ISE 5406 - Optimization II

Project Part 2

Due on Thursday (05/06) by 8 am EST

Team Members' Name:
1. Tai-Jung Chen (Contribution 33.33%)
2. Andrew Hartley (Contribution 33.33%)
3. Ruochen Wang (Contribution 33.33%)

Total Possible Points (150 Points)

The objective of this part of the mini-projects is to give you an experience in implementing algorithm(s) discussed in this course for solving real-world applied nonlinear programming problems.

Assignment:

- Literature Review: Identify different solvers/libraries available to solve nonlinear programming problems, in particular convex optimization problems. Perform a literature review and discuss about the features of these solvers.[50 points]
- Problem Definition and Methodology Selection: Similar to Part I of the project, clearly describe and formulate one problem from either Machine Learning or Portfolio Optimization/Economics. Thereafter among the algorithms discussed in class, identify which algorithm your team would like to consider to solve these problems? Discuss why your team made this selection? [35 points]
- Design of Experiments: Either generate random instances or consider instances available in literature for the selected problems, and utilize one of the above discussed solvers/libraries to solve these problem instances. [25 points]
- Computational Results: Provide a discussion on the results of your computational experiments and discuss about potential future extensions. [40 points]

Evaluation Criteria.

- Report. Each team (with at most 3 members) is required to turn in a well-written typed report (Font: 12 pt Times New Roman; Line spacing: 1.5 or double; One-Inch Left/Right/Top/Bottom Margin). The report should have this page as the cover page, including team members' names and signatures. It is very important to cite references used to prepare this report.
- Peer Evaluation. On the cover sheet, you are required to provide contributions of each team member in this part of the project; for example, in case all members of a team of three students have contributed equally, write 33.33% in front of each member's name. This evaluation will be used while assigning scores for each team member.

Literature Review

In this class, we have learned there are a variety of methods to solve nonlinear programs as well as a variety of types of nonlinear programming problems. Since there are many types of nonlinear programs and different methods to solve them, people have developed many different softwares to solve nonlinear programs. In this literature review, we will examine a few common nonlinear programming solvers: Gurobi, SciPy, and the GAMS repository.

Gurobi is one of the most well-known optimization solvers in industry and academia as it has many options and can be used for a variety of problems including quadratic programs, mixed-integer quadratic programs, mixed-integer quadratically-constrained programs, and mixed-integer second-order cone programs. The type of program should be specified when "optimize" is called in the code. To further specify the problem type the objective function can be set to linear, mixed-integer linear, quadratic, or mixed-integer quadratic. Within the features of setting the objective function, we can also specify a quadratic program to be solved with a continuous objective function, discrete quadratic with a convex relaxation, or discrete quadratic with a nonconvex relaxation. Gurobi also allows the user to set up multiple linear objective functions. The user can also write models with linear, quadratic, special order set constraints as well as some simple general constraints including max/min values, absolute value, indicator, and AND/OR constraints. Gurobi can also handle some functional constraints such as sine, cosine, tangent, logarithmic, exponential, power, and polynomial constraints. Users can also model many types of variables including continuous, general integer, binary, and semi-continuous and semi-integer variables in Gurobi. Finally, since some problems will take a lot of time to solve, Gurobi allows the user to set time limits and/or bounds on solutions. In addition to being able to solve many types of problems, Gurobi can be used through many different programming languages such as Python, MATLAB, and R among others. Since Gurobi is a commercial software, the exact solving methods it uses are not public. We use Gurobi to solve an example portfolio optimization problem that is set up as a quadratic program in the latter part of this assignment.

