University of California, Berkeley

HW 11

Oscar Ortega

April 26, 2019

1 A Portfolio Design Problem

1.

$$\mathbb{P}(y \leq q) \leq \epsilon \tag{1.1}$$
$$\Phi(q) \leq \epsilon \tag{1.2}$$
$$q \leq \Phi^{-1}(\epsilon) \tag{1.3}$$

2.

$$r^T x = \mathcal{N}(r^T x, x^T \Sigma x) \tag{1.4}$$
$$r^T x - \hat{r}^T x = \mathcal{N}(0, x^T \Sigma x) \tag{1.5}$$
$$= \mathcal{N}(0, \|\Sigma^{1/2} x\|_2^2) \tag{1.6}$$
$$\frac{r^T x - \hat{r}^T x}{\|\Sigma^{1/2} x\|_2} = \mathcal{N}(0,1) \tag{1.7}$$
$$\mathbb{P}\left(\frac{r^T x - \hat{r}^T x}{\|\Sigma^{1/2} x\|_2} \leq \frac{q - \hat{r}^T x}{\|\Sigma^{1/2} x\|_2)}\right) \leq \epsilon \tag{1.8}$$
$$\Phi\left(\frac{q - \hat{r}^T x}{\|\Sigma^{1/2} x\|_2}\right) \leq \epsilon \tag{1.9}$$
$$\frac{q - \hat{r}^T x}{\|\Sigma^{1/2} x\|_2} \leq \Phi^{-1}(\epsilon) \tag{1.10}$$
$$q - \hat{r}^T x \leq \Phi^{-1}(\epsilon) \|\Sigma^{1/2} x\|_2 \tag{1.11}$$

Where $q = -0.01$, and $\epsilon = 10^{-4}$

3. c: Jupyter notebook
4. d: Jupyter notebook
5. e: Jupyter notebook

2 Robust Linear Programming

1. Consider the following optimization problem:

$$\min_{y \in \mathbf{R}^n} -x^T y$$

$$\text{s.t } \|y\|_\infty \leq 1$$

We know the problem is convex and can furthermore be reformulated as follows:

$$\min_{y \in \mathbf{R}^n} -x^T y$$

$$\text{s.t } y_i \leq 1 : \forall i$$

$$y_i \geq -1 \forall i$$

We know that because this problem is convex, it will be optimized by activating the constraints so $y_i \in \{-1, 1\} : \forall i$ If we let $y_i = 1$ if the sign of x_i is positive and let $y_i = -1$ if the sign of x_i is negative $x^T y = \|x\|_1$

2. Consider the inequality $\tilde{a}_i^T x + v^T x \leq b_i : \forall v \in \{-\rho, \rho\}^n : i = 1, ..., n$ And consider an x that satisfies the following inequality.

$$\tilde{a}_i^T x + v^T x \leq b_i : \forall v \in \{-\rho, \rho\}^n : i = 1, ..., n \tag{2.1}$$
$$\tilde{a}_i^T x + \rho v^T x \leq b_i : \forall v \in \{-1, 1\}^n : i = 1, ..., n \tag{2.2}$$
$$\tilde{a}_i^T x + \rho v^T x \leq b_i : \forall v \in \{-1, 1\}^n : i = 1, ..., n \tag{2.3}$$
$$\tilde{a}_i^T x + \rho v^T x \leq b_i : \forall v : \|v\|_\infty \leq 1 \tag{2.4}$$
$$\tilde{a}_i^T x + \rho(\max_v v^T x) \leq b_i : v : \|v\|_\infty \leq 1 \tag{2.5}$$
$$\tilde{a}_i^T x + \rho \|x\|_1 \leq b_i \tag{2.6}$$

3. We can reformulate the problem presented as the following convex program.

$$\min_x c^T x$$

$$\text{s.t } \tilde{a}_i^T x + \rho \|x\|_1 \leq b_i : i = 1, ..., m$$

Which can be reconstructed into the following linear program:

$$\min_x c^T x$$

$$\text{s.t } (1 - a_j) x_j + b_i \leq 0 : j = 1, ..., n; i = 1, ..m$$

$$(1 + a_j) x_j - b_i \leq 0 : j = 1, ..., n; i = 1, ..m$$

3 Robust Machine Learning

1. Using the hinge loss as opposed to the 0-1 loss guarantees the finding a local minimum to this problem allows us to find a global minimum to this problem as the hinge loss function is convex.
   Having the hinge loss equal to 0 implies that $1 \leq y_i(x_i^T w + v) : i = 1, ..., m$. This tells us that all the labels were predicted correctly.

