Lab 1

CVXOPT

Lab Objective: *Introduce some of the basic optimization functions available in the CVXOPT package*

You can learn more about CVXOPT at
`http://abel.ee.ucla.edu/cvxopt/documentation/`.

Linear Programs

CVXOPT is a package of Python functions and classes designed for the purpose of convex optimization. In this lab we will focus on linear and quadratic programming. A *linear program* is a linear constrained optimization problem. Such a problem can be stated in several different forms, one of which is

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Gx + s = h \\ & Ax = b \\ & s \geq 0. \end{aligned}$$

This is the formulation used by CVXOPT. In this formulation, we require that the matrix A has full row rank, and that the block matrix $[G \quad A]^T$ has full column rank.

Note that the constraint $Gx + s = h$ includes the term s, which is not part of the objective function, and is known as the *slack variable*. Since $s \geq 0$, the constraint $Gx + s = h$ is equivalent to $Gx \leq h$.

The corresponding *dual program* for the above linear program has the form

$$\begin{aligned} \text{maximize} \quad & -h^T z - b^T y \\ \text{subject to} \quad & G^T z + A^T y + c = 0 \\ & z \geq 0. \end{aligned}$$

CVXOPT provides functions to solve both the original (*primal*) linear program and its dual program.

Consider the following example:

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

The final two constraints, $x_1, x_2 \geq 0$, 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

$$G = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}$$

and

$$h = \begin{bmatrix} 3 \\ 3 \\ 0 \\ 0 \end{bmatrix},$$

then we can express the constraints compactly as

$$Gx \leq h,$$

where

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

By adding a slack variable s, we can write our constraints as

$$Gx + s = h,$$

which matches the form discussed above. In the case of this particular example, we ignore the extra constraint

$$Ax = b,$$

since we were given no equality constraints.

Now we proceed to solve the problem using CVXOPT. We need to initialize the arrays c, G, and h, and then pass them to the appropriate function. CVXOPT uses its own data type for an array or matrix, and while similar to the NumPy array, it does have a few differences, especially when it comes to initialization. Below, we initialize CVXOPT matrices for c, G, and h.

```
>>> from cvxopt import matrix
>>> c = matrix([-4., -5.])
>>> G = matrix([[1., 2., -1., 0.],[2., 1., 0., -1.]])
>>> h = matrix([ 3., 3., 0., 0.])
```

Observe that 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.],[2., 1.],[-1., 0.],[0., -1]])
>>> h = np.array([3., 3., 0., 0.])

>>> #Now convert to CVXOPT matrix type
>>> c = matrix(c)
>>> G = matrix(G)
>>> h = matrix(h)
```

Use whichever method is most convenient. Note that we made sure the entries in the matrices are floats.

Having initialized the necessary objects, we are now ready to solve the problem. We will use the CVXOPT function `solvers.lp`, and we simply need to pass in c, G, and h as arguments.