Although Gurobi can be called in Python, Python has its own optimization package called "scipy.optimize"[1] (we will call it SciPy in latter content for convenience). Compared to Gurobi, SciPy is a bit more limited in what it can solve, but that is likely because it is free to download for anyone (opensource). SciPy does allow the user to decide which algorithm they would like Python to employ as it is a good solver for unconstrained and constrained minimization of multivariate scalar functions as well as least-squares minimization and global optimization. Algorithms can be chosen as a parameter of the minimize function. We use SciPy to perform Support Vector Machine (SVM) from scratch in order to solve the spam email classification problem (later excluded from this project since we only need one problem). We found that the nature of the minimization function (scipy.optimize.minimize()) enforced the program to loop over the arrays that contained data. Such constraint lowers the speed of the program and wastes the vectorization property of NumPy (another useful package for Python when doing matrix operations) array. Vectorization is designed to do parallel computations on the Numpy array to increase the speed of the program. In other words, vectorization allows the elements in an array to be computed simultaneously. That is why looping over the arrays isn't the best way for optimization in Python. On the other hand, owing to the universality and popularity of Python language, one can be adopted to using SciPy easily. Additionally, lots of machine learning implementation packages are done by Python, so using SciPy is also a convenient way to integrate the implementation together.

Finally, GAMS is a solver interface that allows the user to choose from a plethora of solvers that each have different purposes. For the sake of this literature review, we will limit our discussion to PATHNLP, KNITRO, and MINOS. PATHNLP is a solver that focuses on constructing the KKT conditions for a problem which makes it a great option for convex nonlinear programming. If the KKT conditions are difficult to find or solve, this solver is not a great option as it does not have many options. KNITRO is a solver designed for large scale nonlinear programming as it has options to solve using derivative-free, first, or second derivative methods. This solver can also employ the interior point method and iterative or direct approaches in computing the solution. Finally, MINOS is a general-purpose nonlinear programming solver that is designed to find local extreme points of smooth functions. For problems with nonlinear objective functions, MINOS uses a combination of the reduced-gradient method and the quasi-Newton method. The downside of this solver is that the user needs to know a good starting point to reach the optimal solution as Newton's method is dependent on the starting point. Each of these solvers has a specific purpose and it is important to understand this as we choose which solver to use for a given problem.

Problem Definition and Methodology Selection

Now that we have some knowledge of the theory of nonlinear programming and solvers we will look at a few example problems from the existing literature on portfolio optimization. Portfolio optimization is the study of how to allocate funds in the various money markets to provide the best returns. Markowitz's theory of mean-variance optimization is a well-known mechanism in portfolio optimization and can be modeled as a convex quadratic program. We will use this theory to determine the combination of stocks, bonds, and money markets an investor should choose in order to achieve a particular "rate of return", the average amount of growth, given an amount of variance. There is an inherent tradeoff in investing between the rate of return and variance as some markets are more volatile than others which provides more opportunities for gains, but also more opportunities for losses. In this model, we will initially use S&P 500 data from 1960-2003 (Appendix A.) to create a "portfolio" of options with different rates of return and variances which would be used as a predictive model for the future, and then modify the data to provide a "sensitivity analysis" and see how the results change[2]. The objective of this model is to minimize the expected variance with respect to desired rates of return to generate a collection of "efficient portfolios".

That being said, we first need to use the data to define the parameters of our model. We will define stocks to be represented as asset S_1, bonds as S_2, and money markets as S_3. We can denote x_i to be the proportion of the total funds invested in security. One can represent the expected return and the variance of the resulting portfolio $x = (x_1,..., x_n)$as follows:

$$E[x] = x_1\mu_1 + ... + x_n\mu_n = \mu^T x$$

$$Var[x] = \sum_{i,j} \rho_{ij}\sigma_i\sigma_j x_i x_j = x^T\Sigma x$$

where μ_i *is the geometric average rate of return from the data set, and* $\rho_{ii} \equiv 1$.

