Markowitz Portfolio with Integer Constraints

2015122029 Sun-Woo Lim

1. Problem Definition

Original Problem

This problem deals with a verified version of Markowitz Portfolio (Markowitz, H, M (1959)) Optimization problem and its generalization into proportion allocation problems. According to the google dictionary, portfolio is a range of investments held by a person or organization. Forming a portfolio, a user wants to a) maximize the expected profit but also b) does not want the risk to be too high. These two main goals are contradictory because of the "high risk, high rewards" principle. Harry Max Markowitz, an American economist and the 1990 Nobel prize in Economic Sciences nominee is most famous for his portfolio model (Markowitz, H, M (1959)) that seeks to find a balance between these two goals.

Markowitz portfolio optimizes the proportion vector x where each entry x_i denotes the proportion of item i of in a portfolio (e.g., Apple stock: 0.2, US 10 year Bond: 0.5, Alphabet A stock: 0.3). Let n be the number of items in a portfolio so that $x \in R^n$. Let $\vec{r} \in R^n$ be the random vector of returns. We are interested in summary statistics of $\vec{r}$: a) $\vec{e} = E(\vec{r}) \in R^n$ (expected return) and b) $C = E(\vec{r} - \vec{e})(\vec{r} - \vec{e})^T \in R^{n \times n}$(covariance matrix of $\vec{r}$). The original problem is a constrained Quadratic Programming (QP) formulized as the following:

$$\max_{x \in R^n}(e^T x - \gamma x^T C\, x)\, s.t. \sum_i x_i = 1, x_i \geq 0, \forall i \ldots (1).$$

The objective function in (1) finds a balance between two goals I mentioned: make the expected profit $e^T x$ high and keep the risk $\gamma x^T C\, x$ low. $\gamma > 0$ is called as the "risk sensitivity parameter" representing how risk averse the user is. For brevity, I call the objective function value as "utility". Two constraints are properties of proportions: nonnegativity and summing to 1. There are two additional details for the reality of the model. First, $\vec{e}$ and C are unknown so that we use the point estimates of them in our model. Also, in representing $\vec{r} \in R^n$, I need to standardize the price of each item (e.g, 1000\$) because different items have different prices. Thus, I would compare 1000\$ amount of apple stock and 1000\$ amount of Disney stock for instance.

How I modified the original problem

Instead of solving the original problem, I simplify it by rounding up the x_i values to the k digits after the decimal point. This means that even if the optimal x_i is an infinite decimal fraction like 1/3, I represent this in 0.33 when k = 2. Thus, I solve an approximate solution for x and the bigger k is, the closer the obtained answer is to the true solution. The modified version of the problem is as follows:

$$\max_{x \in R^n}(e^T x - \gamma x^T C\, x)\, s.t. \sum_i x_i = 1\,, x_i = \frac{z}{10^k}, z \in N \cup \{0\}, k \in N \ldots (2).$$

The biggest reason I modified the problem is that algorithms for the original problem include integer point methods, extensions of simplex algorithms that are quite far from in class concepts. Also, solving the original QP requires thorough understanding of optimization, which is challenging. Moreover, the original problem is impossible to solve even by brute force. My modification is justifiable because a) people only need finite decimal accuracy for fractions and b) even computers need to approximate an infinite decimal fraction into a finite number because of the limited memory.

Necessity of Efficient Algorithm

Efficient algorithm for this problem is required because there are numerous options to choose from (n is large), even restricting my attention to stock items. More importantly, since return of investment items are random, there is possibility that inference of e, C change over time. Thus, algorithms should quickly adjust to new data.

2. Two Ways of Data Generation

Before addressing algorithms for this problem, I need to generate data of $\vec{e}$ and C. For $\vec{e}$, since I can either gain or lose from investment, all entries of $\vec{e}$ can be either positive or negative. As I know of, there are more items having positive overall expected return than negative return. Thus, I randomly sample e_i from $U(-0.5, 1.5)$.

I sample C_{ii}, the diagonal elements of C from $U(0,1)$ because the variance of each return $\vec{r}_i$ need to be nonnegative. To add more reality of high risk, high return, I first planned to set $C_{ii} = f(e_i) + Z, Z \sim N(0,1)$ but decided not to because it is hard to think of a plausible increasing function f.

For non-diagonal elements $C_{ij} = C_{ji} = Cov(r_i, r_j) = E[(r_i - e_i)(r_j - e_j)], i \neq j$, there are two options. First is to make them all zero, assuming that returns of two items are uncorrelated. With that assumption, the objective function is $\sum_i (e_i x_i - \gamma C_{ii} x_i^2)$. Second option is assuming covariance and sampling them from $U(-1,1)$. In this general case, the objective function is $\sum_i e_i x_i - \sum_i \sum_j C_{ij}\, x_i x_j$. I round all e_i and C_{ij} values to the 2 digits after the decimal point.