```
>>> from cvxopt import solvers
>>> sol = solvers.lp(c, G, h)
     pcost       dcost       gap    pres   dres   k/t
 0: -8.1000e+00 -1.8300e+01  4e+00  0e+00  8e-01  1e+00
 1: -8.8055e+00 -9.4357e+00  2e-01  1e-16  4e-02  3e-02
 2: -8.9981e+00 -9.0049e+00  2e-03  1e-16  5e-04  4e-04
 3: -9.0000e+00 -9.0000e+00  2e-05  1e-16  5e-06  4e-06
 4: -9.0000e+00 -9.0000e+00  2e-07  1e-16  5e-08  4e-08
Optimal solution found.
>>> print sol['x']
[ 1.00e+00]
[ 1.00e+00]
>>> print sol['primal objective']
-8.99999981141
```

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

Problem 1. Solve the following convex optimization problem:

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

Report the values for x and the primal objective value that you obtain. Remember to make the necessary adjustments so that all inequality constraints are $\leq$ rather than $\geq$.

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 1.3. The company wants to minimize shipping costs for the pianos while meeting the demand. How many pianos should each supply center send to each demand center?

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

Table 1.1: Number of pianos available at each supply center

Demand Center	Number of pianos needed
4	5
5	8

Table 1.2: Number of pianos needed at each demand center

Supply Center	Demand Center	Cost of transportation	Number of pianos
1	4	4	p
1	5	7	q
2	4	6	r
2	5	8	s
3	4	8	t
3	5	9	u

Table 1.3: Cost of transporting one piano from a supply center to a demand center

The variables p, q, r, s, t, and u must be nonnegative and satisfy the following three supply constraints and two demand constraints:

$$\begin{aligned} p + q &= 7 \\ r + s &= 2 \\ t + u &= 4 \\ p + r + t &= 5 \\ q + s + u &= 8 \end{aligned}$$

The objective function is the number of pianos shipped from each location multiplied by the respective cost:

$$4p + 7q + 6r + 8s + 8t + 9u.$$

There a several ways to solve this linear program. We want our answers to be integers, and this added constraint in general 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.

Here, G and h constrain the variables to be non-negative. Because CVXOPT uses the format $Gx \leq h$, we see that G must be a 6×6 identity matrix multiplied by -1, and h is just a column vector of zeros. The matrices A and b represent the supply and demand constraints, since these are equality constraints. Try initializing these arrays and solving the linear program by entering the code below. (Notice that we pass more arguments to `solvers.lp` since we have equality constraints.)