Since variance is always nonnegative, it follows that $x^T\Sigma x \geq 0$ for any x, i.e., Σ is positive semidefinite. It is usually assumed that it is positive definite, which is essentially equivalent to assuming that there are no redundant assets in our collection $S_1,\ldots, S_n$. We can further assume that the set of admissible portfolios is a nonempty polyhedral set and represent it as $X := \{x: Ax = b, Cx >= d\}$, where A is an m × n matrix, b is an m-dimensional vector, C is a p × n matrix, and d is a p-dimensional vector. In particular, one of the constraints in the set X is

$$\sum_{i=1}^{n} x_i = 1$$

When we assume that Σ is positive definite, the variance is a strictly convex function of the portfolio variables and there exists a unique portfolio in X that has the minimum variance. Let us denote this portfolio with x_{min} and its return $\mu^T x_{min}$ with R_{min}. Note that x_{min} is an efficient portfolio. Let R_{max} denote the maximum return for an admissible portfolio.

Then to find the minimum variance portfolio of the securities 1 to n that yields at least a target value of the expected return (say b), mathematically we can formulate a quadratic programming problem as follows:

$$min_x \frac{1}{2} x^T \Sigma x$$

$$s.t.\ \mu^T x >= R$$

$$Ax = b$$

$$Cx >= d$$

The first constraint indicates that the expected return is no less than the target value R. Solving this problem for values of R ranging between R_{min} and R_{max}, we can obtain all efficient portfolios (portfolios that have the minimum variance among all portfolios that have at least a certain expected return). The objective function corresponds to one-half of the total variance of the portfolio. The constant $\frac{1}{2}$ is added for convenience in the optimality conditions and obviously, it does not affect the optimal solution. This is a convex quadratic programming problem for which the first-order conditions are both necessary and sufficient for optimality. x_R is an optimal solution of the above convex quadratic problem if and only if there exists $\lambda_R \in R$, $\gamma_E \in R^m$,and $\gamma_I \in R^p$ satisfying the following conditions:

$$\Sigma x_R - \lambda_R \mu - A^T \gamma_E - C^T \gamma_I = 0,$$

$$\mu^T x_R >= R, A x_R = b, C x_R >= d,$$

$$\lambda_R >= 0, \lambda_R(\mu^T x_R - R) = 0,$$

$$\gamma_I >= 0, \gamma_I^T (C x_R - d) = 0.$$

By noticing that the portfolio optimization model addressed here is a convex quadratic problem with linear constraints, we think that the projected steepest descent method can be an efficient algorithm to solve this problem because when the objective function is a convex function, the method converges to a global minimizer as long as it converges. We will employ Gurobi to solve this model as it is a state-of-the-art solver for quadratic programming problems.

Design of Experiments

Since we have defined the problem in a general sense we will define a specific instance of the problem. We can use the data in the appendix to find the necessary mean rates of return as well as the covariance matrix Σ:

$$\begin{vmatrix} 0.02778 & 0.00387 & 0.00021 \\ 0.00387 & 0.01112 & -0.00020 \\ 0.00021 & -0.00020 & 0.01115 \end{vmatrix}$$

Using this covariance matrix we can formulate the problem as follows:

$$min \quad 0.02778x_1^2 + 2 \cdot 0.00387x_1x_2 + 2 \cdot 0.00021x_1x_3 + 0.01112x_2^2 - 2 \cdot 0.0002x_2x_3 +$$

$$Subject\ to: 0.1073x_1 + 0.0737x_2 + 0.0627x_3 \geq R$$

$$x_1 + x_2 + x_3 = 1$$

$$x_1, x_2, x_3 \geq 0$$

We will solve this model from R = 6.5 % to R = 10.5 % with a step length of 0.5 %, that is $R \in \{0.065, 0.07, 0.075, 0.08, 0.085, 0.09, 0.095, 0.1, 0.105\}$,to get the set of investing strategies at each return rate with the smallest variance.