I mainly solved the problem for diagonal case. Although it is more realistic to assume covariance between $r_i, r_j, i \neq j$, the drawback of solving with the diagonal case is not significant because:

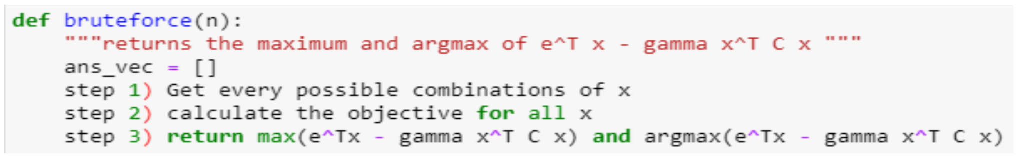
(i) The existence of cross product terms in $\gamma x^T C\, x$ overcomplicates the solution while the general idea for both cases is identical.

(ii) Despite assuming uncorrelation between $r_i, r_j, i \neq j$, the basic idea of Markowitz Portfolio: seeking balance between a) big expected return and b) small risk is still remaining.

3. Algorithms

3.1. Brute Force

The first way is to solve by Brute Force. In the specific case of brute force, I can easily solve either for correlated return or uncorrelated return. Figure 1 has the pseudo code for this approach.



```
def bruteforce(n):
    """returns the maximum and argmax of e^T x - gamma x^T C x """
    ans_vec = []
    step 1) Get every possible combinations of x
    step 2) calculate the objective for all x
    step 3) return max(e^Tx - gamma x^T C x) and argmax(e^Tx - gamma x^T C x)
```

Figure 1. Pseudo Code for Bruteforce

To derive the time complexity, I think of basic operations: 1) elementwise multiplication of two numbers to get $e^T x - \gamma x^T C\, x$, and 2) elementwise comparison of two numbers to obtain the maximum of ans_{vec} in the pseudo code. Let $M(n)$ and $C(n)$ each indicate the time complexity of multiplication and comparison. $M(n) =$

$(total\ \#\ of\ cases\ of\ x) \cdot (complexity\ of\ calculating\ e^T x - \gamma x^T C\ x\ for\ each\ x)$. The number of cases of x is the number of multicombinations of $(a_1, \dots, a_n)\ s.t.\ a_1 + \cdots + a_n = 10^k, a_i \in N \cup \{0\}$. This is $P(n + 10^k - 1; 10^k, n-1) = \binom{n+10^k-1}{10^k} \in \Theta\left(\frac{n^{10^k}}{10^k\,!}\right)$. For particular x, calculating $e^T x - \gamma x^T C\ x$ takes $\Theta(n^2)$ time for general C and $\Theta(n)$ for diagonal C. Thus, $M(n) \in \Theta\left(\frac{n^{10^k+2}}{10^k\,!}\right)$ for general C and $\Theta\left(\frac{n^{10^k+1}}{10^k\,!}\right)$ for diagonal C.

$C(n) = \ length(ans_{vec}) \in \Theta\left(\frac{n^{10^k}}{10^k\,!}\right)$. Since multiplication and comparison takes similar execution time, $M(n)$ dominates $C(n)$. Thus, $T(n) \in\ \Theta\left(\frac{n^{10^k+2}}{10^k\,!}\right) \in \Theta\left(n^{10^k+2}\right)$ for general C and $\Theta\left(\frac{n^{10^k+1}}{10^k\,!}\right) \in \Theta(n^{10^k+1})$ for diagonal C.

Space complexity $S(n)$ = (length of ans_vec)= $\Theta\left(\frac{n^{10^k}}{10^k\,!}\right) \in\ \Theta\left(n^{10^k}\right)$ for both diagonal and general case. Since we need to consider all cases of x, the asymptotic time complexity and space complexity is the same for best, average and the worst.

Empirical Analysis

For empirical analysis, I measure the execution time with various n values for k = 1 and k = 2. For k = 1, I dealt with n = 3,6,9,12,15 and for k = 2, I dealt with n = 1,2,3,4,5. Dealing with non-diagonal C matrices, I check if $T(n) \in \Theta(n^{10+2})$ for k = 1 and $\Theta(n^{100+2})$ for k = 2. Figure 2 shows the execution time (vertical axis) to compute the maximum utility for different n values (horizontal axis) for k = 1 (left subfigure) and k = 2 (right). In both cases, fitted regression lines (blue) approximate the real execution time (red). For k = 1, even for 15 items, it takes about 6 minutes and for k = 2, even for 5 items, it takes more than 13 minutes. More efficient algorithms are required.

