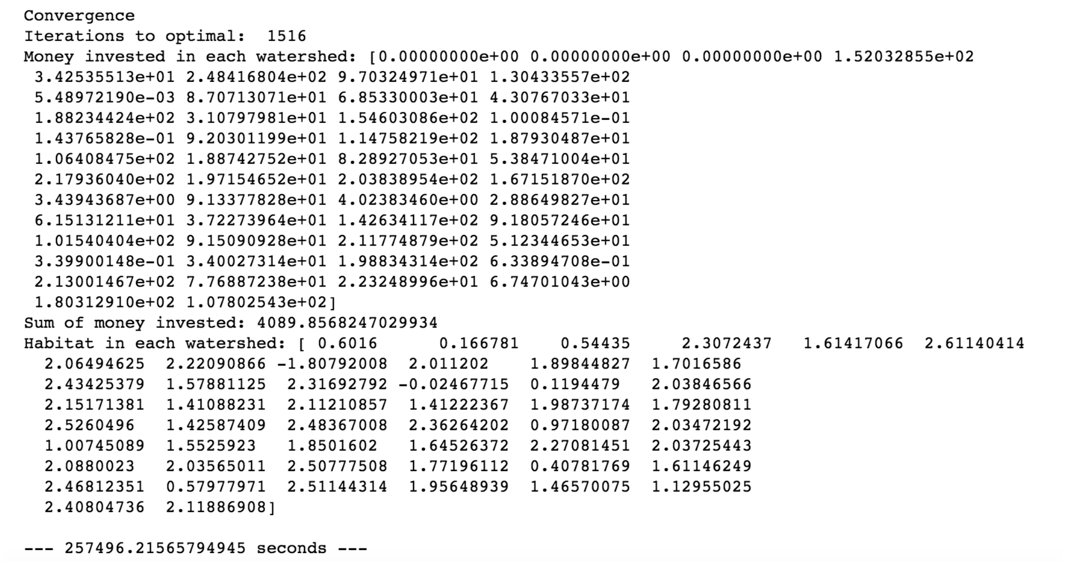
We also tried to use ADMM to solve this problem. This is how long it takes in the first few iterations.



```
# get execution time for ADMM ...
print("\n--- %s seconds ---" % (time.time() - start_time))
```

```
round =  1  takes 48.059380292892456  seconds

round =  2  takes 63.73762512207031  seconds

round =  3  takes 68.68707609176636  seconds

round =  4  takes 75.28918695449829  seconds

round =  5  takes 91.84622001647949  seconds

round =  6  takes 80.26887512207031  seconds

round =  7  takes 94.5497260093689  seconds

round =  8  takes 100.11077976226807  seconds

round =  9  takes 99.16831994056702  seconds

round =  10  takes 99.44712209701538  seconds
```

The 11.28 Python code took ADMM 71.5 hours, 1516 iterations to solve this problem, with slightly worse solution than that obtained by scipy.minimize. See 11.28 (2) Python code for details.



```
Convergence
Iterations to optimal:  1516
Money invested in each watershed: [0.00000000e+00 0.00000000e+00 0.00000000e+00 1.52032855e+02
 3.42535513e+01 2.48416804e+02 9.70324971e+01 1.30433557e+02
 5.48972190e-03 8.70713071e+01 6.85330003e+01 4.30767033e+01
 1.88234424e+02 3.10797981e+01 1.54603086e+02 1.00084571e-01
 1.43765828e-01 9.20301199e+01 1.14758219e+02 1.87930487e+01
 1.06408475e+02 1.88742752e+01 8.28927053e+01 5.38471004e+01
 2.17936040e+02 1.97154652e+01 2.03838954e+02 1.67151870e+02
 3.43943687e+00 9.13377828e+01 4.02383460e+00 2.88649827e+01
 6.15131211e+01 3.72273964e+01 1.42634117e+02 9.18057246e+01
 1.01540404e+02 9.15090928e+01 2.11774879e+02 5.12344653e+01
 3.39900148e-01 3.40027314e+01 1.98834314e+02 6.33894708e-01
 2.13001467e+02 7.76887238e+01 2.23248996e+01 6.74701043e+00
 1.80312910e+02 1.07802543e+02]
Sum of money invested: 4089.8568247029934
Habitat in each watershed: [ 0.6016      0.166781    0.54435     2.3072437   1.61417066  2.61140414
  2.06494625  2.22090866 -1.80792008  2.011202    1.89844827  1.7016586
  2.43425379  1.57881125  2.31692792 -0.02467715  0.1194479   2.03846566
  2.15171381  1.41088231  2.11210857  1.41222367  1.98737174  1.79280811
  2.5260496   1.42587409  2.48367008  2.36264202  0.97180087  2.03472192
  1.00745089  1.5525923   1.8501602   1.64526372  2.27081451  2.03725443
  2.0880023   2.03565011  2.50777508  1.77196112  0.40781769  1.61146249
  2.46812351  0.57977971  2.51144314  1.95648939  1.46570075  1.12955025
  2.40804736  2.11886908]

--- 257496.21565794945 seconds ---
```

3.4.3 Observations and thoughts on experiments

A random observation: In some small-sized problems (num_species = 5, num_watersheds_total = 5), there's an instance where ADMM converges within 32 iterations, scipy.minimize around 55 iterations. Of course scipy.minimize did it much much faster. (we did not time this though).