2. Let $\bar{x}_i = \begin{bmatrix} x_i & 1 \end{bmatrix}^T$ and let $\bar{w} = \begin{bmatrix} w & v \end{bmatrix}$ In this case, we know that if $\bar{w}$ is feasible than this implies that w, v are feasible as well. This means we can reformulate this optimization problem as follows.

$$\min_{\bar{w}} \max_{\|\bar{w}-w\|_\infty \leq \epsilon} \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i(\bar{x}_i^T \bar{w}))$$

3. To obtain a low precision classifier for this problem, we can go ahead and choose any $\tilde{w}$ that is within the bounds of our tolerance. In other words, if $\|\tilde{w} - w^*\|_\infty \leq \epsilon$, we can guarantee that we are within the bounds of our robustness.

4. Applying the same set of reasoning in part 2b to this problem, we know the following:

$$\max_{w:\|w\|_\infty \leq \epsilon} w^T x = \epsilon \|x\|_1$$

   Therefore, we can reframe the optimization problem as the following:

$$\min_w \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i(x_i^T w + \epsilon \|x_i\|_1)$$

5. Assuming a normalized dataset:

$$\min_w \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i(x_i^T w) + \epsilon \|x_i\|_1)$$

$$= \min_w \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i(x_i^T w) + \epsilon)$$

   Here, if we let $\bar{x}_i = \begin{bmatrix} x_i & 1 \end{bmatrix}^T$, $\bar{w} = \begin{bmatrix} w & \epsilon \end{bmatrix}$ and let $\bar{y} = \begin{bmatrix} y & 0 \end{bmatrix}$ and we can further transform the problem to the following.

$$\min_w \frac{1}{m} \sum_{i=1}^{m} \max(0, 1 - y_i(x_i^T w))$$

   Because this problem is in the same form that was shown in part b, we can then reverse the changes to obtain the form in equation 4.

4 Convexity of functions and KKT

1. We know that $\sigma_{\max}(X) = \max_{u:\|u\|_2 \leq 1} \|Xu\|_2 = \|X\|_2$. We know norms are convex functions, therefore the function is convex.

2. This set is also convex
   Let $\gamma \in [0,1], X_1, X_2 \in C$

$$\begin{aligned}
\lambda_{\min}(\gamma X_1 + (1-\gamma)X_2) &= \min_{z:\|z\|_2 \leq 1} z^T(\gamma X_1(1-\gamma)X_2)z && (4.1)\\
&\geq \min_{z:\|z\|_2 \leq 1} \gamma z^T X_1 (1-\gamma) z + \min_{z:\|z\|_2 \leq 1} (1-\gamma) z^T X_2 z && (4.2)\\
&\geq \gamma 2 + (1-\gamma)2 && (4.3)\\
&\geq 2 && (4.4)
\end{aligned}$$

   Therefore this set is convex.

3. This statement is true:

$$f_o(\tilde{x}) \geq \inf f_0(x) = p^* \geq d^* = \sup g(\lambda) \geq g(\tilde{\lambda})$$

   The two statements imply strong duality holds.

4. This statement is false, for one we do not know anything about the constraints being convex, so strong duality does not hold. Furthermore, because we don't know if the constraints are affine, we cannot say slater's condition is satisfied and cannot say strong duality holds.

portfolio_opt

April 25, 2019

```
In [220]: import cvxpy as cp
          import numpy as np
          from scipy.linalg import sqrtm
          from scipy.stats import norm as Normal_Dist
          import matplotlib.pyplot as plt
```

1 Part C

```
In [221]: # Create our r_hat, Sigma, and other things required as outlined in the problem
          ones = np.ones(4)
          r_hat = np.array([0.12, 0.10, 0.07, 0.03])
          Sigma = np.array([
              [0.0064, 0.0008, -0.0011, 0.0],
              [0.0008, 0.0025, 0.0, 0.0],
              [-0.0011, 0.0, 0.0004, 0.0],
              [0.0, 0.0, 0.0, 0.0]
          ])
          Sigma_root = sqrtm(Sigma)# todo, find the square root of the Sigma matrix

In [222]: # Here, we form the constraints and solve the problem within the function (so it is
          # later on)
          def optimization(epsilon, r_hat, Sigma_root):

              val_opt, soln_opt = None, None
              x = cp.Variable(4)
              forty = np.array([0.4, 0.4, 0.4, 0.4])
              five = np.array([0.05, 0.05, 0.05, 0.05])
              ############## todo: your beautiful code starts here ##############
              constraints = [np.sum(x) <= 1,
                             x[3] <= 0.2,
                             x >= five,
                             x <= forty,
                  cp.norm(Sigma_root@x) <= Normal_Dist().cdf(epsilon) * (r_hat.T@x + 0.01)
              ]
              p = cp.Problem(cp.Maximize(r_hat.T@x), constraints)
              p.solve(gp = False)
              val_opt, soln_opt = p.value, x.value
```

```
        #print(p.value)
        ############## todo: your beautiful code ends here ###############
        return val_opt, soln_opt #Make sure you return numpy arrays here and not cvxpy v
```