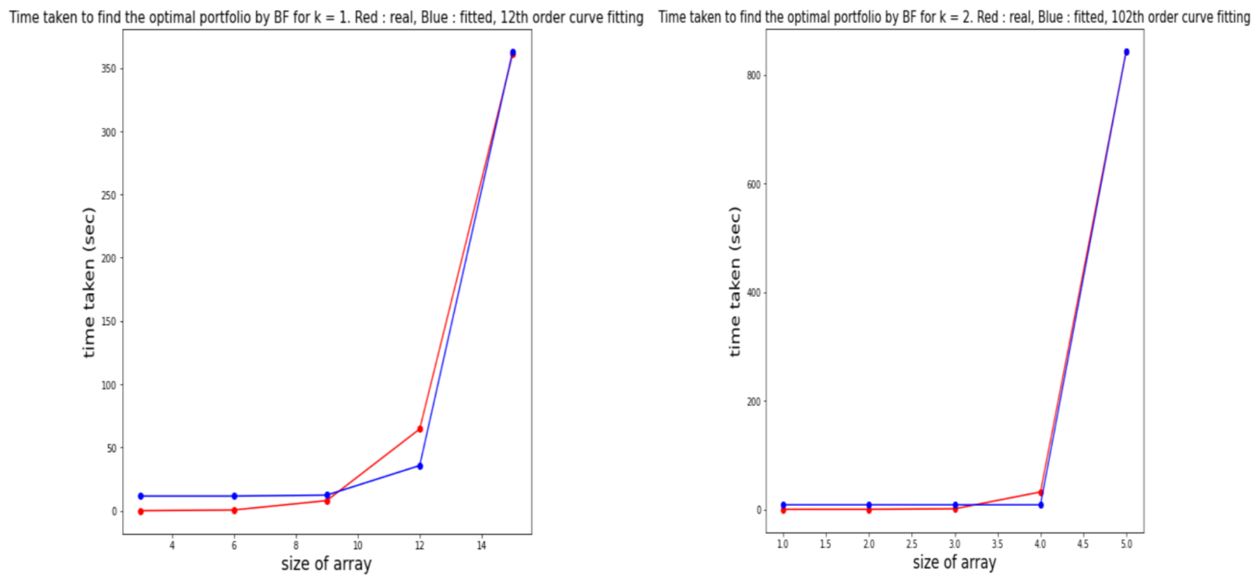


Figure 2. Empirical Analysis of k = 1 (left) and k = 2(right)

Although not shown in the report, dealing with diagonal C matrices resulted in $T(n) \in \Theta(n^{10+1})$ for k = 1, $\Theta(n^{100+1})$ for k = 2. Whether I deal with diagonal C matrix or not does not affect a lot in execution time.

3.2. Whether greedy algorithm leads to an optimal solution for diagonal C

The following issue is only for the case of diagonal C matrix where objective function is "separable" into $\sum_i (e_i x_i - \gamma C_{ii} x_i^2)$. My first consideration is greedy algorithm and check if it can result in the optimal point. Since there are no cross-product terms, the idea is to find an item i that has the maximum e_i compared to C_{ii} and only invest on that item. This seems plausible but did not work as in the following counterexample represented in Table 1. Table 1 contains the optimal proportion (4th line) for e_i (2nd line), C_{ii} (3rd line). Looking at the 4th line, I can see that multiple options (1,2,5,8) are chosen resulting in the utility 0.9222. Doubting if there are ties, I slightly changed the proportions within these items but resulted in smaller utility. Swapping the proportions between item 5 and 8 resulted in utility 0.902 and only investing in the item 5, the utility is 0.53, both smaller than 0.9222. The greedy algorithm did not work because there is no reasonable metric to see how big e_i is compared to C_{ii}. Although $\frac{e_i}{C_{ii}}$, $e_i - C_{ii}$, and so forth are candidates, it is unsure if each is a reasonable measure for this problem.

Item	1	2	3	4	5	6	7	8
Expected Return	0.71	0.97	-0.22	0.13	1.49	-0.24	-0.14	1.01
Variance	0.66	0.78	0.1	0.06	0.96	0.62	0.09	0.56
Optimal Proportion	10%	20%	0	0	40%	0	0	30%

Table 1. Counterexample of greedy algorithm, objective function value = 0.9222

3.3. Dynamic Programming: Optimal Solution

Dynamic Programming leads to the optimal solution. Let $Mark(n, k, p) = \max_{\sum_i x_i = p} e^T x - \gamma x^T C\, x$. This value is the (n, p) element of the record table "Mark". Parameters n, k each represents number of items and the digits. Additional parameter $\mathrm{p} = \sum_i x_i$ seems unnecessary because $\sum_i x_i = 1$. To obtain the answer, I call $\mathrm{Mark(n, k, 1)}$ for sure but p is needed to set up a recursive relation. Since the recurrence relation is excessively complicated for general C, I mainly address diagonal C and later deal with non-diagonal C in 3.3.2.

3.3.1. DP for diagonal C

To describe the algorithm easily, I present the algorithm using an example of n = 8, k = 1: the optimal proportion vector represented in 1 digit after the decimal point, where there are 8 number of candidate items to choose from.

I make a function call $Mark(n = 8, k = 1, p = 1)$ and Table 2 represents the order of filling in the record table until I obtain the answer.

n \ p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	1	2	3	4	5	6	7	8	9	10
2	11	17	18	19	20	21	22	23	24	25
3	12	26	27	28	29	30	31	32	33	34
4	13	35	36	37	38	39	40	41	42	43
5	14	44	45	46	47	48	49	50	51	52
6	15	53	54	55	56	57	58	59	60	61
7	16	62	63	64	65	66	67	68	69	70
8										71

Table 2. The order of executing $Mark(n = 8, k = 1, p = 1)$.

