# cvxpy_intro

January 28, 2017

## 1 Introduction to CVXPY

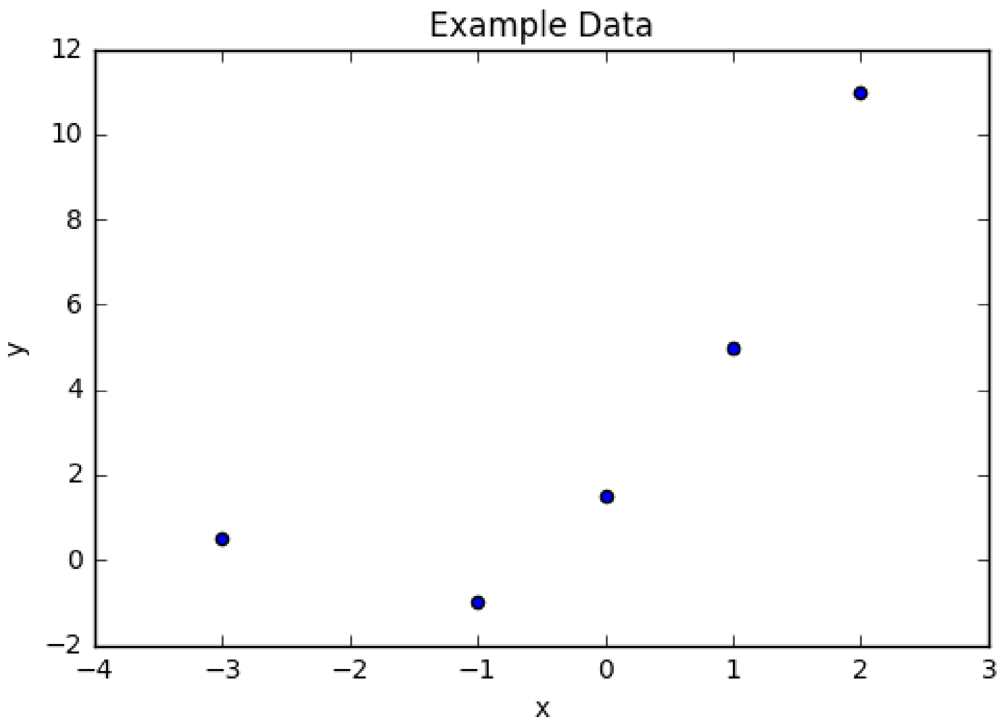
CVXPY is a Python-embedded modeling language for (disciplined) convex optimization problems. Much like CVX in MATLAB, it allows you to express the problem in a natural way that follows the math, instead of expressing the problem in a way that conforms to a specific solver's syntax.

CVXPY Homepage
CVXPY Tutorial Documentation
CVXPY Examples

```
In [1]: import numpy as np # we can use np.array to specify problem data
        import matplotlib.pyplot as plt
        %matplotlib inline
        import cvxpy as cvx
```

### 1.1 Example: Least-Squares Curve Fitting

```
In [2]: x = np.array([-3,  -1, 0,   1, 2])
        y = np.array([0.5, -1, 1.5, 5, 11])
        plt.scatter(x,y)
        plt.xlabel('x'); plt.ylabel('y'); plt.title('Example Data')
        plt.show()
```




The data look like they follow a quadratic function. We can set up the following Vandermonde system and use unconstrained least-squares to estimate parameters for a quadratic function.