In [223]:
```
opt_val, opt_soln = optimization(1e-4, r_hat, Sigma_root )
print("Optimal value:", opt_val)
print("Optimal soln:", opt_soln)
```

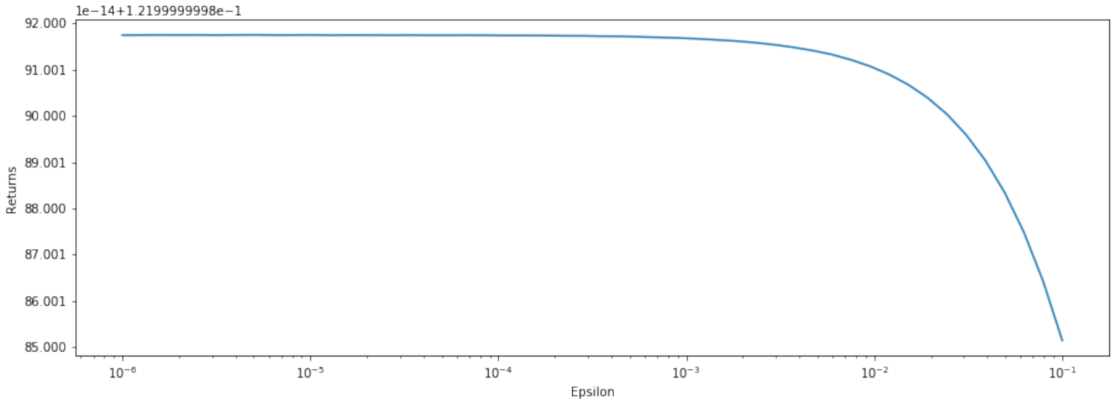
```
Optimal value: 0.12199999998491738
Optimal soln: [0.4 0.4 0.4 0.2]
```

2 Part D

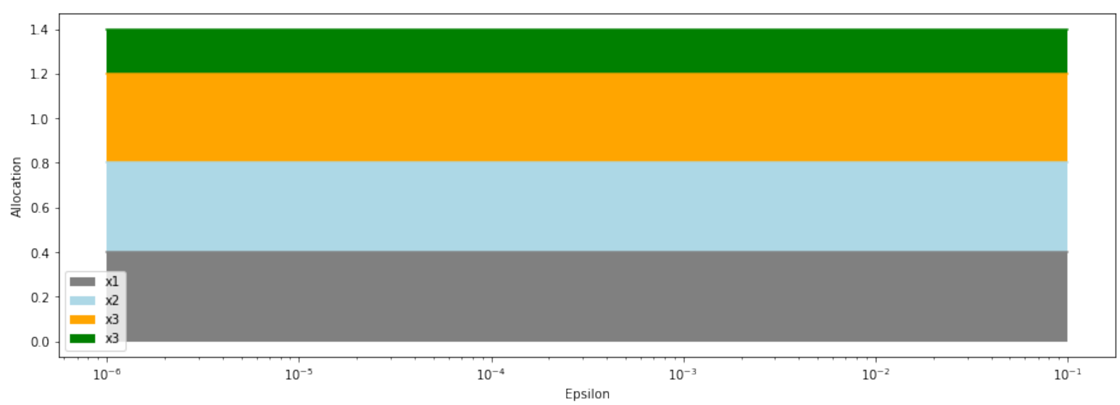
In [224]:
```
# epsilons generated between 1e-6 and 1e-1 in log scale
epsilons = np.logspace(-6,-1)
```

In [225]:
```
rhat_vals = [] # optimal return values (i.e. the prob.value outputs)
opt_x_vals = [] # optimal portfolio allocations

#going over the values of epsilon
for eps in epsilons:
    ret, x = optimization(eps, r_hat, Sigma_root)
    rhat_vals.append(ret)
    opt_x_vals.append(x)
```