```
>>> c = matrix([4., 7., 6., 8., 8., 9])
>>> G = matrix(-1*np.eye(6))
>>> h = matrix(np.zeros(6))
>>> A = matrix([[1., 0., 0., 1., 0.],
                [1., 0., 0., 0., 1.],
                [0., 1., 0., 1., 0.],
                [0., 1., 0., 0., 1.],
                [0., 0., 1., 1., 0.],
                [0., 0., 1., 0., 1.]])
>>> b = matrix([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  4e-17  2e-01  1e+00
Terminated (singular KKT matrix).
>>> print sol['x']
[ 3.00e+00]
[ 4.00e+00]
[ 5.00e-01]
[ 1.50e+00]
[ 1.50e+00]
[ 2.50e+00]
>>> print sol['primal objective']
89.5
```

Notice that some problems occurred. First, CVXOPT alerted us to the fact that the algorithm terminated prematurely (due to a singular matrix). Further, the solution that was obtained does 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 some of the equality constraints into *inequality* constraints, so that the remaining equality constraints define a new matrix A with linearly independent rows.

Rather than fuss about which equality constraints to convert into inequality constraints, let us simply convert all of the equality constraints. This is done as follows:

Suppose we have the equality constraint

$$x + 2y - 3z = 4.$$

This is equivalent to the pair of inequality constraints

$$x + 2y - 3z \leq 4,$$
$$x + 2y - 3z \geq 4.$$

Of course, we require only $\leq$ constraints, so we obtain the pair of constraints

$$x + 2y - 3z \leq 4,$$
$$-x - 2y + 3z \leq -4.$$

Apply this process to each of the equality constraints. You will obtain a new matrix G with several additional rows (to account for the new inequality constraints), and a new vector h, also with more entries. Having done this, we no longer have equality constraints A and b, so these can be ignored.

Problem 2. Solve the problem by converting all equality constraints into inequality constraints. Report the optimal values for x and the primal objective function.

Quadratic Programming

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

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^T Q x + p^T x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b, \end{aligned}$$

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 $[P \quad G \quad A]^T$ has full column rank.

As an example, let us minimize the quadratic function

$$f(x, y) = 2x^2 + 2xy + y^2 + x - y.$$

Note that there are no constraints, so we only need to initialize the matrix Q and the vector p.

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

Building the matrix Q from the function f is straightforward. The coefficients for each squared term are doubled (because in the formulation we use we are minimizing $1/2$ of the x^2 term) and then placed on the main diagonal of Q (so the term $2x^2$

yields 4 in the upper left entry of Q, and the term y^2 yields 2 in the lower right entry). The coefficient of each mixed term appears twice in the matrix (and hence does not need to be doubled), corresponding to the row and column of the two variables in the mixed term (so the term $2xy$ yields a 2 placed in the first row, second column, and a 2 in the second row, first column).

Problem 3. Find the minimizer and minimum of

$$g(x,y,z) = \frac{3}{2}x^2 + 2xy + xz + 2y^2 + 2yz + \frac{3}{2}z^2 + 3x + z$$

Allocation Models

Allocation models lead to simple linear programs. An allocation model seeks to allocate a valuable resource among competing needs. The following example is taken from "Optimization in Operations Research" by Ronald L. Rardin.

The U.S. Forest service has used an allocation model to deal with the task of managing national forests. The model begins by dividing the land into a set of analysis areas. Several land management policies (also called prescriptions) are then proposed and evaluated for each area. An *allocation* is an assignment of land (in acreage) in each analysis area to each of the prescriptions for that analysis area. We seek to find the best possible allocation, subject to forest-wide restrictions on land use.

The file `ForestData.npy` contains data for a fictional national forest (you can also find the data in Table **??**). There are 7 areas of analysis and 3 prescriptions for each of them. The first column is the area of analysis, denoted i. The second column is the size of the analysis area (in thousands of acres), denoted s_i. The third column is a prescription number, denoted j. The forth column is the net present value (NPV) per acre of all uses in area i under prescription j, and is denoted $p_{i,j}$. The fifth column is protected timber yield (in board-feet per acre) in area i under prescription j, denoted $t_{i,j}$. The sixth column is protected grazing capability (in animal-unit months per acre) for area i under prescription j, denoted $g_{i,j}$. The seventh and last column is the wilderness index rating (0 to 100) for area i under prescription j, denoted $w_{i,j}$. Let $x_{i,j}$ be the amount of land in area i allocated to prescription j.

Under this notation, an allocation is just a vector consisting of the $x_{i,j}$'s. For this particular example, the allocation vector is of size $7 \cdot 3 = 21$. Our goal is to find the allocation vector that maximizes net present value, while producing at least 40 million board-feet of timber, at least 5 thousand animal-unit months of grazing, and keeping the average wilderness index at least 70.

Of course, the allocation vector is also constrained to be nonnegative, and all of the land must be allocated precisely.

Note that since acres are in thousands, we divide the constraints of timber and

Forest Data

Analysis Area, i	Acres (1000)'s s_i	Prescription j	NPV, (per acre) $p_{i,j}$	Timber, (per acre) $t_{i,j}$	Grazing, (per acre) $g_{i,j}$	Wilderness Index, $w_{i,j}$
1	75	1	503	310	0.01	40
		2	140	50	0.04	80
		3	203	0	0	95
2	90	1	675	198	0.03	55
		2	100	46	0.06	60
		3	45	0	0	65
3	140	1	630	210	0.04	45
		2	105	57	0.07	55
		3	40	0	0	60
4	60	1	330	112	0.01	30
		2	40	30	0.02	35
		3	295	0	0	90
5	212	1	105	40	0.05	60
		2	460	32	0.08	60
		3	120	0	0	70
6	98	1	490	105	0.02	35
		2	55	25	0.03	50
		3	180	0	0	75
7	113	1	705	213	0.02	40
		2	60	40	0.04	45
		3	400	0	0	95

animal months of grazing by 1000. We can summarize our problem as follows:

$$
\begin{aligned}
\text{maximize } & \sum_{i=1}^{7}\sum_{j=1}^{3} p_{i,j}x_{i,j} \\
\text{subject to } & \sum_{j=1}^{3} x_{i,j} = s_i \text{ for } i = 1,..,7 \\
& \sum_{i=1}^{7}\sum_{j=1}^{3} t_{i,j}x_{i,j} \geq 40,000 \\
& \sum_{i=1}^{7}\sum_{j=1}^{3} g_{i,j}x_{i,j} \geq 5 \\
& \frac{1}{788}\sum_{i=1}^{7}\sum_{j=1}^{3} w_{i,j}x_{i,j} \geq 70 \\
& x_{i,j} \geq 0 \text{ for } i = 1,...,7 \text{ and } j = 1,2,3
\end{aligned}
$$

Problem 4. Solve the problem above. Output the value of each $x_{i,j}$ and the maximum total net present value (the primal objective of the appropriately minimized linear function multiplied by -1000).