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \end{bmatrix}$$

Solving the following least-squares problem for $\beta$ will give us parameters for a quadratic model:

$$\min_{\beta} \|A\beta - y\|_2$$

Note that we could easily solve this simple problem with a QR factorization ( in MATLAB, np.linalg.lstsq).

```
In [3]: A = np.column_stack((np.ones(5,), x, x**2))

        # now setup and solve with CVXPY
        beta = cvx.Variable(3)

        # CVXPY's norm behaves like np.linalg.norm
        obj = cvx.Minimize(cvx.norm(A*beta-y))
        prob = cvx.Problem(obj)
```

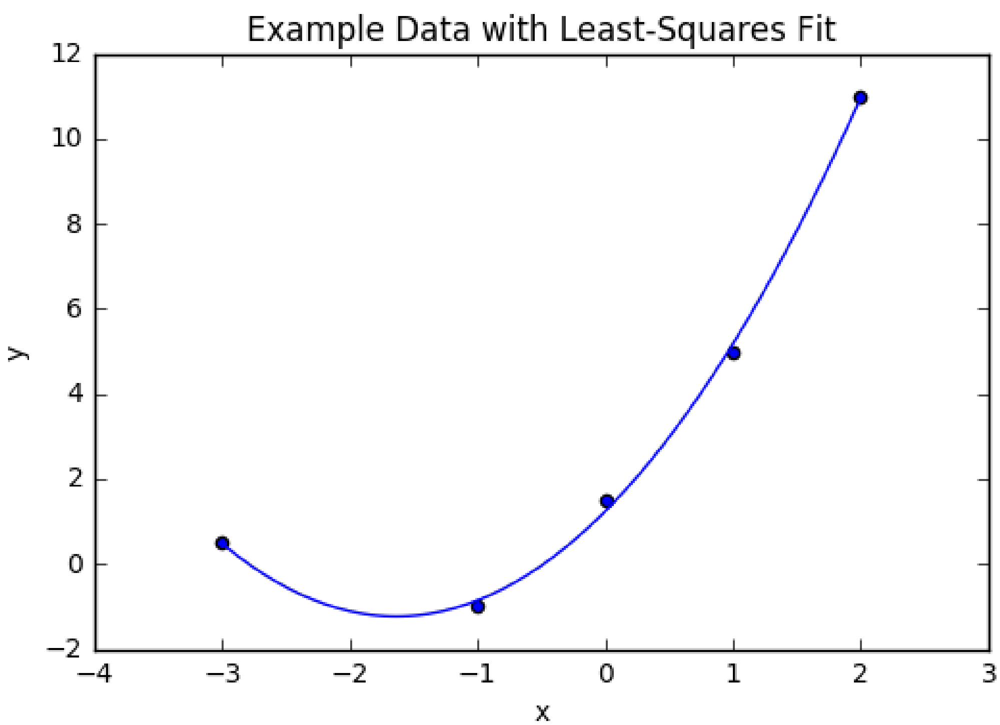
```
# Assuming the problem follows the DCP ruleset,
# CVXPY will select a solver and try to solve the problem.
# We can check if the problem is a disciplined convex program
# with prob.is_dcp().
prob.solve()

print("Problem status: ", prob.status)
print("Optimal value:  ", prob.value)
print("Optimal var:\n", beta.value)
```

```
Problem status:  optimal
Optimal value:   0.3496263519482917
Optimal var:
 [[ 1.23858616]
 [ 3.01656848]
 [ 0.92157585]]
```

Let's check the solution to see how we did:

```
In [4]: _beta = beta.value # get the optimal vars
        _x = np.linspace(x.min(), x.max(), 100)
        _y = _beta[0,0]*np.ones_like(_x) + _beta[1,0]*_x + _beta[2,0]*_x**2
        plt.scatter(x,y)
        plt.plot(_x,_y,'-b')
        plt.xlabel('x'); plt.ylabel('y'); plt.title('Example Data with Least-Squares Fit')
        plt.show()
```



### 1.2 Example: $\ell_1$-norm minimization

Consider the basis pursuit problem

$$\begin{array}{ll} \text{minimize} & \|x\|_1 \\ \text{subject to} & Ax = y. \end{array}$$

This is a least $\ell_1$-norm problem that will hopefully yield a sparse solution $x$.
We now have an objective, $\|x\|_1$, and an equality constraint $Ax = y$.