In [226]:
```
# Plotting returns
plt.figure(figsize=(15,5))
plt.plot(epsilons, rhat_vals)
plt.xlabel('Epsilon')
plt.ylabel('Returns')
plt.xscale('log')

plt.savefig('partb_rhats_2.png')
```



```
In [227]: # Plotting allocations
          plt.figure(figsize=(15, 5))
          y1 = np.array([item[0] for item in opt_x_vals])
          plt.plot(epsilons, y1, c = 'gray')
          plt.fill_between(epsilons, 0, y1, facecolor = 'gray', label = 'x1')

          y2= np.array([item[1] for item in opt_x_vals])
          plt.plot(epsilons, y1+y2, c= 'lightblue')
          plt.fill_between(epsilons, y1, y1+y2, facecolor = 'lightblue', label = 'x2')

          y3= np.array([item[2] for item in opt_x_vals])
          plt.plot(epsilons, y1 + y2+y3, c= 'orange')
          plt.fill_between(epsilons, y1+y2, y1+y2+y3, facecolor = 'orange', label = 'x3')



          y4= np.array([item[3] for item in opt_x_vals])
          plt.plot(epsilons, y1 + y2+y3+y4, c= 'green')
          plt.fill_between(epsilons, y1+y2+y3, y1+y2+y3+y4, facecolor = 'green', label = 'x3')

          plt.legend(loc = 'best')
          plt.xlabel('Epsilon')
          plt.ylabel('Allocation')
          plt.xscale('log')
          plt.savefig('area_plot_ptfs_2.png')
```

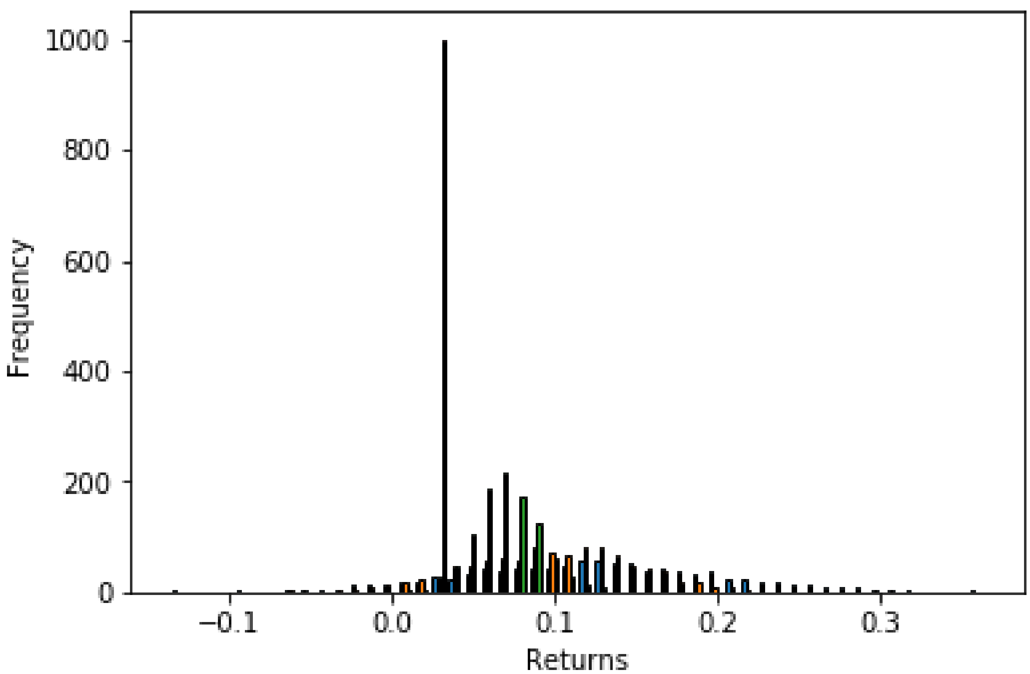


3 Part E

```
In [228]: # 3rd part: Just regenerating the optimal x again with 10e-4 as our epsilon
          optimal_x_part_1 = optimization(1e-4, r_hat, Sigma_root)[1]


          optimal_x_part_1 = np.array(optimal_x_part_1)
```

```
In [229]: returns =  np.random.multivariate_normal(mean = r_hat,
                                                   cov = Sigma,
                                                   size = 1000)# todo: generate 10000 random s


          np.random.seed(777)
```

```
In [230]: # Plot the histogram of random returns
          plt.hist(returns, 50, ec='black')
          plt.xlabel('Returns')
          plt.ylabel('Frequency')
          plt.savefig('monte_carlo.png')
```




```
In [231]: mean = np.sum(returns @ optimal_x_part_1, axis = 0)
          print("Mean of the returns:", mean)
```

```
Mean of the returns: 123.09493531698021
```

```
In [232]: pct = len(np.where(returns @ optimal_x_part_1 <= 0)) / 1000
          print("Percentage of loss:", pct)
```

```
Percentage of loss: 0.001
```