Base Case

a)$Mark(n = 1, k, p) = \max(pe_1 - \gamma p^2 C_{11}, 0)$

b)$Mark\left(n \geq 2, k, p = \frac{1}{10^k} = \frac{1}{10}\right) = \max_i(0.1e_i - 0.1^2\gamma C_{ii}, 0) = \max\left(Mark(n-1, k, 0.1), 0.1e_n - 0.01\gamma C_{nn}\right)$

I start with two basic operations. (a) means that if there is only one item to choose, I just need to see if investing all my proportion in that item would gain positive utility. As in table2, I first fill in the 1st row of the record table.

Equation (b) means that with $p = \frac{1}{10^k} = \frac{1}{10}$, I cannot select more than 1 item because I only have the unit resource.

Thus, I obtain $\max_i(0.1e_i - 0.1^2\gamma C_{ii})$ and compare this with 0: if it is worth investing (if utility is nonnegative). However, I can further simplify this equation. The $(n-1)^{th}$ row, 1st column of the record table has the information of maximum over n-1 items. Thus, to fill in the n^{th} row, 1st column, I only need to compare between 1) $(n-1)^{th}$ row, 1st column value and 2) utility of investing in n^{th} item using $p = 0.1$. Further, I do not need to compare with 0 because I start filling in the 1st column 2nd row and 1st column 1st row value is guaranteed to be nonnegative. As in table2, I fill in the 1st column of the record table.

Recursive Relation

$Mark(n, k, p) = \max\left[Mark(n-1, k, p), Mark(n-1, k, p-0.1) + (0.1e_n - 0.1^2\gamma C_{nn}), \ldots, Mark(n-1, k, 0.1) + \left((p-0.1)e_n - (p-0.1)^2\gamma C_{nn}\right), pe_n - \gamma p^2 C_{nn}\right].$

The intuition when $p = 1$ is the following. To get the maximum utility from n items allocating total proportion, I either invest all my money in the first (n-1) items or allocate 90% of my money in first n-1 items and allocate 10% in the last item, …, or allocate 10% in first n-1 items and allocate 90% in the last item or allocate all my money in the last item. Because I need values of $Mark(n-1, k, 0.9), Mark(n-1, k, 0.8)$, …, I approach from bottom up. In table2, I fill in the grey shaded areas using this recursive equation.

Memory Function

To fill in n^{th} row, $n - 1^{th}$ row values are needed. Thus, n^{th} row values are not required to be filled except from the rightmost entry $Mark(n = n, k = k, p = 1)$. All remaining terms are required to be calculated.

How to obtain the optimal proportion

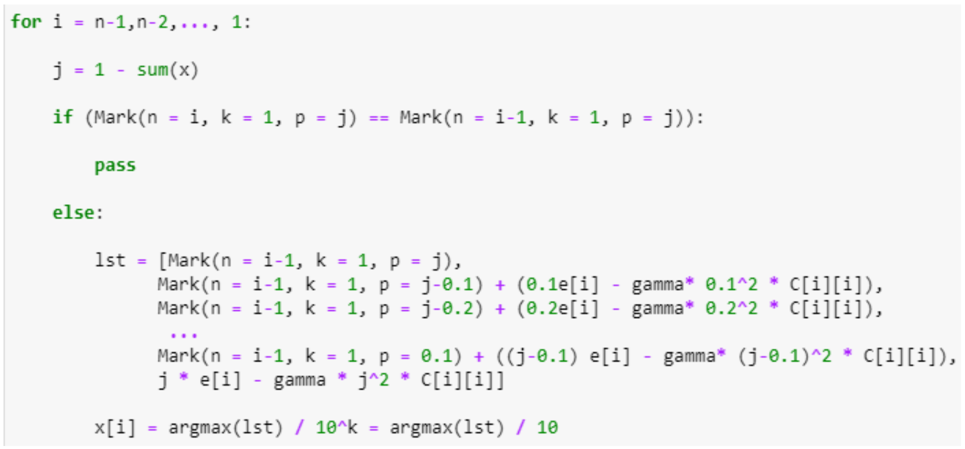
Until now, I have obtained the optimal utility value. However, a more important thing is to obtain the optimal proportion vector $\vec{x}$. While I filled in the record table in an increasing order of n, I fill in $\vec{x} = (x_1, \ldots, x_n)$ in a decreasing order of n by backtracking the record table. When k = 1, procedures are as follows:

1st) Obtain x_n from recursive equation $Mark(n, k = 1, 1) = \max\left[Mark(n-1, k, 1), Mark(n-1, k, 0.9) + (0.1e_n - 0.1^2\gamma C_{nn}), \ldots, Mark(n-1, k, 0.1) + (0.9e_n - 0.9^2\gamma C_{nn}), pe_n - \gamma p^2 C_{nn}\right].$

Let “lst” be the list I obtain the maximum from:

$$lst[0] = Mark(n-1,k,1), lst[1] = Mark(n-1,k,1), \ldots lst[10^k] = Mark(n-1,k,1) \rightarrow x_n = \frac{argmax(lst)}{10^k}$$

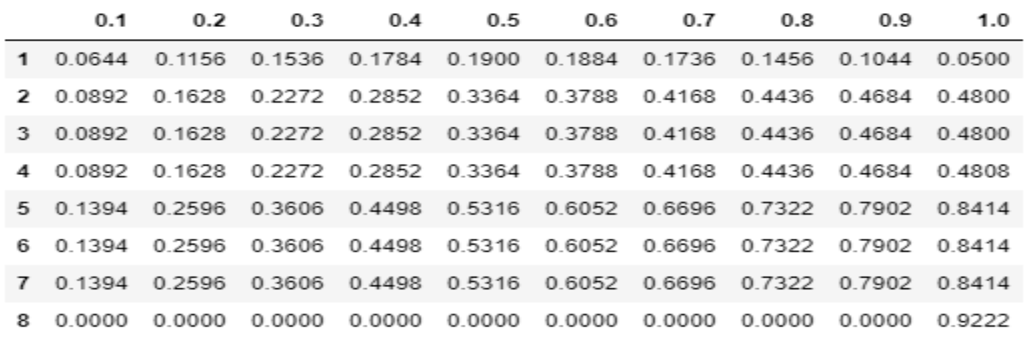
2nd) Iterate the following:



```
for i = n-1,n-2,..., 1:

    j = 1 - sum(x)

    if (Mark(n = i, k = 1, p = j) == Mark(n = i-1, k = 1, p = j)):

        pass

    else:

        lst = [Mark(n = i-1, k = 1, p = j),
               Mark(n = i-1, k = 1, p = j-0.1) + (0.1e[i] - gamma* 0.1^2 * C[i][i]),
               Mark(n = i-1, k = 1, p = j-0.2) + (0.2e[i] - gamma* 0.2^2 * C[i][i]),
                 ...
               Mark(n = i-1, k = 1, p = 0.1) + ((j-0.1) e[i] - gamma* (j-0.1)^2 * C[i][i]),
               j * e[i] - gamma * j^2 * C[i][i]]

        x[i] = argmax(lst) / 10^k = argmax(lst) / 10
```

Figure 3. Procedures to obtain $x_{n-1}, x_{n-2}, \ldots, x_1$

Since the procedure is complicated, I keep addressing the example of $Mark(n = 8, k = 1, p = 1)$.Table 3 is the obtained record table. When there are 8 items with certain e, C, 0.9222 is the maximum utility.



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1	0.0644	0.1156	0.1536	0.1784	0.1900	0.1884	0.1736	0.1456	0.1044	0.0500
2	0.0892	0.1628	0.2272	0.2852	0.3364	0.3788	0.4168	0.4436	0.4684	0.4800
3	0.0892	0.1628	0.2272	0.2852	0.3364	0.3788	0.4168	0.4436	0.4684	0.4800
4	0.0892	0.1628	0.2272	0.2852	0.3364	0.3788	0.4168	0.4436	0.4684	0.4808
5	0.1394	0.2596	0.3606	0.4498	0.5316	0.6052	0.6696	0.7322	0.7902	0.8414
6	0.1394	0.2596	0.3606	0.4498	0.5316	0.6052	0.6696	0.7322	0.7902	0.8414
7	0.1394	0.2596	0.3606	0.4498	0.5316	0.6052	0.6696	0.7322	0.7902	0.8414
8	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.9222

Table 3. The Record table obtained for calculating $Mark(n = 8, k = 1, p = 1)$.

To obtain x_8, use $Mark(8,1,1) = \max\,[Mark(7,k,1), Mark(7,k,0.9) + (0.1e_8 - 0.1^2\gamma C_{88}), \ldots, pe_8 - \gamma p^2 C_{88}]$. Since $Mark(7,k,0.7) + (0.3e_8 - 0.3^2\gamma C_{88})$ is the largest, $argmax(lst) = 3 \rightarrow x_8 = \frac{3}{10^1} = 0.3$.

$i = 7$ & $j = 0.7$. To obtain x_7, since $Mark(n = 7, k = 1, p = 0.7) = Mark(6,1,0.7) = 0.6696$, pass until $i = 5$. Intuitively, it means there is no additional utility gain from investing such items.

$i = 5$ & $j = 0.7$. To obtain x_5, use the recursive equation in figure 3 and get $argmax(lst) = 4 \rightarrow x_5 = 0.4$.

$i = 4$ & $j = 1 - 0.3 - 0.4 = 0.3$. Since $Mark(4, k = 1,0.3) = Mark(3,1,0.3) = Mark(2,1,0.3) = 0.2272$, pass until $i = 2$ & $j = 0.3$. To obtain x_2, use the recursive equation in figure 3 and get $argmax(lst) = 2 \rightarrow x_2 = 0.2$. When $i = 1$ & $j = 1 - 0.3 - 0.4 - 0.2 = 0.1$, do the final step to obtain $x_1 = 0.1$. Using backtracking, I have finally obtained that $\vec{x} = (0.1, 0.2, 0,0,0.4,0,0,0.3) \in R^8$.

Time and Space Complexity Analysis

I derive space complexity $S(n)$ first: $S(n) = (size\ of\ record\ table + length\ of\ \vec{x}) \in \Theta(10^k \cdot n + n) = \Theta(n)$.

For time complexity analysis, $T(n) = (Time\ for\ filling\ in\ the\ record\ table + Time\ for\ filling\ in\ \vec{x})$.

Procedure #1) Filling in the Record Table.

There are two basic operations. First, elementwise multiplication "M" for getting $e_i x_i - \gamma C_{ii} x_i^2$. Since the calculation of $e_i x_i - \gamma C_{ii} x_i^2$ is done by $\Theta(1)$, $M(n)$ is equivalent to the number of $e_i x_i - \gamma C_{ii} x_i^2$ calculations. Next basic operation is the elementwise comparison "C" to get a maximum. Interestingly, to fill in each item, exactly the same number of multiplication and comparison is needed. Entries of Table 4 illustrates how many basic operations are taken to fill in the record table. Thus, time taken for the first procedure is approximately sum of all elements of table 4 $\approx \frac{(10^k)(10^k+1)}{2} \times n \in \Theta\left(\frac{100^k}{2} \cdot n\right)$. Also, the number of basic operations is deterministic (best = avg = worst).

n \ p	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	1	1	1	1	1	1	1	1	1	1
2	1	2	3	4	5	6	7	8	9	10
3	1	2	3	4	5	6	7	8	9	10
4	1	2	3	4	5	6	7	8	9	10
5	1	2	3	4	5	6	7	8	9	10
6	1	2	3	4	5	6	7	8	9	10
7	1	2	3	4	5	6	7	8	9	10
8										10

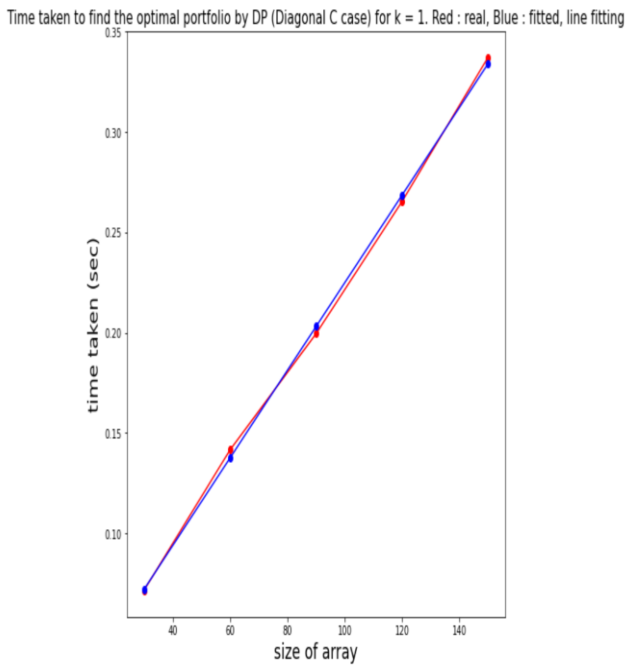
Table 4. Number of Basic Operations Required. $T(n = 8, k = 1) =$ sum of all the entries

Procedure #2) Filling in $\vec{x}$

Like in procedure #1, two basic operations are elementwise multiplication (M) and comparison (C). Though pp5-6, I illustrated backtracking method to obtain $\vec{x}$. Unlike in procedure #1, I can fortunately "pass" if $Mark(n = i, k = k, p = j) = Mark(n = i - 1, k = k, p = j)$ but the number of passes is stochastic, not deterministic. The worst case is when I need to include all items in my portfolio, resulting in time complexity of $\Theta\left(\frac{100^k}{2} \cdot n\right)$. Thus, agglomerating two steps, $T(n) \in \Theta(100^k \times n) \in \Theta(n)$.

Empirical Analysis a) Time Complexity Analysis

The first empirical analysis is the execution time with various n values k = 1 and k = 2. Since I am dealing with diagonal C matrices, $T(n) \in \Theta(n^{10^k+1})$ for brute force with small coefficients and $T(n) \in \Theta(n)$ for DP with large coefficients. Thus, it is important to empirically analyze which method is more efficient in various cases. As I wished, unlike in Brute Force, I could experiment with large n values with DP: 30,60,90,120,150 for k = 1 and n = 10,20,30,40,50 for k = 2. Figure 4 contains the empirical result and it shows that the real execution time (red) and fitted 1st order line (blue) almost coincide. When k =1, even for n = 150, it took about 0.33 second and for k = 2, even for n =50, it took only about 6.3 seconds. If it were brute force, it would have taken a few days. Thus, I could conclude that DP is superior than brute force for this problem.