In [5]:
```
# make a bogus sparse solution and RHS
m = 200; n = 100;
A = np.random.randn(m,n)
_x = np.zeros((n,1))
_k = 10
_I = np.random.permutation(n)[0:_k]
_x[_I] = np.random.randn(_k,1)
y = np.dot(A,_x)

x = cvx.Variable(n)

# Even though the cvx.norm function behaves very similarly to
# the np.linalg.norm function, we CANNOT use the np.linalg.norm
# function on CVXPY objects.  If we do, we'll probably get a strange
# error message.
obj = cvx.Minimize(cvx.norm(x,1))

# specify a list of constraints
constraints = [ A*x == y ]

# specify and solve the problem
prob = cvx.Problem(obj, constraints)
prob.solve(verbose=True) # let's see the underlying solver's output

print("Problem status: ", prob.status)
print("Optimal value:  ", prob.value)

print("True nonzero inds:      ", sorted(_I))
print("Recovered nonzero inds: ", sorted(np.where(abs(x.value) > 1e-14)[0]))
```

```
ECOS 2.0.4 - (C) embotech GmbH, Zurich Switzerland, 2012-15. Web: www.embotech.com/ECOS

It     pcost       dcost      gap   pres   dres    k/t    mu     step   sigma     IR    |   BT
 0  +0.000e+00  -0.000e+00  +4e+02  9e-01  1e-02  1e+00  2e+00    ---    ---     1  1  - |  -  -
 1  +4.162e+00  +4.219e+00  +5e+01  4e-01  2e-03  2e-01  3e-01  0.9182  8e-02   1  0  0 |  0  0
```

```
 2  +6.690e+00  +6.699e+00  +7e+00  7e-02  2e-04  3e-02  4e-02  0.8962  3e-02   1  0  0 |  0  0
 3  +7.018e+00  +7.020e+00  +1e+00  1e-02  3e-05  4e-03  5e-03  0.8622  8e-03   1  0  0 |  0  0
 4  +7.080e+00  +7.080e+00  +2e-02  2e-04  8e-07  1e-04  1e-04  0.9890  1e-02   1  0  0 |  0  0
 5  +7.081e+00  +7.081e+00  +3e-04  3e-06  8e-09  1e-06  1e-06  0.9890  1e-04   1  0  0 |  0  0
 6  +7.081e+00  +7.081e+00  +3e-06  3e-08  9e-11  1e-08  2e-08  0.9890  1e-04   1  0  0 |  0  0
 7  +7.081e+00  +7.081e+00  +3e-08  3e-10  1e-12  2e-10  2e-10  0.9890  1e-04   1  0  0 |  0  0

OPTIMAL (within feastol=3.4e-10, reltol=4.8e-09, abstol=3.4e-08).
Runtime: 0.037927 seconds.

Problem status:  optimal
Optimal value:   7.081145255502928
True nonzero inds:       [6, 10, 21, 39, 41, 43, 57, 59, 61, 69]
Recovered nonzero inds:  [6, 10, 21, 39, 41, 43, 57, 59, 61, 69]
```

### 1.3 Example: Relaxation of Boolean LP

Consider the Boolean linear program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \\ & x_i \in \{0,1\}, \quad i = 1, ..., n. \end{array}$$

Note: the generalized inequality $\preceq$ is just element-wise $\leq$ on vectors.

This is not a convex problem, but we can relax it to a linear program and hope that a solution to the relaxed, convex problem is "close" to a solution to the original Boolean LP. A relaxation of the Boolean LP is the following LP:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \\ & \mathbf{0} \preceq x \preceq \mathbf{1}. \end{array}$$

The relaxed solution $x^{\text{rlx}}$ can be used to guess a Boolean point $\hat{x}$ by rounding based on a threshold $t \in [0,1]$:

$$\hat{x}_i = \begin{cases} 1 & x_i^{\text{rlx}} \geq t \\ 0 & \text{otherwise,} \end{cases}$$

for $i = 1, ..., n$. However, the Boolean point $\hat{x}$ might not satisfy $Ax \preceq b$ (i.e., $\hat{x}$ might be infeasible).

