15 CVXOPT

Lab Objective: *CVXOPT is a package of Python functions and classes designed for the purpose of convex optimization. In this lab we use these tools for linear and quadratic programming. We will solve various optimization problems using CVXOPT and optimize eating healthily on a budget.*

Linear Programs

A *linear program* is a linear constrained optimization problem. Such a problem can be stated in several different forms, one of which is

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & G\mathbf{x} \preceq \mathbf{h} \\ & A\mathbf{x} = \mathbf{b}. \end{aligned}$$

The symbol $\preceq$ denotes that the components of $G\mathbf{x}$ are less than the components of $\mathbf{h}$. In other words, if $\mathbf{x} \preceq \mathbf{y}$, then $x_i < y_i$ for all $x_i \in \mathbf{x}$ and $y_i \in \mathbf{y}$.

Define vector $\mathbf{s} \succeq \mathbf{0}$ such that the constraint $G\mathbf{x} + \mathbf{s} = \mathbf{h}$. This vector is known as a *slack variable*. Since $\mathbf{s} \succeq \mathbf{0}$, the constraint $G\mathbf{x} + \mathbf{s} = \mathbf{h}$ is equivalent to $G\mathbf{x} \preceq \mathbf{h}$.

With a slack variable, a new form of the linear program is found:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & G\mathbf{x} + \mathbf{s} = \mathbf{h} \\ & A\mathbf{x} = \mathbf{b} \\ & \mathbf{s} \succeq \mathbf{0}. \end{aligned}$$

This is the formulation used by CVXOPT. It requires that the matrix A has full row rank, and that the block matrix $[G \quad A]^\mathsf{T}$ has full column rank.

Consider the following example:

$$\begin{aligned} \text{minimize} \quad & -4x_1 - 5x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 3 \\ & 2x_1 + x_2 = 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Recall that all inequalities must be less than or equal to, so that $G\mathbf{x} \preceq \mathbf{h}$. Because the final two constraints are $x_1, x_2 \geq 0$, they need to be adjusted to be $\leq$ constraints. This is easily done by multiplying by -1, resulting in the constraints $-x_1, -x_2 \leq 0$. If we define

$$G = \begin{bmatrix} 1 & 2 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad \mathbf{h} = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}, \qquad A = \begin{bmatrix} 2 & 1 \end{bmatrix}, \quad \text{and} \qquad \mathbf{b} = \begin{bmatrix} 3 \end{bmatrix}$$

then we can express the constraints compactly as

$$\begin{aligned} G\mathbf{x} &\preceq \mathbf{h}, \\ A\mathbf{x} &= \mathbf{b}, \end{aligned} \qquad \text{where} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

By adding a slack variable $\mathbf{s}$, we can write our constraints as

$$G\mathbf{x} + \mathbf{s} = \mathbf{h},$$

which matches the form discussed above.

To solve the problem using CVXOPT, initialize the arrays $\mathbf{c}$, G, $\mathbf{h}$, A, and $\mathbf{b}$ and pass them to the appropriate function. CVXOPT uses its own data type for an array or matrix. While similar to the NumPy array, it does have a few differences, especially when it comes to initialization. Below, we initialize CVXOPT matrices for $\mathbf{c}$, G, $\mathbf{h}$, A, and $\mathbf{b}$. We then use the CVXOPT function for linear programming `solvers.lp()`, which accepts $\mathbf{c}$, G, $\mathbf{h}$, A, and $\mathbf{b}$ as arguments.

```
>>> from cvxopt import matrix, solvers

>>> c = matrix([-4., -5.])
>>> G = matrix([[1., -1., 0.],[2., 0., -1.]])
>>> h = matrix([ 3., 0., 0.])
>>> A = matrix([[2.],[1.]])
>>> b = matrix([3.])

>>> sol = solvers.lp(c, G, h, A, b)
     pcost       dcost       gap    pres   dres   k/t
 0: -8.5714e+00 -1.4143e+01  4e+00  0e+00  3e-01  1e+00
 1: -8.9385e+00 -9.2036e+00  2e-01  3e-16  1e-02  3e-02
 2: -8.9994e+00 -9.0021e+00  2e-03  3e-16  1e-04  3e-04
 3: -9.0000e+00 -9.0000e+00  2e-05  1e-16  1e-06  3e-06
 4: -9.0000e+00 -9.0000e+00  2e-07  1e-16  1e-08  3e-08
Optimal solution found.
>>> print(sol['x'])
[ 1.00e+00]
[ 1.00e+00]
>>> print(sol['primal objective'])
-8.999999939019435
>>> print(type(sol['x']))
<class 'cvxopt.base.matrix'>
```

Achtung!

CVXOPT matrices only accept floats. Other data types will raise a `TypeError`.

Additionally, CVXOPT matrices are initialized column-wise rather than row-wise (as in the case of NumPy). Alternatively, we can initialize the arrays first in NumPy (a process with which you should be familiar), and then simply convert them to the CVXOPT matrix data type.

```
>>> import numpy as np

>>> c = np.array([-4., -5.])
>>> G = np.array([[1., 2.],[-1., 0.],[0., -1]])
>>> h = np.array([3., 0., 0.])
>>> A = np.array([[2., 1.]])
>>> b = np.array([3.])

# Convert the arrays to the CVXOPT matrix type.
>>> c = matrix(c)
>>> G = matrix(G)
>>> h = matrix(h)
>>> A = matrix(A)
>>> b = matrix(b)
```

In this lab we will initialize non-trivial matrices first as NumPy arrays for consistency.

Note

Although it is often helpful to see the progress of each iteration of the algorithm, you may suppress this output by first running,

```
solvers.options['show_progress'] = False
```

The function `solvers.lp()` returns a dictionary containing useful information. For now, we will only focus on the value of **x** and the primal objective value (i.e. the minimum value achieved by the objective function).

Achtung!

Note that the minimizer `x` returned by the `solvers.lp()` function is a `cvxopt.base.matrix` object. `np.ravel()` is a NumPy function that takes an object and returns its values as a flattened NumPy array. Use `np.ravel()` to return all minimizers in this lab as flattened NumPy arrays.

Problem 1. Solve the following convex optimization problem:

$$\begin{aligned} \text{minimize} \quad & 2x_1 + x_2 + 3x_3 \\ \text{subject to} \quad & x_1 + 2x_2 \geq 3 \\ & 2x_1 + 10x_2 + 3x_3 \geq 10 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_3 \geq 0 \end{aligned}$$

Return the minimizer $\mathbf{x}$ and the primal objective value.
(Hint: make the necessary adjustments so that all inequality constraints are $\leq$ rather than $\geq$).

l_1 Norm

The l_1 norm is defined

$$||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|.$$

A l_1 minimization problem is minimizing a vector's l_1 norm, while fitting certain constraints. It can be written in the following form:

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{x}\|_1 \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b}. \end{aligned}$$

This problem can be converted into a linear program by introducing an additional vector $\mathbf{u}$ of length n. Define $\mathbf{u}$ such that $|x_i| \leq u_i$. Thus, $-u_i - x_i \leq 0$ and $-u_i + x_i \leq 0$. These two inequalities can be added to the linear system as constraints. Additionally, this means that $||\mathbf{x}||_1 \leq ||\mathbf{u}||_1$. So minimizing $||\mathbf{u}||_1$ subject to the given constraints will in turn minimize $||\mathbf{x}||_1$. This can be written as follows:

$$\begin{aligned} \text{minimize} \quad & \begin{bmatrix} \mathbf{1}^\top & \mathbf{0}^\top \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} -I & I \\ -I & -I \\ -I & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} \preceq \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \\ & \begin{bmatrix} \mathbf{0} & A \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} = \mathbf{b}. \end{aligned}$$

Solving this gives values for the optimal $\mathbf{u}$ and the optimal $\mathbf{x}$, but we only care about the optimal $\mathbf{x}$.

Problem 2. Write a function called `l1Min()` that accepts a matrix A and vector $\mathbf{b}$ as NumPy arrays and solves the l_1 minimization problem. Return the minimizer $\mathbf{x}$ and the primal objective value. Remember to first discard the unnecessary u values from the minimizer.

To test your function consider the matrix A and vector $\mathbf{b}$ below.

$$A = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 0 & 3 & -2 & -1 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 7 \\ 4 \end{bmatrix}$$

The linear system $A\mathbf{x} = \mathbf{b}$ has infinitely many solutions. Use `l1Min()` to verify that the solution which minimizes $||\mathbf{x}||_1$ is approximately $\mathbf{x} = [0., 2.571, 1.857, 0.]^T$ and the minimum objective value is approximately 4.429.

The Transportation Problem

Consider the following transportation problem: A piano company needs to transport thirteen pianos from their three supply centers (denoted by 1, 2, 3) to two demand centers (4, 5). Transporting a piano from a supply center to a demand center incurs a cost, listed in Table 15.3. The company wants to minimize shipping costs for the pianos while meeting the demand.

Supply Center	Number of pianos available
1	7
2	2
3	4

Table 15.1: Number of pianos available at each supply center

Demand Center	Number of pianos needed
4	5
5	8

Table 15.2: Number of pianos needed at each demand center

Supply Center	Demand Center	Cost of transportation	Number of pianos
1	4	4	p_1
1	5	7	p_2
2	4	6	p_3
2	5	8	p_4
3	4	8	p_5
3	5	9	p_6

Table 15.3: Cost of transporting one piano from a supply center to a demand center

A system of constraints is defined for the variables p_1, p_2, p_3, p_4, p_5, and p_6, First, there cannot be a negative number of pianos so the variables must be nonnegative. Next, the Tables 15.1 and 15.2 define the following three supply constraints and two demand constraints:

$$\begin{aligned} p_1 + p_2 &= 7 \\ p_3 + p_4 &= 2 \\ p_5 + p_6 &= 4 \\ p_1 + p_3 + p_5 &= 5 \\ p_2 + p_4 + p_6 &= 8 \end{aligned}$$

The objective function is the number of pianos shipped from each location multiplied by the respective cost (found in Table 15.3):

$$4p_1 + 7p_2 + 6p_3 + 8p_4 + 8p_5 + 9p_6.$$

Note

Since our answers must be integers, in general this problem turns out to be an NP-hard problem. There is a whole field devoted to dealing with integer constraints, called *integer linear programming*, which is beyond the scope of this lab. Fortunately, we can treat this particular problem as a standard linear program and still obtain integer solutions.

Recall the variables are nonnegative, so $p_1, p_2, p_3, p_4, p_5, p_6 \geq 0$. Thus, G and $\mathbf{h}$ constrain the variables to be non-negative. Because CVXOPT uses the format $G\mathbf{x} \preceq \mathbf{h}$, we see that this inequality must be multiplied by -1. So, G must be a 6×6 identity matrix multiplied by -1, and

$\mathbf{h}$ is a column vector of zeros. Since the supply and demand constraints are equality constraints, they are A and $\mathbf{b}$. Initialize these arrays and solve the linear program by entering the code below.

```
>>> c = matrix(np.array([4., 7., 6., 8., 8., 9.]))
>>> G = matrix(-1*np.eye(6))
>>> h = matrix(np.zeros(6))
>>> A = matrix(np.array([[1.,1.,0.,0.,0.,0.],
                         [0.,0.,1.,1.,0.,0.],
                         [0.,0.,0.,0.,1.,1.],
                         [1.,0.,1.,0.,1.,0.],
                         [0.,1.,0.,1.,0.,1.]]))
>>> b = matrix(np.array([7., 2., 4., 5., 8.]))
>>> sol = solvers.lp(c, G, h, A, b)
     pcost       dcost       gap    pres   dres   k/t
 0:  8.9500e+01  8.9500e+01  2e+01  2e-16  2e-01  1e+00
 1:  8.7023e+01  8.7044e+01  3e+00  1e-15  3e-02  2e-01
Terminated (singular KKT matrix).
>>> print(sol['x'])
[ 4.31e+00]
[ 2.69e+00]
[ 3.56e-01]
[ 1.64e+00]
[ 3.34e-01]
[ 3.67e+00]
>>> print(sol['primal objective'])
87.023
```

Notice that some problems occurred. First, CVXOPT alerted us to the fact that the algorithm terminated prematurely (due to a singular matrix). Second, the minimizer and solution obtained do not consist of integer entries.

So what went wrong? Recall that the matrix A is required to have full row rank, but we can easily see that the rows of A are linearly dependent. We rectify this by converting the last row of the equality constraints into two *inequality* constraints, so that the remaining equality constraints define a new matrix A with linearly independent rows.

This is done as follows:

Suppose we have the equality constraint

$$x_1 + 2x_2 - 3x_3 = 4.$$

This is equivalent to the pair of inequality constraints

$$\begin{aligned} x_1 + 2x_2 - 3x_3 &\leq 4, \\ x_1 + 2x_2 - 3x_3 &\geq 4. \end{aligned}$$

The linear program requires only $\leq$ constraints, so we obtain the pair of constraints

$$\begin{aligned} x_1 + 2x_2 - 3x_3 &\leq 4, \\ -x_1 - 2x_2 + 3x_3 &\leq -4. \end{aligned}$$

Apply this process to the last equality constraint of the transportation problem. Then define a new matrix G with several additional rows (to account for the new inequality constraints), a new vector $\mathbf{h}$ with more entries, a smaller matrix A, and a smaller vector $\mathbf{b}$.

Problem 3. Solve the transportation problem by converting the last equality constraint into an inequality constraint. Return the minimizer $\mathbf{x}$ and the primal objective value.

Quadratic Programming

Quadratic programming is similar to linear programming, but the objective function is quadratic rather than linear. The constraints, if there are any, are still of the same form. Thus, $G, \mathbf{h}, A$, and $\mathbf{b}$ are optional. The formulation that we will use is

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\mathbf{x}^\mathsf{T} Q \mathbf{x} + \mathbf{r}^\mathsf{T}\mathbf{x} \\ \text{subject to} \quad & G\mathbf{x} \preceq \mathbf{h} \\ & A\mathbf{x} = \mathbf{b}, \end{aligned}$$

where Q is a positive semidefinite symmetric matrix. In this formulation, we require again that A has full row rank and that the block matrix $[Q \quad G \quad A]^\mathsf{T}$ has full column rank.

As an example, consider the quadratic function

$$f(x_1, x_2) = 2x_1^2 + 2x_1x_2 + x_2^2 + x_1 - x_2.$$

There are no constraints, so we only need to initialize the matrix Q and the vector $\mathbf{r}$. To find these, we first rewrite our function to match the formulation given above. If we let

$$Q = \begin{bmatrix} a & b \\ b & c \end{bmatrix}, \qquad \mathbf{r} = \begin{bmatrix} d \\ e \end{bmatrix}, \qquad \text{and} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

then

$$\begin{aligned} \frac{1}{2}\mathbf{x}^\mathsf{T} Q \mathbf{x} + \mathbf{r}^\mathsf{T}\mathbf{x} &= \frac{1}{2}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\mathsf{T} \begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} d \\ e \end{bmatrix}^\mathsf{T} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= \frac{1}{2}ax_1^2 + bx_1x_2 + \frac{1}{2}cx_2^2 + dx_1 + ex_2 \end{aligned}$$

Thus, we see that the proper values to initialize our matrix Q and vector $\mathbf{r}$ are:

$$\begin{aligned} a &= 4 & \qquad d &= 1 \\ b &= 2 & \qquad e &= -1 \\ c &= 2 \end{aligned}$$

Now that we have the matrix Q and vector $\mathbf{r}$, we are ready to use the CVXOPT function for quadratic programming `solvers.qp()`.

```
>>> Q = matrix(np.array([[4., 2.], [2., 2.]]))
>>> r = matrix([1., -1.])
>>> sol=solvers.qp(Q, r)
>>> print(sol['x'])
[-1.00e+00]
[ 1.50e+00]
>>> print sol['primal objective']
-1.25
```

Problem 4. Find the minimizer and minimum of

$$g(x_1, x_2, x_3) = \frac{3}{2}x_1^2 + 2x_1x_2 + x_1x_3 + 2x_2^2 + 2x_2x_3 + \frac{3}{2}x_3^2 + 3x_1 + x_3$$

(Hint: Write the function g to match the formulation given above before coding.)

Problem 5. The l_2 minimization problem is to

$$\begin{array}{ll} \text{minimize} & \|\mathbf{x}\|_2 \\ \text{subject to} & A\mathbf{x} = \mathbf{b}. \end{array}$$

This problem is equivalent to a quadratic program, since $\|\mathbf{x}\|_2 = \mathbf{x}^\mathsf{T}\mathbf{x}$. Write a function that accepts a matrix A and vector $\mathbf{b}$ and solves the l_2 minimization problem. Return the minimizer $\mathbf{x}$ and the primal objective value.

To test your function, use the matrix A and vector $\mathbf{b}$ from Problem 2. The minimizer is approximately $\mathbf{x} = [0.966, 2.169, 0.809, 0.888]^T$ and the minimum primal objective value is approximately 7.079.

Eating on a Budget

In 2009, the inmates of Morgan County jail convinced Judge Clemon of the Federal District Court in Birmingham to put Sheriff Barlett in jail for malnutrition. Under Alabama law, in order to encourage less spending, "the chief lawman could go light on prisoners' meals and pocket the leftover change."[1]. Sheriffs had to ensure a minimum amount of nutrition for inmates, but minimizing costs meant more money for the sheriffs themselves. Judge Clemon jailed Sheriff Barlett one night until a plan was made to use all allotted funds, 1.75 per inmate, to feed prisoners more nutritious meals. While this case made national news, the controversy of feeding prisoners in Alabama continues as of 2019[2].

[1] Nossiter, Adam, 8 Jan 2009, "As His Inmates Grew Thinner, a Sheriff's Wallet Grew Fatter", *New York Times*,https://www.nytimes.com/2009/01/09/us/09sheriff.html

[2] Sheets, Connor, 31 January 2019, "Alabama sheriffs urge lawmakers to get them out of the jail food business", https://www.al.com/news/2019/01/alabama-sheriffs-urge-lawmakers-to-get-them-out-of-the-jail-food-business.html

The problem of minimizing cost while reaching healthy nutritional requirements can be approached as a convex optimization problem. Rather than viewing this problem from the sheriff's perspective, we view it from the perspective of a college student trying to minimize food cost in order to pay for higher education, all while meeting standard nutritional guidelines.

The file `food.npy` contains a dataset with nutritional facts for 18 foods that have been eaten frequently by college students working on this text. A subset of this dataset can be found in Table 15.4, where the "Food" column contains the list of all 18 foods.

The columns of the full dataset are:

Column 1: p, price (dollars)

Column 2: s, number of servings

Column 3: c, calories per serving

Column 4: f, fat per serving (grams)

Column 5: $\hat{s}$, sugar per serving (grams)

Column 6: $\hat{c}$, calcium per serving (milligrams)

Column 7: $\hat{f}$, fiber per serving (grams)

Column 8: $\hat{p}$, protein per serving (grams)

Food	**Price** p dollars	**Serving Size** s	**Calories** c	**Fat** f g	**Sugar** $\hat{s}$ g	**Calcium** $\hat{c}$ mg	**Fiber** $\hat{f}$ g	**Protein** $\hat{p}$ g
Ramen	6.88	48	190	7	0	0	0	5
Potatoes	0.48	1	290	0.4	3.2	53.8	6.9	7.9
Milk	1.79	16	130	5	12	250	0	8
Eggs	1.32	12	70	5	0	28	0	6
Pasta	3.88	8	200	1	2	0	2	7
Frozen Pizza	2.78	5	350	11	5	150	2	14
Potato Chips	2.12	14	160	11	1	0	1	1
Frozen Broccoli	0.98	4	25	0	1	25	2	1
Carrots	0.98	2	52.5	0.3	6.1	42.2	3.6	1.2
Bananas	0.24	1	105	0.4	14.4	5.9	3.1	1.3
Tortillas	3.48	18	140	4	0	0	0	3
Cheese	1.88	8	110	8	0	191	0	6
Yogurt	3.47	5	90	0	7	190	0	17
Bread	1.28	6	120	2	2	60	0.01	4
Chicken	9.76	20	110	3	0	0	0	20
Rice	8.43	40	205	0.4	0.1	15.8	0.6	4.2
Pasta Sauce	3.57	15	60	1.5	7	20	2	2
Lettuce	1.78	6	8	0.1	0.6	15.5	1	0.6

Table 15.4: Subset of table containing food data

According to the FDA[1] and US Department of Health, someone on a 2000 calorie diet should have no more than 2000 calories, no more than 65 grams of fat, no more than 50 grams of sugar[2], at

[1] urlhttps://www.accessdata.fda.gov/scripts/InteractiveNutritionFactsLabel/pdv.html

[2] https://www.today.com/health/4-rules-added-sugars-how-calculate-your-daily-limit-t34731

least 1000 milligrams of calcium[1], at least 25 grams of fiber, and at least 46 grams of protein[2] per day.

We can rewrite this as a convex optimization problem below.

$$\begin{aligned}
\text{minimize } & \sum_{i=1}^{18} p_i x_i, \\
\text{subject to } & \sum_{i=1}^{18} c_i x_i \leq 2000, \\
& \sum_{i=1}^{18} f_i x_i \leq 65, \\
& \sum_{i=1}^{18} \hat{s}_i x_i \leq 50, \\
& \sum_{i=1}^{18} \hat{c}_i x_i \geq 1000, \\
& \sum_{i=1}^{18} \hat{f}_i x_i \geq 25, \\
& \sum_{i=1}^{18} \hat{p}_i x_i \geq 46, \\
& x_i \geq 0.
\end{aligned}$$

Problem 6. Read in the file `food.npy`. Use CVXOPT to identify how much of each food item a college student should each to minimize cost spent each day. Return the minimizing vector and the total amount of money spent.

What is the food you should eat most each day? What are the three foods you should eat most each week?

(Hint: Each nutritional value must be multiplied by the number of servings to get the nutrition value of the whole product).

You can learn more about CVXOPT at `http://cvxopt.org/index.html`.

[1] 26 Sept 2018, `https://ods.od.nih.gov/factsheets/Calcium-HealthProfessional/`

[2] `https://www.accessdata.fda.gov/scripts/InteractiveNutritionFactsLabel/protein.html`