In all instances we've looked at, ADMM's solution is almost always extremely close to that obtained by scipy.minimize. In some cases, the solution by ADMM is smaller, so better, though not strictly feasible, but maybe the violations were insignificant. Perhaps we can be confident that ADMM give optimal solutions when there is indeed a feasible region, but we are not able to prove optimality by KKT conditions yet.

As we already know, ADMM is very sensitive to ρ.

In [5], section 4. Locally Adaptive ADMM for Deterministic Optimization, it seems that the only part that we can try on our problem is Algorithm 2. LA-ADMM. I've tried this on small-sized problems (num_species = 5, num_watersheds_total = 5) by initially setting p = p0 (0.01, 0.1, etc), then set p = 2*p after each iteration. ADMM converges after a small number of iterations, but KKT conditions are not satisfied, and the solution is way worse than that obtained by scipy.minimize. Perhaps ADMM's solution is not optimal when we set ρ this way. The other section of this paper is for stochastic optimization, which I did not look at since our problem is deterministic.

Several papers explaining how to set ρ all depend on the details of the standard form problems. Our problem is not in standard form, so it's tricky to convert those suggestions to what we should do on our problem.

Perhaps, comparing performance of ADMM and scipy by plotting their number of iterations is not an ideal choice. In the dual update step updating x, for complicated enough problems, maybe ADMM will iterate over a lot of times before finding the optimal x (at that step), leading to a large number of iterations till convergence. Maybe this is a problem of applying ADMM on our problem that we should keep in mind.

4 More experiments

Example 1. ADMM found optimal solution, but KKT conditions are not satisfied.

```
# a[i][j] = 1 if species i is available in watershed j, 0 otherwise
a = np.array([[1, 1, 0],
              [1, 1, 1],
              [1, 0, 1]])

iters = 500
```

```
Checking primal feasibility:
Total habitat vs. target habitat for species 0 :  1.5000000448355655  |  1.5
Total habitat vs. target habitat for species 1 :  3.8983998504753385  |  3.5
Total habitat vs. target habitat for species 2 :  2.9999998056430237  |  3.0

Checking complementary slackness: y*(b-Af)
[-2.30748744e-05  4.05562299e-08  1.80634991e-04]

y =  [ 5.14655590e+02 -1.01797804e-07  9.29398032e+02]
S =  [1.98012290e-08 3.98399834e-01 1.98012290e-08]
X =  [1.98012290e-08 3.63386101e+02 1.23734249e+03]

b - Af + S =  [-2.50343365e-08 -1.60727466e-08  2.14158205e-07]

Checking primal variables are positive:
Primal Variables x:  True
slack variables:  True

f'(x) * dot(a.T, y)
[2.79046039 0.99451013 1.05442069]
```

(Iterations to convergence: 370)
Comment:

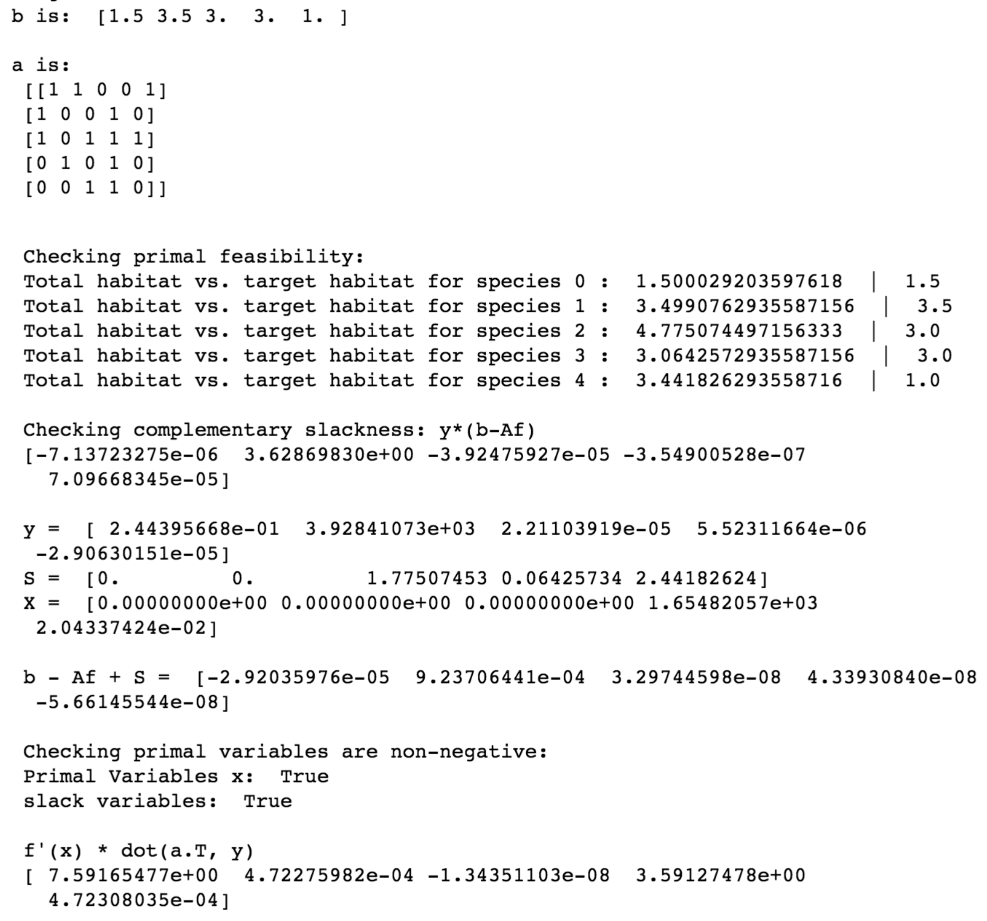
- In this case $x_2, x_3 > 0$, by complementary slackness, $w_2 = w_3 = 0$. We want to have: $f_2'(x_2)\sum_{i=1}^{3} y_i \cdot a_{i2} \approx 1$, and $f_3'(x_3)\sum_{i=1}^{3} y_i \cdot a_{i3} \approx 1$, and we actually have them.

- Besides, $x_1 \approx 0$, so $w_1 x_1 = 0$ for any value of $w_1 \leq 0$. There are no other constraints on w_1. We can set w_1 so that $1 - f_1'(x_1)\sum_{i=1}^{3} y_i \cdot a_{i1} + w_1 \approx 0$.
  However, we restrict $w_1 \leq 0$, so we can't make this expression ≈ 0.

- We have $s_2 \neq 0$. By complementary slackness, $z_2 = 0$. We therefore want $y_2 \approx 0$ to satisfy the 2nd complementary slackness, and it's true that $y_2 \approx 0$.

- We have $s_1, s_3 \approx 0$, so $z_1 s_1, z_3 s_3 = 0$, for any values $z_1, z_3 \leq 0$. There are no other constraints on z_1, z_3. We can set z_1, z_3 so that $y_1 + z_1 = y_3 + z_3 = 0$ to satisfy the 2nd complementary slackness condition.

Example 2. ADMM found optimal solution, but KKT conditions are not satisfied



```
b is:  [1.5 3.5 3.   3.   1. ]

a is:
 [[1 1 0 0 1]
 [1 0 0 1 0]
 [1 0 1 1 1]
 [0 1 0 1 0]
 [0 0 1 1 0]]


 Checking primal feasibility:
 Total habitat vs. target habitat for species 0 :  1.500029203597618  |  1.5
 Total habitat vs. target habitat for species 1 :  3.4990762935587156  |  3.5
 Total habitat vs. target habitat for species 2 :  4.775074497156333  |  3.0
 Total habitat vs. target habitat for species 3 :  3.0642572935587156  |  3.0
 Total habitat vs. target habitat for species 4 :  3.441826293558716  |  1.0

 Checking complementary slackness: y*(b-Af)
 [-7.13723275e-06  3.62869830e+00 -3.92475927e-05 -3.54900528e-07
   7.09668345e-05]

 y =  [ 2.44395668e-01  3.92841073e+03  2.21103919e-05  5.52311664e-06
  -2.90630151e-05]
 S =  [0.         0.         1.77507453 0.06425734 2.44182624]
 X =  [0.00000000e+00 0.00000000e+00 0.00000000e+00 1.65482057e+03
  2.04337424e-02]

 b - Af + S =  [-2.92035976e-05  9.23706441e-04  3.29744598e-08  4.33930840e-08
  -5.66145544e-08]

 Checking primal variables are non-negative:
 Primal Variables x:  True
 slack variables:  True

 f'(x) * dot(a.T, y)
 [ 7.59165477e+00  4.72275982e-04 -1.34351103e-08  3.59127478e+00
   4.72308035e-04]
```

(Iterations to convergence: 107. This solution should be optimal since it is almost exactly the same as the solution obtained by scipy.minimize)

Comment:

- $x_1, x_2, x_3, x_5 \approx 0$, and we need: $1 \geq f'_k(x_k) \sum_{i=1}^{5} y_i \cdot a_{ik}$, for $k = 1, 2, 3, 5$. However we do not.

- $x_4 > 0$, so we need $1 = f'_4(x_4) \sum_{i=1}^{5} y_i \cdot a_{i4}$, but we do not.

- $s_3, s_4, s_5 > 0$, so we need $y_3, y_4, y_5 \approx 0$, and it's true that $y_3, y_4, y_5 \approx 0$.

- $s_1, s_2 = 0$, so y_1, y_2 can have any values.

Example 3. ADMM did not find a feasible solution.

```
# a[i][j] = 1 if species i is available in watershed j, 0 otherwise
a = np.array([[1, 0, 0],
              [1, 1, 1],
              [1, 0, 1]])

iters = 5000
```

```
Checking primal feasibility:
Total habitat vs. target habitat for species 0 :  1.4509299999997174  |  1.5
Total habitat vs. target habitat for species 1 :  3.5000001896675115  |  3.5
Total habitat vs. target habitat for species 2 :  2.9999999033021183  |  3.0

Checking complementary slackness: y*(b-Af)
[ 9.89831630e+02 -9.42054992e-05  3.01312757e-05]

y =  [20171.82861957   496.68758997   311.60223086]
S =  [1.9801229e-08 1.9801229e-08 1.9801229e-08]
X =  [12994.99999999   165.50588444   531.14332862]

b - Af + S =  [ 4.90700198e-02 -1.69866282e-07  1.16499111e-07]

Checking primal variables are positive:
Primal Variables x:  True
slack variables:  True

f'(x) * dot(a.T, y)
[2.15897344 0.95978918 1.00000002]
```

(Convergence, iterations to optimal: 3130)
Comment:

- 1st primal feasibility does NOT hold.
- Checking complementary slackness $y * (b - Af)$ first component does NOT hold.
- We have $s_1, s_2, s_3 \approx 0$, so $z_1 s_1 = z_2 s_2 = z_3 s_3 = 0$ for any values $z_1, z_2, z_3 \leq 0$. We can set z_1, z_2, z_3 so that $y_1 + z_1 = y_2 + z_2 = y_3 + z_3 = 0$ to satisfy the 2nd complementary slackness (can we?)
- $x_1 \neq 0$, thus $w_1 = 0$ by complement slackness.
  We want to have $f_1'(x_1) \sum_{i=1}^{m} y_i \cdot a_{i1} = 1$, but we do NOT have it.
- (x_1 is suspiciously too big compared to x_2 and x_3.)

5 Conclusion & Future Work

In this project, we learned about convex optimization and the Alternating Direction Method of Multipliers. In the experiments we have done, ADMM often took much longer time to solve for the optimal solutions compared to scipy optimize, but the solutions obtained by ADMM are usually extremely closed to those by scipy. We suspect that there a few reasons for this problem. First, we did not optimize the code, which may potentially reduce time complexity. Second, the subproblem for finding optimal x in each iteration of the ADMM can be very complicated, which might require many iterations until convergence.

In futre work, we would like to optimize and profile the code to see where it is spending the most time at. We will do more experiments on large-scale problems to compare runtime of ADMM versus scipy. One possible comparison is to graph the number of iterations to convergence of both methods as we increase the problem size.

We are also interested in understanding the proof of convergence for standard ADMM, and perhaps try to extend it to non-linear constraints problems. As we know, ADMM is very sensitive to ρ, but we did not have a systematic way to set ρ for our problem, which may have significantly increased the run time. One possible future direction is to explore these questions, extend the method to non-linear problems, and apply it to more practical applications.

We thank Professor Dan Sheldon for his guidance on this project. This work would have been impossible without his guidance and encouragement.

References

[1] Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine learning, 3(1), 1-122.

[2] Boyd, Stephen, and Lieven Vandenberghe. Convex Optimization. Cambridge University Press, 2015.

[3] http://apmonitor.com/me575/index.php/Main/KuhnTucker

[4] http://adl.stanford.edu/aa222/Lecture_Notes_files/chapter5_constrainopt.pdf

[5] Xu, Yi, Mingrui Liu, Qihang Lin, and Tianbao Yang. "ADMM without a Fixed Penalty Parameter: Faster Convergence with New Adaptive Penalization." In Advances in Neural Information Processing Systems, pp. 1267-1277. 2017.

https://papers.nips.cc/paper/6726-admm-without-a-fixed-penalty-parameter-faster-convergence-with-new-adaptive-penalization.pdf