From Boyd and Vandenberghe: > You can think of $x_i$ as a job we either accept or decline, and $-c_i$ as the (positive) revenue we generate if we accept job $i$. We can think of $Ax \preceq b$ as a set of limits on $m$ resources. $A_{ij}$, which is positive, is the amount of resource $i$ consumed if we accept job $j$; $b_i$, which is positive, is the amount of recourse $i$ available.

```
In [6]: m = 300; n = 100;
        A = np.random.rand(m,n)
        b = A.dot(np.ones((n,1)))/2.
        c = -np.random.rand(n,1)
```

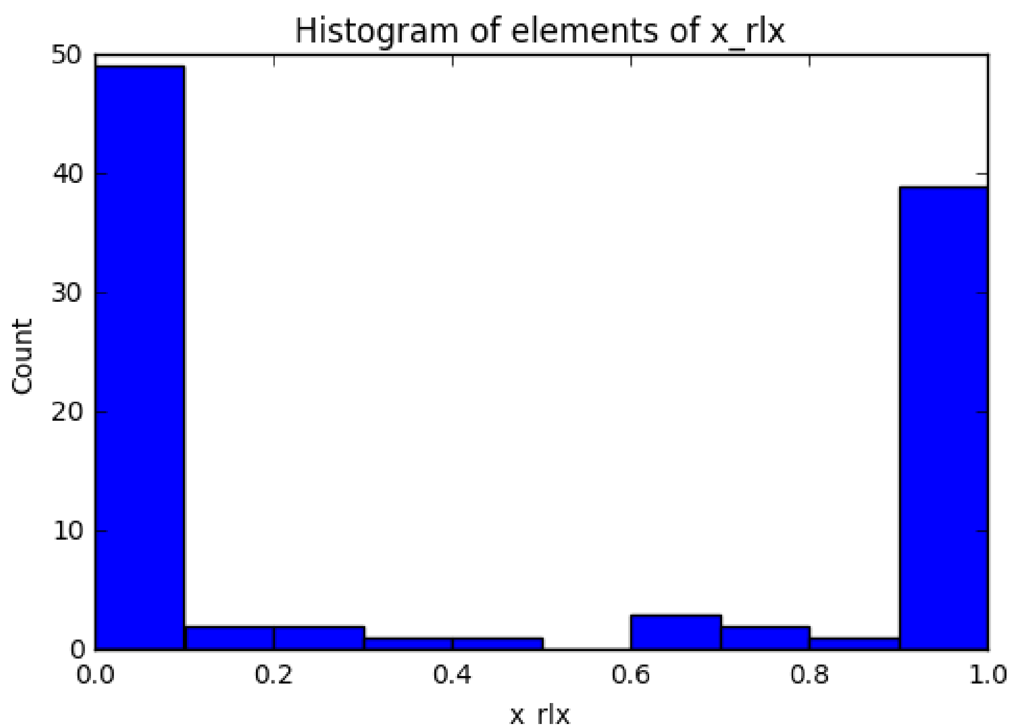
```
x_rlx = cvx.Variable(n)
obj = cvx.Minimize(c.T*x_rlx)
constraints = [ A*x_rlx <= b,
                0 <= x_rlx,
                x_rlx <= 1 ]

prob = cvx.Problem(obj, constraints)
prob.solve()

print("Problem status: ", prob.status)
print("Optimal value:  ", prob.value)

plt.hist(x_rlx.value)
plt.xlabel('x_rlx'); plt.ylabel('Count')
plt.title('Histogram of elements of x_rlx')
plt.show()
```

```
Problem status:  optimal
Optimal value:   -34.08078116975795
```



### 1.4 Example: Minimum Volume Ellipsoid

Sometimes an example is particularly hard and we might need to adjust solver options, or use a different solver.