Computational Results

1. Original Results

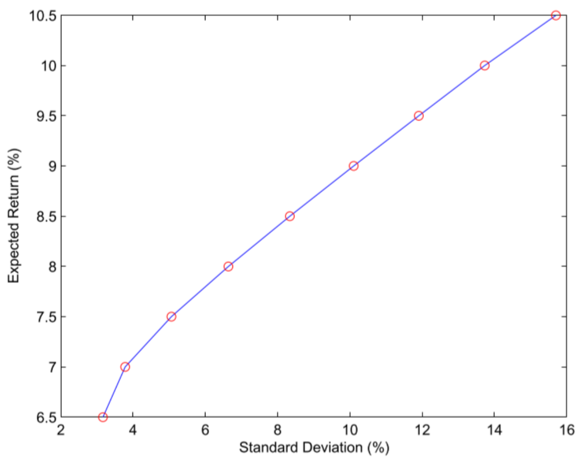
Running this through the code in Appendix B. with the dataset from Appendix A., we get results summarized in **Table 1**.

Table 1. Efficient Portfolios generated by the original dataset

Rate of Return R	Variance	Stocks	Bonds	MM
0.065	0.00100	0.02630	0.10244	0.87126
0.070	0.00143	0.13373	0.12143	0.74484
0.075	0.00256	0.24115	0.14043	0.61842
0.080	0.00441	0.34857	0.15942	0.49201
0.085	0.00696	0.45599	0.17841	0.36559
0.090	0.01022	0.56342	0.19740	0.23918
0.095	0.01418	0.67084	0.21640	0.11276
0.100	0.01886	0.78274	0.21726	0.00000
0.105	0.02465	0.93155	0.06845	0.00000

MM: Money Market

Note that a feasible portfolio x is called "efficient" if it has the maximal expected return among all portfolios with the same variance, or alternatively if it has the minimum variance among all portfolios that have at least a certain expected return. The efficient frontier is formed by the collection of efficient portfolios. The efficient frontier is often represented in a form of a curve in a two-dimensional graph where the coordinates of a plotted point correspond to the standard deviation and the expected return of an efficient portfolio. In **Figure 1**, the expected return - standard deviation plot was depicted[2] to visualize the efficient frontier. Every possible portfolio consisting of long positions in stocks, bonds, and money market investments is represented by a point lying on or below the efficient frontier in the standard deviation/expected return plane. Based on the results in **Table 1**, we are able to identify the optimal allocations of the portfolios on the efficient frontier[2] shown in **Figure 2**.



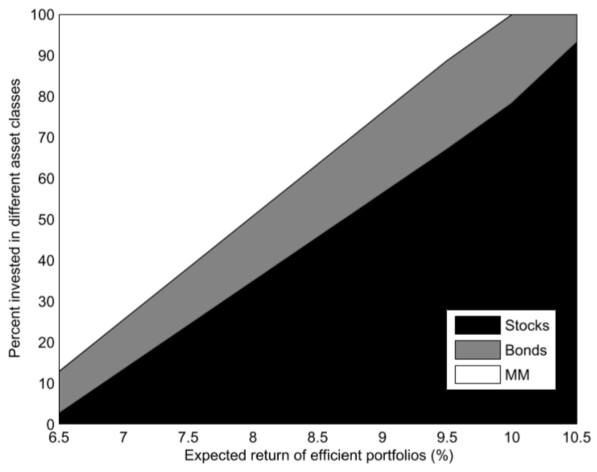


Figure 1. Efficient Frontier **Figure 2.** Composition of Efficient Portfolios

2. Modified Results

Next, we modified the data to see how it impacts the results of the model. Specifically, we changed the rate of return data for the bonds market to decrease the variance associated with bonds. We hypothesized that this would result in a higher proportion of overall investments in the bonds market and that the overall minimum variances would decrease since its variance decreased relative to the other markets. The results are shown in **Table 2**.

Table 2. Efficient Portfolios generated by the modified dataset

Rate of Return R	Variance	Stocks	Bonds	MM
0.065	0.00043	0.0000	0.79621	0.20379
0.070	0.00043	0.0000	0.79622	0.20378
0.075	0.00044	0.0000	0.79665	0.20335
0.080	0.00060	0.06164	0.93836	0.0000
0.085	0.00199	0.23288	0.76712	0.0000
0.090	0.00498	0.40411	0.5959	0.0000
0.095	0.00954	0.57534	0.42466	0.00000
0.100	0.01574	0.74658	0.25342	0.00000
0.105	0.02351	0.91780	0.08219	0.00000

MM: Money Market

This table confirms our hypothesis as bonds are invested in at a much higher rate in each run with respect to return rate and since the variance associated with bonds is lower than in the original run, each of the respective variances are smaller in this run than the original run. There are many other different ways we could modify the data to get different results but it is important to see how the model adjusts to modified data. Since portfolio optimization provides a prescriptive model to get particular returns on investment it is important to study how sensitive the model is to the data. People make many predictions about financial markets and a potential extension of looking into "sensitivity analysis" with portfolio optimization would be to test different forecasted data in the model to see how it impacts the results. While we applied the model to money markets, this model could be extended to other types of "investments" as we could define the parameters to fit many similar problems.

In this project, we are happy to be able to extend our knowledge of nonlinear optimization by learning about some of the available solvers and applying the solvers to a real-world problem which is portfolio optimization. Comparing to Linear Programming we learned in the last semester, Nonlinear optimization is a much broader topic that has a great impact in many settings such as machine learning and economics among others. Because nonlinear optimization is useful in so many areas and is such a broad subject itself, it is an important and interesting topic to study and research.

Reference

a. Paper and textbooks

[1] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) **SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python**. *Nature Methods*, 17(3), 261-272.

[2] Gerard Cornuejols, and Reha Tutuncu: Optimization Methods in Finance (textbook)

b. Reference links

1. List of optimization software:
https://en.wikipedia.org/wiki/List_of_optimization_software

2. Gurobi:
https://www.gurobi.com/documentation/9.1/refman/py_python_api_overview.html

3. GAMS:
https://gams.com/latest/docs/S_PATHNLP.html
https://gams.com/latest/docs/S_KNITRO.html
https://gams.com/latest/docs/S_MINOS.html

4. CPLEX
https://ibmdecisionoptimization.github.io/tutorials/html/Beyond_Linear_Programming.html

Appendix A:

The annual times series for the “Total Return" for each asset between 1960 and 2003.

A-1. Original Data:

Year	Stock	Bond	MM	**Year**	Stock	Bond	MM	**Year**	Stock	Bond	MM
1961	26.81	2.20	2.33	**1980**	32.50	-2.48	18.90	**1999**	21.04	-7.51	5.30
1962	-8.78	5.72	2.93	**1981**	-4.92	4.04	12.37	**2000**	-9.10	17.22	6.40
1963	22.69	1.79	3.38	**1982**	21.55	44.28	8.95	**2001**	-11.9	5.51	1.82
1964	16.36	3.71	3.85	**1983**	22.56	1.29	9.47	**2002**	-22.1	15.15	1.24
1965	12.36	0.93	4.32	**1984**	6.27	15.29	8.38	**2003**	28.68	0.54	0.98
1966	-10.1	5.12	5.40	**1985**	31.17	32.27	8.27				
1967	23.94	-2.86	4.51	**1986**	18.67	22.39	6.91				
1968	11.00	2.25	6.02	**1987**	5.25	-3.03	6.77				
1969	-8.47	-5.63	8.97	**1988**	16.61	6.84	8.76				
1970	3.94	18.92	4.90	**1989**	31.69	18.54	8.45				
1971	14.30	11.24	4.14	**1990**	-3.10	7.74	7.31				
1972	18.99	2.39	5.33	**1991**	30.46	19.36	4.43				
1973	-14.7	3.29	9.95	**1992**	7.62	7.34	2.92				
1974	-26.5	4.00	8.53	**1993**	10.08	13.06	2.96				
1975	37.23	5.52	5.20	**1994**	1.34	-7.32	5.45				
1976	23.93	15.56	4.65	**1995**	37.58	25.94	5.60				
1977	-7.16	0.38	6.56	**1996**	22.96	0.13	5.29				
1978	6.57	-1.26	10.03	**1997**	33.36	12.02	5.50				
1979	18.61	-1.26	13.78	**1998**	28.58	14.45	4.68				

Note: MM denotes Money Market

Appendix B:

The Matlab code for implementing the portfolio optimization problem

B-1. portfolio.m

```
function portfolio()
% Copyright 2020, Gurobi Optimization, LLC
%
%  minimize
%      0.02778x_s^2 + 2*0.00387x_s*x_b +
2*0.00021x_s*x_m+0.01112x_b^2-2*0.00020x_b*x_m+0.00115x_m^2
%  subject to
%      0.1073x_s+0.0737x_b+0.0627x_m >= R
%      x_s+x_b+x_m=1
%      x, y, z non-negative
%
% It solves it once as a continuous model, and once as an integer
% model.

names = {'x_s', 'x_b', 'x_m'};
model.varnames = names;
model.Q = sparse([0.02778 0.00387 0.00021; 0.00387 0.01112 -0.00020; 0.00021 -0.00020
0.00115]);  % x^T*Q*x define the quadratic term in objective function
model.A = sparse([0.1073 0.0737 0.0627; 1 1 1; -1 -1 -1]);   % A define the constraint matrix
of LHS
model.obj = [0 0 0];   %obj define the linear term in objective function
model.rhs = [0.105 1 -1];   % rhs define the rhs of the constraints
model.sense = '>';

gurobi_write(model, 'qp.lp');

results = gurobi(model);

for v=1:length(names)
    fprintf('%s %e\n', names{v}, results.x(v));
end

fprintf('Obj: %e\n', results.objval);
```

```
model.vtype = 'B';

results  = gurobi(model);

for v=1:length(names)
    fprintf('%s %e\n', names{v}, results.x(v));
end

fprintf('Obj: %e\n', results.objval);

end
```

B-2. IniPortfolio.m

```
function IniPortfolio
    p_s=zeros(1,43);
    p_b=zeros(1,43);
    p_m=zeros(1,43);

p_s=[26.81,-8.78,22.69,16.36,12.36,-10.10,23.94,11.00,-8.47,3.94,14.30,18.99,-14.69,-26.47,3
7.23,23.93,-7.16,6.57,18.61,32.50,-4.92,21.55,22.56,6.27,31.17,18.67,5.25,16.61,31.69,-3.10,3
0.46,7.62,10.08,1.32,37.58,22.96,33.36,28.58,21.04,-9.10,-11.89,-22.10,29.68]
    p_s=0.01.*p_s

p_b=[0.1042,0.0397,0.0419,0.0520,0.0904,0.0970,0.0903,0.0746,0.0823,0.0622,0.0981,0.053
6,0.0934,0.0532,0.0680,0.0885,0.1009,0.0450,0.1129,0.1006,0.0774,0.0734,0.0742,0.0830,0.0
792,0.0824,0.1039,0.1021,0.0900,0.0688,0.1034,0.0811,0.0666,0.1136,0.1086,0.0825,0.0883,
0.0855,0.0551,0.0626,0.0762,0.0569,0.01060]
    %for i=1:43
    %   p_b(i)=3.85+8*rand(1)
    %end

p_m=[2.33,2.93,3.38,3.85,4.32,5.40,4.51,6.02,8.97,4.90,4.14,5.33,9.95,8.53,5.20,4.65,6.56,10.
03,13.78,18.90,12.37,8.95,9.47,8.38,8.27,6.91,6.77,8.76,8.45,7.31,4.43,2.92,2.96,5.45,5.60,5.2
9,5.50,4.68,5.30,6.40,1.82,1.24,0.98]
    p_m=0.01.*p_m
    r_s=mean(p_s)     %arithmetic mean
    r_b=mean(p_b)
    r_m=mean(p_m)
    t_s=1;t_b=1;t_m=1;
    cov_sb=0;cov_sm=0;cov_bm=0;
    cov_ss=0;cov_bb=0;cov_mm=0;
    for i=1:43
        t_s=(1+p_s(i))*t_s;
        t_b=(1+p_b(i))*t_b;
        t_m=(1+p_m(i))*t_m;
        cov_sb=cov_sb+(p_s(i)-r_s)*(p_b(i)-r_b);
        cov_sm=cov_sm+(p_s(i)-r_s)*(p_m(i)-r_m);
        cov_bm=cov_bm+(p_b(i)-r_s)*(p_m(i)-r_m);
        cov_ss=cov_ss+(p_s(i)-r_s)^2;
        cov_bb=cov_bb+(p_b(i)-r_b)^2;
        cov_mm=cov_mm+(p_m(i)-r_m)^2;
    end
    mu_s=t_s^(1/43)-1,mu_b=t_b^(1/43)-1,mu_m=t_m^(1/43)-1    %geometric mean
    cov_sb=cov_sb/43,cov_sm=cov_sm/43,cov_bm=cov_bm/43     %covariance
    cov_bb=cov_bb/43,cov_ss=cov_ss/43,cov_mm=cov_mm/43
end
```

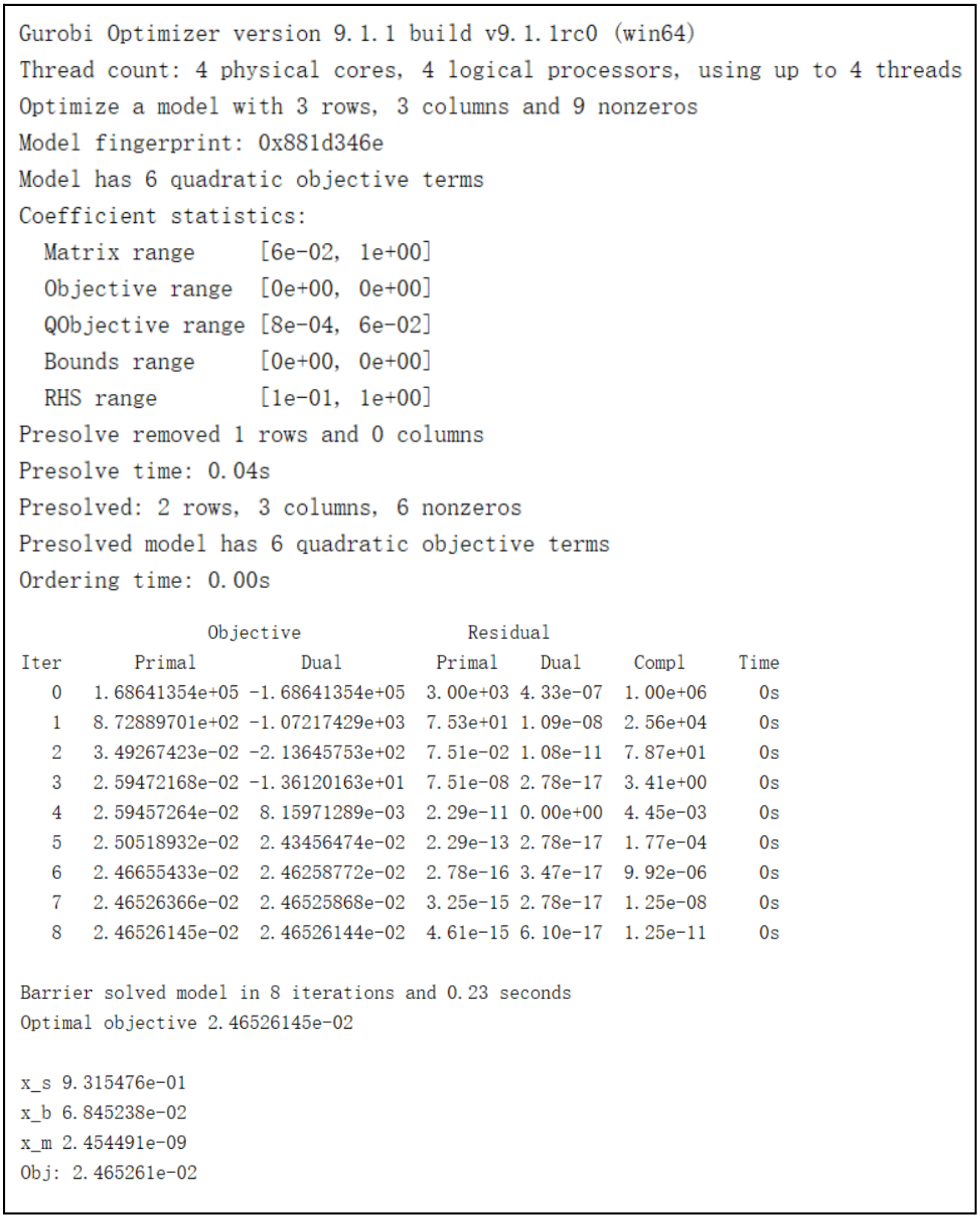
B-3. portfolio2.m

```
function portfolio2()

names = {'x_s', 'x_b', 'x_m'};
model.varnames = names;
model.Q = sparse([0.0278 5.5977*(10^(-4)) 1.9200*(10^(-4)); 5.5977*(10^(-4))
4.8741*(10^(-4)) 2.4230*(10^(-4)); 1.9200*(10^(-4)) 2.4230*(10^(-4))  0.0012]);  %
x^T*Q*x define the quadratic term in objective function
model.A = sparse([0.1074 0.0782 0.0627; 1 1 1; -1 -1 -1]);   % A define the constraint matrix
of LHS
model.obj = [0 0 0];   %obj define the linear term in objective function
model.rhs = [0.065 1 -1];   % rhs define the rhs of the constraints
model.sense = '>';

gurobi_write(model, 'qp.lp');

results = gurobi(model);

for v=1:length(names)
    fprintf('%s %e\n', names{v}, results.x(v));
end
```

```
fprintf('Obj: %e\n', results.objval);

end
```

Appendix C:

The output of the Matlab code



```
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (win64)
Thread count: 4 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 3 rows, 3 columns and 9 nonzeros
Model fingerprint: 0x881d346e
Model has 6 quadratic objective terms
Coefficient statistics:
  Matrix range     [6e-02, 1e+00]
  Objective range  [0e+00, 0e+00]
  QObjective range [8e-04, 6e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e-01, 1e+00]
Presolve removed 1 rows and 0 columns
Presolve time: 0.04s
Presolved: 2 rows, 3 columns, 6 nonzeros
Presolved model has 6 quadratic objective terms
Ordering time: 0.00s

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   1.68641354e+05 -1.68641354e+05  3.00e+03 4.33e-07  1.00e+06     0s
   1   8.72889701e+02 -1.07217429e+03  7.53e+01 1.09e-08  2.56e+04     0s
   2   3.49267423e-02 -2.13645753e+02  7.51e-02 1.08e-11  7.87e+01     0s
   3   2.59472168e-02 -1.36120163e+01  7.51e-08 2.78e-17  3.41e+00     0s
   4   2.59457264e-02  8.15971289e-03  2.29e-11 0.00e+00  4.45e-03     0s
   5   2.50518932e-02  2.43456474e-02  2.29e-13 2.78e-17  1.77e-04     0s
   6   2.46655433e-02  2.46258772e-02  2.78e-16 3.47e-17  9.92e-06     0s
   7   2.46526366e-02  2.46525868e-02  3.25e-15 2.78e-17  1.25e-08     0s
   8   2.46526145e-02  2.46526144e-02  4.61e-15 6.10e-17  1.25e-11     0s

Barrier solved model in 8 iterations and 0.23 seconds
Optimal objective 2.46526145e-02

x_s 9.315476e-01
x_b 6.845238e-02
x_m 2.454491e-09
Obj: 2.465261e-02
```