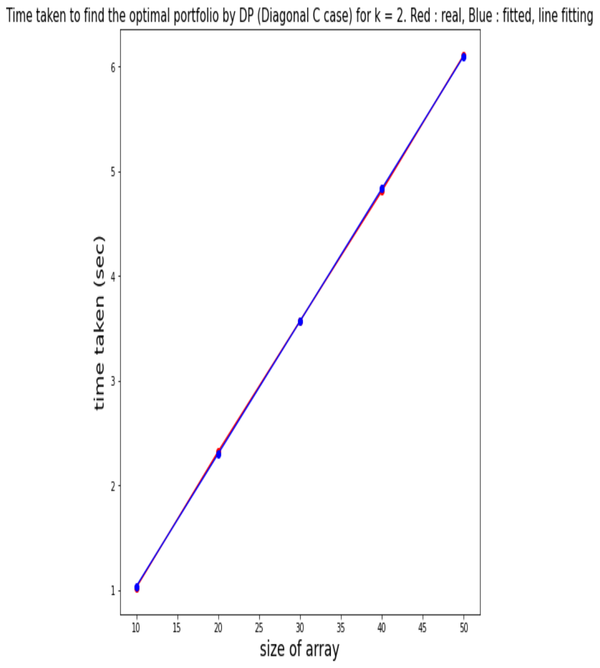


Figure 4. Empirical result from k = 1 (left) and k =2 (right)

Empirical Analysis b) If results from Brute Force and DP are consistent

If the implementation is correct, using the same data e, C, solutions from Brute Force and DP should be the same. Dealing with the following e, C values, I obtained the same answer for both approaches. Since brute force takes a lot of time, I had to deal with small n. Details are in the source code report.

(i) (n, k) = (8, 1), $e = (0.71, 0.97, -0.22, 0.13, 1.49, -0.24, -0.14, 1.01)$, $C = diag(0.66, 0.78, 0.1, 0.06, 0.96, 0.62, 0.09, 0.56)$.

(ii) (n, k) = (9, 1), $e = (0.71, 0.97, -0.22, 0.13, 1.49, -0.24, -0.14, 1.01, 0.82)$, $C = diag(0.78, 0.1, 0.06, 0.96, 0.62, 0.09, 0.56, 0.65, 0.96)$.

(iii) (n, k) = (3, 2), $e = (0.71, 0.97, -0.22)$, $C = diag(0.31, 1, 0.13)$

Comparison with Knapsack problem

How I applied Dynamic Programming is similar as in knapsack problem. However, there are some differences. In knapsack, the recursive equation is $F(i,j) = \max[F(i-1,j), v_i + F(i-1, j-w_i), if\ j - w_i \geq 0, F(i-1,j), otherwise$. Filling in the record table from left to right, I only need $\Theta(1)$ number of additions and comparisons. This led to the time efficiency of $T(n) = S(n) \in \Theta(Wn)$. However for this problem, filling in the record table from left to right, the number of basic operations gets larger by 1. This led to $T(n) \in \Theta(100^k \cdot n) > S(n) \in \Theta(10^k \cdot n)$ if I consider k as a main factor. Also, regarding the memory function, it is hard to find a pattern about which elements to remember in the Knapsack problem. This is compared to the easy pattern of memory function in the Markowitz portfolio problem: elements except from the last row should be remembered.

3.3.2. DP for non-diagonal C

I have mainly led the discussion using diagonal C. Because the recursive function is overly complicated, I could not finish the code implementation for non-diagonal C. I use the same definition of the Mark function: $Mark(n,k,p) = \max_{\sum_i x_i = p} e^T x - \gamma x^T C\, x$. However, $e^T x - \gamma x^T C\, x = \sum_i e_i x_i - \sum_i \sum_j C_{ij}\, x_i x_j \neq \sum (e_i x_i - C_{ii} x_i^2)$. Base cases are the same because in both base cases, I only invest in one item that I do not need to consider the cross product terms. The difference is the recursive equation. To form the recursive equation, the first thing to do is set up a relation of $e^T x - \gamma x^T C\, x$ between $e'^T x' - \gamma x'^T C' x'$ where $e' = (e_1, \dots, e_{n-1})$, $x' = (x_1, \dots, x_{n-1})$, C': first $(n-1) \times (n-1)$ part of C. Figure 5 contains the relationship between $e^T x - \gamma x^T C\, x$ and $e'^T x' - \gamma x'^T C' x'$. Blue highlighted terms are already existing terms and red highlighted terms are newly introduced terms.

$$e^T x - \gamma x^T C x = \sum_{i=1}^{n} e_i x_i - \gamma \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij} x_i x_j$$

$$= e_1 x_1 + \cdots + e_{n-1} x_{n-1} + e_n x_n - \gamma \left[(x_1 \cdots x_{n-1}\ \ x_n) \begin{pmatrix} C_{11}x_1 + \cdots + C_{1,n-1}x_{n-1} + C_{1n}x_n \\ C_{21}x_1 + \cdots + C_{2,n-1}x_{n-1} + C_{2n}x_n \\ \vdots \\ C_{n-1,1}x_1 + \cdots + C_{n-1,n-1}x_{n-1} + C_{n-1,n}x_n \\ C_{n,1}x_1 + \cdots + C_{n,n-1}x_{n-1} + C_{nn}x_n \end{pmatrix} \right]$$

$$= e'^T x'^T - \gamma x'^T C' x' + e_n x_n - \gamma \big(x_1 C_{1n} x_n + x_2 C_{2n} x_n + \cdots + x_{n-1} C_{n-1,n} C_n + x_n C_{n,1} x_1 + \cdots + x_n C_{n,n-1} x_{n-1} + x_n C_{nn} x_n \big)$$

Figure 5. Relationship between $e^T x - \gamma x^T C\, x$ and $e'^T x' - \gamma x'^T C' x'$

To take an example of k = 1, I can obtain the recursive equation by putting $0, 0.1, \dots, p$ in the position of x_n:

$$\begin{aligned}
Mark(n, k=1, p) = \max \Big[& Mark(n-1, 1, p), \\
& Mark(n-1,1,p-0.1) + 0.1 e_n - 0.1\gamma \big(x_1 C_{1n} + x_2 C_{2n} + \cdots + x_{n-1} C_{n-1,n} + C_{n,1} x_1 + \cdots + C_{n,n-1} x_{n-1} + 0.1\, C_{nn} \big), \\
& Mark(n-1,1,p-0.2) + 0.2 e_n - 0.2\gamma \big(x_1 C_{1n} + x_2 C_{2n} + \cdots + x_{n-1} C_{n-1,n} + C_{n,1} x_1 + \cdots + C_{n,n-1} x_{n-1} + 0.2\, C_{nn} \big), \\
& \vdots \\
& Mark(n-1,1,0.1) + (p-0.1) e_n - (p-0.1)\gamma \big(x_1 C_{1n} + x_2 C_{2n} + \cdots + x_{n-1} C_{n-1,n} + C_{n,1} x_1 + \cdots + C_{n,n-1} x_{n-1} + (p-0.1) C_{nn} \big), \\
& p e_n - p\gamma \big(x_1 C_{1n} + x_2 C_{2n} + \cdots + x_{n-1} C_{n-1,n} + C_{n,1} x_1 + \cdots + C_{n,n-1} x_{n-1} + p\, C_{nn} \big) \Big]
\end{aligned}$$

Figure 6. Recursive Equation obtained by putting $0, 0.1, \dots, p$ in the position of x_n

Time and Space Complexity Analysis

For time complexity analysis, it is notable that the order of filling in the record table and the memory function are identical as in 3.3.1. The key difference is how many more basic operations (multiplication and comparison) are needed to fill in the record table starting from left to right. In the diagonal case, the number of basic operations got larger by 1. However, in non-diagonal case, the number of basic operations gets larger by $\Theta(2n) = \Theta(n)$. This change leads to $T(n) \in \Theta(100^k \times n \times n) = \Theta(n^2)$. The space complexity stays the same as $S(n) \in \Theta(10^k \times n)$ because the record table structure remains the same.

4. Other Approaches for Markowitz Portfolio

Since the Markowitz Portfolio is a famous model, there have been approaches to solve this. Fu et al. (1998) addressed approximation algorithms for QP and Kamath et al. (1992) addressed the interior point approaches. Both are specific models for Quadratic Programming and are reliant on knowledge of optimization. Faaland (1974) deals with a Quadratic Integer Programming method. This could seem similar as my problem but the problem itself is different. In Faaland (1974), the term “integer” means that the investor limits the number of items in a portfolio. In my problem, I limit the number of digit representation.

5. Generalization of this problem

The Markowitz portfolio has a foundational role in “mean-variance analysis”, a subfield of econometrics. However, the true value of this problem is generalization into proportion optimization problems. In our life, we often confront situations to allocate our resources (time / money / energy). Allocating a day in sleep (30%), study (30%), leisure (30%), driving (10%) can be this example. Constraints of problems in this class are universally the same and we only need to modify the objective function for each problem. In the Markowitz portfolio problem, the objective function is $e^T x - \gamma x^T C\, x$ and in a problem of capacity of discrete memoryless channel, the objective contains a form of entropy. This means that I do not have to build “structurally” different models for different class of objective functions (e.g., QP for quadratic function, LP for affine function, etc).

6. References

Markowitz, H. (1952). Portfolio Selection, Journal of Finance. *Markowitz HM—1952.—№*, 77-91.

Fu, M., Luo, Z. Q., & Ye, Y. (1998). Approximation algorithms for quadratic programming. *Journal of combinatorial optimization*, *2*(1), 29-50.

Kamath, A. P., Karmarkar, N. K., Ramakrishnan, K. G., & Resende, M. G. (1992). A continuous approach to inductive inference. *Mathematical programming*, *57*(1), 215-238.

Faaland, B. (1974). An integer programming algorithm for portfolio selection. *Management Science*, *20*(10), 1376-1384.