Consider the problem of finding the minimum volume ellipsoid (described by the matrix $A$ and vector $b$) that covers a finite set of points $\{x_i\}_{i=1}^n$ in $\mathbb{R}^2$. The MVE can be found by solving

$$\begin{array}{ll} \text{maximize} & \log(\det(A)) \\ \text{subject to} & \|Ax_i + b\| \leq 1, \quad i = 1, ..., n. \end{array}$$

To allow CVXPY to see that the problem conforms to the DCP ruleset, we should use the function `cvx.log_det(A)` instead of something like `log(det(A))`.

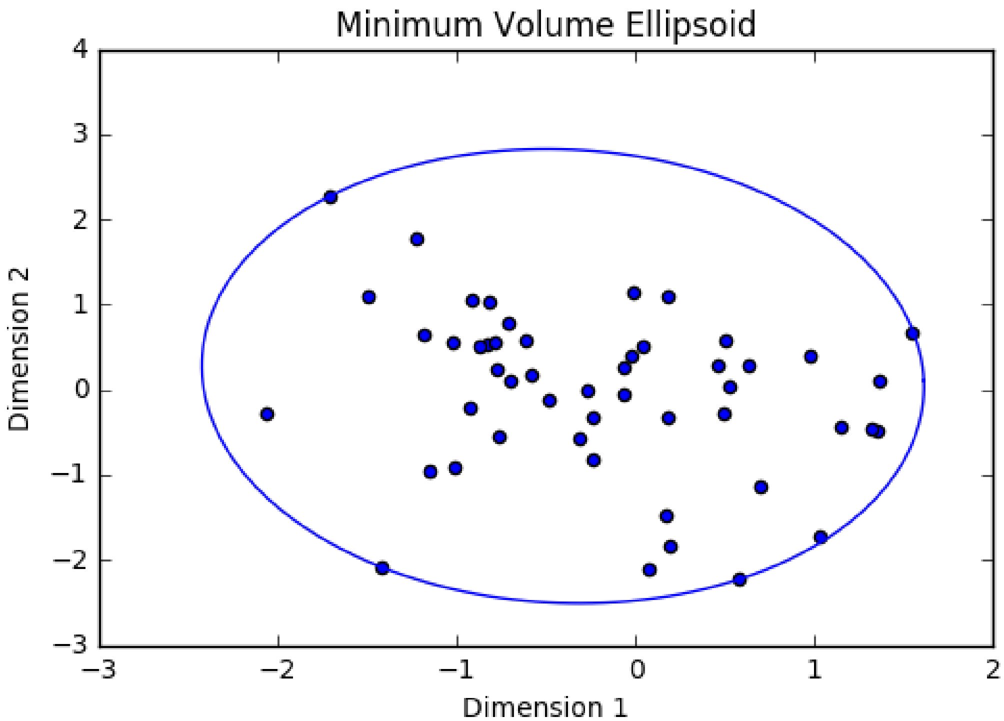
In [7]:
```
# Generate some data
np.random.seed(271828) # solver='CVXOPT' reaches max_iters
m = 2; n = 50
x = np.random.randn(m,n)

A = cvx.Variable(2,2)
b = cvx.Variable(2)
obj = cvx.Maximize(cvx.log_det(A))
constraints = [ cvx.norm(A*x[:,i] + b) <= 1 for i in range(n) ]

prob = cvx.Problem(obj, constraints)
#prob.solve(solver='CVXOPT', verbose=True) # progress stalls
#prob.solve(solver='CVXOPT', kktsolver='robust', verbose=True) # progress still stalls
prob.solve(solver='SCS', verbose=False) # seems to work, although it's not super accurat

# plot the ellipse and data
angles = np.linspace(0, 2*np.pi, 200)
rhs = np.row_stack((np.cos(angles) - b.value[0], np.sin(angles) - b.value[1]))
ellipse = np.linalg.solve(A.value, rhs)

plt.scatter(x[0,:], x[1,:])
plt.plot(ellipse[0,:].T, ellipse[1,:].T)
plt.xlabel('Dimension 1'); plt.ylabel('Dimension 2')
plt.title('Minimum Volume Ellipsoid')
plt.show()
```


Minimum Volume Ellipsoid
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
Dimension 1
Dimension 2

In [ ]: