# Solving Minimax problems with Feasible Sequential Quadratic Programming

Qianli Deng (Shally)

E-mail: dqianli@umd.edu

Advisor: Dr. Mark A. Austin

Department of Civil and Environmental Engineering

Institute for Systems Research

University of Maryland, College Park, MD 20742

E-mail: austin@isr.umd.edu

# Abstract:

As one of the SQP-type programming, Feasible Sequential Quadratic Programming (FSQP) is particularly suited to the various classes of engineering applications, where the number of variables is not too large but the evaluations of objective or constraint functions and of their gradients are highly time consuming. This project is intended to implement the FSQP algorithm in Java, named JFSQP. As an optimization tool set, JFSQP is designed to solve the minimax optimization problems with both linear and nonlinear, equality and inequality constraints. Sequential quadratic programs and feasible iteration times are involved.

Keywords: Nonlinear optimization; minimax problem; sequential quadratic programming; Java

# 1. Introduction

The role of optimization in both engineering analysis and design is continually expanding. As a result, the faster and more powerful optimization algorithms are in constant demand. Motivated by problems from engineering analysis and design, feasible sequential quadratic programming (FSQP) are developed as a dramatic reduction in the amount of computation while still enjoying the same global and fast local convergence properties. The application of FSQP includes all branches of engineering, medicine, physics, astronomy, economics and finances, which abounds of special interest. In particular, the algorithms are particularly appropriate for problems where the number of variables is not so large, while the function evaluations are expensive and feasibility of iterates is desirable. But for problems with large numbers of variables, FSQP might not be a good fit. The minimax problems with large numbers of objective functions or inequality constraints, such as finely discretized semi-infinite optimization problems, could be handled effectively, for instance, problems involving time or frequency responses of dynamical systems.

The typical constrained minimax problem is showing in eq.(1) and (2).

|  |  |  |
| --- | --- | --- |
|  | minimize |  |

where  is smooth. FSQP generates a sequence  such that  for all  and .  where stands for the number of objective functions . If , .  is a set of points  satisfying the following constraints, as shown in eq.(2).

|  |  |  |
| --- | --- | --- |
|  |  |  |

where  and  are smooth;  stands for the number of boundary constraints;  stands for the number of nonlinear inequality constraints;  stands for the number of linear inequality constraints;  stands for the number of nonlinear equality constraints;  stands for the number of linear equality constraints.  stands for the constraint of lower boundary and  stands for the constraint of upper boundary. , ,  and  stand for the parameters of linear constraints.

# 2. Algorithm

FSQP solves the original problem with nonlinear equality constraints by solving a modified optimization problem with only linear constraints and nonlinear inequality constraints. Four critical points are derived as a sequence in order to solve the problem: 1) a random point; 2) a point fitting all linear constraints; 3) a point fitting all constraints; and 4) an optimal point within all constraints.

In order to take the nonlinear equality constraints into consideration, original optimization problem is switched to the problem in eq.(3). Since the constraints, , the original objective function  is replaced by the modified objective function:

|  |  |  |
| --- | --- | --- |
|  | minimize |  |

where *X* is the set of points . , which are positive penalty parameters that are iteratively adjusted, where. For , replace  by  whenever .

## 2.1 Find an initial point fitting all linear constraints

A feasible point, an approximate Hessian Matrix , the penalty parameters  and the iteration time  are initialized in the first step and then update each time within the loop. For finding the initial feasible point , an initial guess  is randomly picked from the space. If  is infeasible for linear constraints, a strictly convex quadratic program is generated in order find point  that fitting all linear constraints as showing in eq.(4). A new variable  is added in as an adjustment to the variables, where.

|  |  |  |
| --- | --- | --- |
|  | minimize |  |

Then, by adding the nonlinear constraints in, a new Minimax problem is generated which is used to find an initial feasible point, showing in eq.(5).

|  |  |  |
| --- | --- | --- |
|  | minimize |  |

## 2.2 Find a feasible point and an optimal point

Same minimax problem with initial feasible point is needed for both finding a feasible point fitting all constraints  and an optimal point within all constraints . The following four steps are repeated as  converges to the solution: 1) initialization; 2) computation of a search direction; 3) line search; and 4) updates. Fig. 1 below shows how FSQP works for solving the constrained minimax problem.



Fig.1. Process for solving minimax problem with initial feasible point

### 2.2.1 Initialization

 is derived as an initial feasible point.  is an identity matrix.  is the penalty parameters for nonlinear equality constraints, where.  is the iteration times, where .

### 2.2.2 Computation of a search direction

Three steps are used to compute the search direction: 1) compute ; 2) compute ; and 3) compute .  stands for the direction of descent for the objective function and  stands for an arbitrary feasible descent direction.  stands for the feasible descent direction between the directions of  and . Inner relations among the parameters are displayed in Figure 2.



Fig.2. Search direction calculations for the constrained minimax program.

Compute the quadratic programming  for . At each iteration , the quadratic program  that yields the SQP direction  is defined at  for  symmetric positive definite by eq.(6).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Given , following notation is made in eqs.(7) and (8).

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Adding additional variables to transform the standard quadratic programming, showing in eq.(9). Detailed solver information is referred in Appendix B.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Corresponding KKT multipliers can be derived simultaneously, where  stands for the objective functions that scaled at .  stands for the boundary constraints. stands for the nonlinear and linear inequality constraints.  stands for the nonlinear and linear equality constraints. Then, in order to compute , solution of  is derived by solving the strictly convex quadratic program in eq.(10).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Adding additional variables to transform to the standard quadratic programming in eq.(11). Detailed solver information is also referred in Appendix B.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Set  with , where .

### 2.2.3 Line search

Compute , the first number  in the sequence  satisfying both eqs.(12) and (13).

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

where. If , , while if , .

### 2.2.4 Updates

 and  are updated within each loop. Firstly, the new point  is calculated based on eq.(14).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Then, the updating scheme for the Hessian estimates uses BFGS formula with Powell’s modification to compute the new approximation  as the Hessian of the Lagrangian, where 

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

where  stands for the Lagrange function.  and  stands for the KKT multipliers that scale to.

|  |  |
| --- | --- |
|  |  |
|  |  |

A scalar  is then defined by:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Defining  as:

|  |  |  |
| --- | --- | --- |
|  |  |  |

The rank two Hessian update is:

|  |  |  |
| --- | --- | --- |
|  |  |  |

To update the penalty parameter , solve the unconstrained quadratic problem in . Method for solving is referenced in Appendix C.

|  |  |
| --- | --- |
|  |  |

For , update the penalty parameters as

|  |  |  |
| --- | --- | --- |
|  |  |  |

Set 

## 2.3 Check stopping criteria

If  and , iteration stops and return the value .

# 3. Implementation

The program is implemented in Java with Eclipse IDE on my personal desktop. The general framework of classes is shown in Fig.3. In general, the method minimax within MiniMax.java class has been run twice. One is included in Initial.java which is intended to find a feasible point fitting for both the linear and nonlinear constraints. The other is for finding an optimal point within all constraints. Detailed inputs and outputs information can be referred in Appendix D.



Fig.3. General framework of classes for JFSQP.

In order to prove the validity of the java optimization programs listed, testing classes are built correspondingly. As Table 1 indicating below, different testing cases are picked from linear programming, quadratic programming and nonlinear constrained and unconstrained programming.

Table . Testing modules list for JFSQP

|  |  |  |
| --- | --- | --- |
| JFSQP Classes | Test classes | Test method |
| JFSQP.java | testJFSQP.java | LP, QP, NP without constraints, NP with constraints |
| Initial.java | testInitial.java | LP, NP constraints, combination |
| MiniMax.java | testMiniMax.java | Feasible point, NP with constraints |
| Direction\_d0.java |  |  |
| Direction\_d1.java |  |  |
| Linesearch.java |  |  |
| BFGSPowell.java | testBFGS\_Powell.java | Same program checking with Matlab |
| QP.java | testQP.java | QP with linear constraints |
| KKT.java | testKKT.java | linear constraints in QP |
| Check.java |  |  |

Also, in order to solve the minimax problem and test the JFSQP programming, three open-source Java libraries are leveraged showing in Table 2 below.

Table . Java libraries leveraged in JFSQP

|  |  |  |
| --- | --- | --- |
| Libraries | Release date | Usage |
| JAMA | Version 1.0.3; 11/2012 | Provide fundamental operations of numerical linear algebra. Basic arithmetic operations include matrix addition and multiplication, matrix norms and selected element-by-element array operations. |
| Apache Commons Lang | Version 3.3.2; 04/2014 | Provide a host of helper utilities for the java.lang API, notably String manipulation methods, basic numerical methods, object reflection, concurrency, creation and serialization and System properties. |
| JOptimizer | Version 3.3.0; 04/2014 | Provide the solution of a minimization problem with equality and inequality constraints. |

Note: official website for JAMA http://math.nist.gov/javanumerics/jama/; Apache Commons Lang http://commons.apache.org/; JOptimizer http://www.joptimizer.com/

Parameters are set as , , , , , , , , , , .

# 4. Validation and testing

## 4.1 Test module for finding an initial feasible point

Three initial guesses are picked randomly: 1) the first one visually fits the linear constraints; 2) the second one violates the linear constraints; and 3) the third one is far away from the two previous points.

Case 1. Find a feasible point within the boundary and nonlinear inequality constraints:

Subject to: 

Table . Feasible point searching results from different initial guesses

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Initial guess | Objective function | Point fitting all linear constraints | Iterations for  finding feasible point | Feasible point return |
|  | 4.36 |  | 3 |  |
|  | 9.04 |  | 4 |  |
|  | 6914 |  | 2 |  |

Case 2. Find a feasible point within the boundary, the linear and nonlinear inequality constraints:

Subject to: 

Table . Feasible point searching results from different initial guesses

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Initial guess | Objective function | Point fitting all linear constraints | Iterations for  finding feasible point | Feasible point return |
|  | -5 |  | 10 |  |
|  | 7 |  | 7 |  |
|  | 37 |  | 10 |  |

## 4.2 Test JFSQP

Case 3. Minimize 

Subject to: 

The known global solution 

Table . Optimal point searching results from different initial guesses

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Initial guess | Objective function | Point fitting all linear constraints | Iterations for finding feasible point | Feasible point return | Iterations for finding optimal point | Final point return |
|  | 125 |  | 15 |  | 24 |  |
|  | -13,357 |  | 20 |  | 25 |  |
|  | -280,000 |  | 20 |  | 67 |  |

Case 4. Minimize 

Subject to: 

The known global solution ; 

Table . Optimal point searching results from different initial guesses

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Initial guess | Objective function | Point fitting all linear constraints | Iterations for finding feasible point | Feasible point return | Iterations for finding optimal point | Final point return |
|  | 7.2 |  | 3 |  | 5 |  |
|  | 31.36 |  | 2 |  | 5 |  |
|  | 13.25 |  | 2 |  | 5 |  |

Fig.4 below shows the optimal point searching process from the three different initial guesses. As all the above test cases indicating, the different initial guesses might generate different points fitting all linear constraints, but soon they all reach to very close feasible points fitting both linear and nonlinear constraints, and finally search for the optimal point.

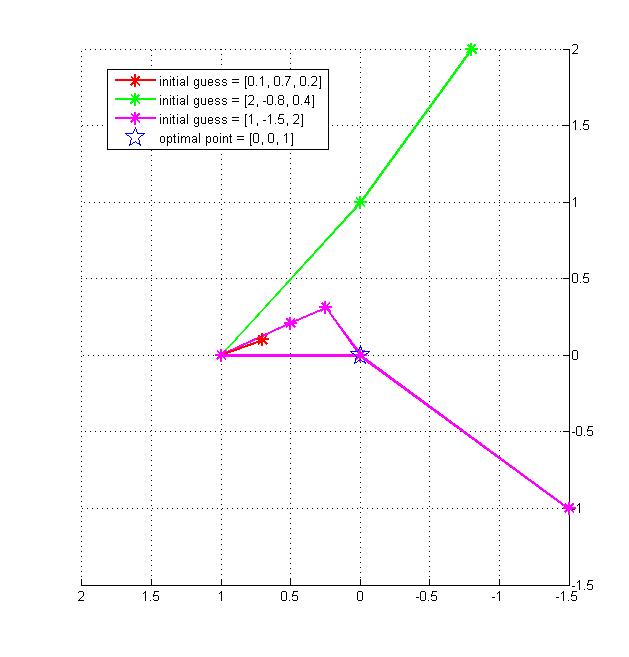
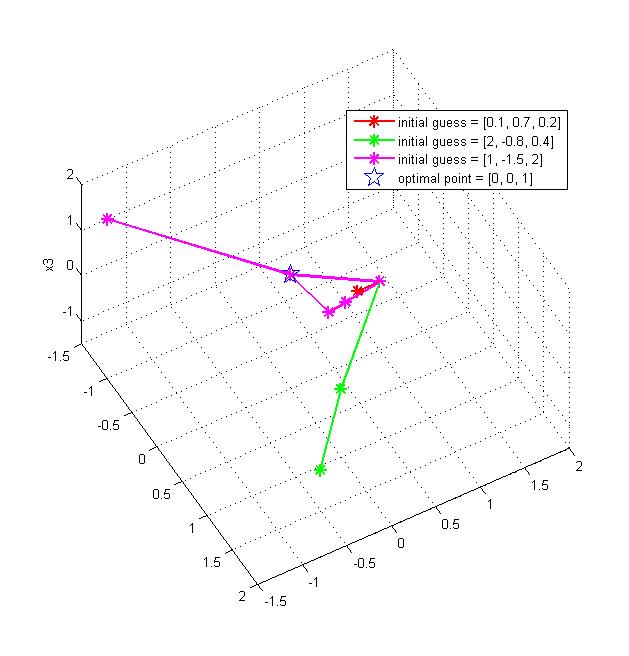
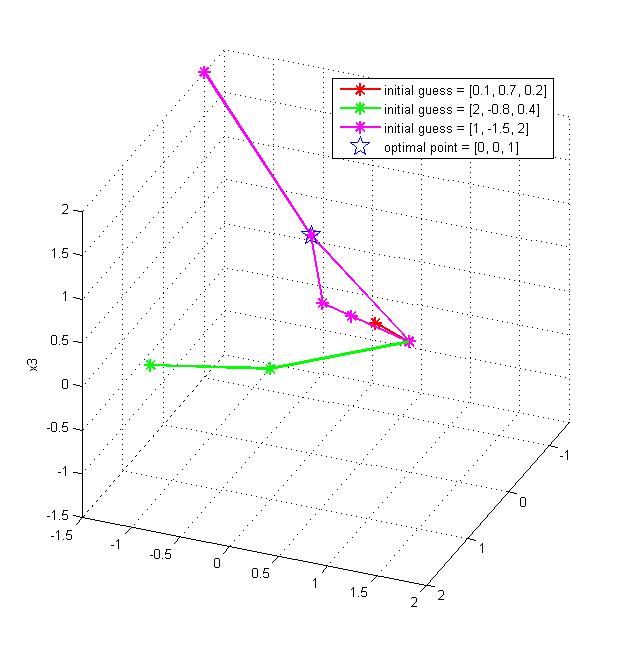
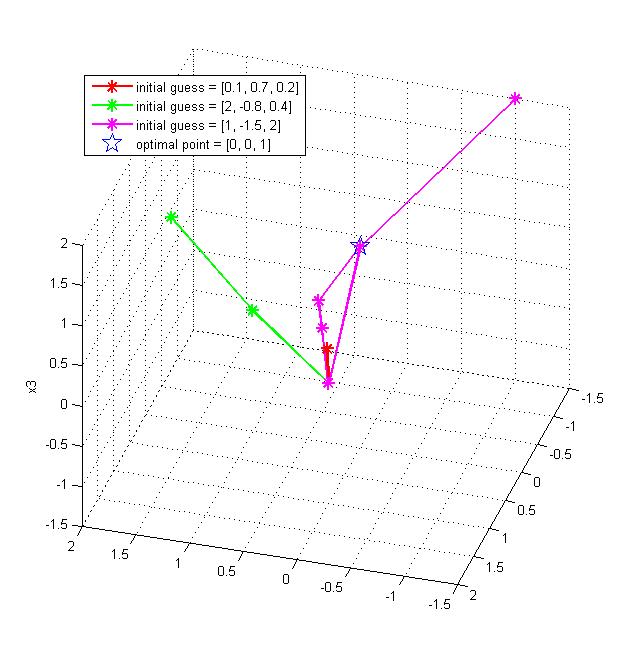


Fig.4. Optimal point searching process from three different initial guesses.

# 5. Conclusions

Several conclusions can be addressed based on the project:

1. FSQP switch the complicated minimax optimization problem into sequential quadratic programs with both linear and nonlinear, equality and inequality constraints.
2. Critical points during searching sequence follows an initial random guess, a point fitting all linear constraints, a feasible point fitting all constraints, and an optimal point within all constraints.
3. Different initial guesses would still lead to the close initial feasible points that are used as the input for final optimal point search.
4. More iteration times would take for finding an optimal point than finding a feasible point fitting all listed constraints.

# 6. Further work

Further work can be done through the following aspects:

1. Test the sensitivity of different parameters setting in JFSQP;
2. Implement the arch search variable  and compare efficiency;
3. Develop user interface.

# 7. Schedule

|  |  |  |
| --- | --- | --- |
| Planed schedule | Tasks | Actual finish time |
| October | * Literature review; * Specify the implementation module details; * Structure the implementation; | October  November  October |
| November | * Develop the quadratic programming module; * Unconstrained quadratic program; * Strictly convex quadratic program; * Validate the quadratic programming module; | November  November  December  December |
| December | * Develop the Gradient and Hessian matrix calculation module; * Validate the Gradient and Hessian matrix calculation module; * Midterm project report and presentation; | December  January  December |
| January | * Develop Armijo line search module; * Validate Armijo line search module; | January  January |
| February | * Develop the feasible initial point module; * Validate the feasible initial point module; * Integrate the program; | March  April  April |
| March | * Debug and document the program; * Validate and test the program with case application; | April & May  April |
| April | * Add arch search variable  in; * Compare calculation efficiency of line search with arch search methods; | (Undone)  (Undone) |
| May | * Develop the user interface if time available; * Final project report and presentation; | (Undone)  May |

# 8. Deliverables

* Project proposal;
* Well-documented Java codes;
* Test cases;
* Validation results;
* Project reports;
* Presentations.

# 9. Bibliography

Charalambous, Conn and AR Conn. "An Efficient Method to Solve the Minimax Problem Directly." *SIAM Journal on Numerical Analysis* 15, no. 1 (1978): 162-187.

Goldfarb, Donald and Ashok Idnani. "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs." *Mathematical programming* 27, no. 1 (1983): 1-33.

Lawrence, Craig T and André L Tits. "Nonlinear Equality Constraints in Feasible Sequential Quadratic Programming∗." *Optimization Methods and Software* 6, no. 4 (1996): 265-282.

Lawrence, Craig T and André L Tits. "A Computationally Efficient Feasible Sequential Quadratic Programming Algorithm." *Siam Journal on optimization* 11, no. 4 (2001): 1092-1118.

Li, Wu and John Swetits. "A New Algorithm for Solving Strictly Convex Quadratic Programs." *SIAM Journal on Optimization* 7, no. 3 (1997): 595-619.

Madsen, Kaj and Hans Schjaer-Jacobsen. "Linearly Constrained Minimax Optimization." *Mathematical Programming* 14, no. 1 (1978): 208-223.

Mayne, DQ and E Polak. "Feasible Directions Algorithms for Optimization Problems with Equality and Inequality Constraints." *Mathematical Programming* 11, no. 1 (1976): 67-80.

Watson, GA. "The Minimax Solution of an Overdetermined System of Non-Linear Equations." *IMA Journal of Applied Mathematics* 23, no. 2 (1979): 167-180.

# Appendix A – Nomenclature

|  |  |  |
| --- | --- | --- |
| Symbols |  | Interpretation |
|  |  | Objective functions |
|  |  | Number of objective functions |
|  |  | Inequality constraints function |
|  |  | Equality constraints function |
|  |  | Number of nonlinear inequality constraints |
|  |  | Number of linear inequality constraints |
|  |  | Number of nonlinear equality constraints |
|  |  | Number of linear equality constraints |
|  |  | Lower boundary of the variables |
|  |  | Upper boundary of the variables |
|  |  | Parameters of linear equality constraints |
|  |  | Parameters of linear equality constraints |
|  |  | Parameters of linear inequality constraints |
|  |  | Parameters of linear inequality constraints |
|  |  | A random point |
|  |  | A point fitting all linear constraints |
|  |  | An initial feasible point |
|  |  | An initial Hessian matrix as identity matrix |
|  |  | Feasible point in *k* iteration |
|  |  | Hessian matrix in *k* iteration |
|  |  | Positive penalty parameters in *k* iteration |
|  |  | Iteration index |
|  |  | Descent direction for the objective function in *k* iteration |
|  |  | An arbitrary feasible descent direction in *k* iteration |
|  |  | A feasible descent direction between the directions of  and  in *k* iteration |
|  |  | Weight between directions of  and  in *k* iteration |

|  |  |
| --- | --- |
| Symbols | Values |
|  | 0.1 |
|  | 0.01 |
|  | 0.1 |
|  | 0.5 |
|  | 2.1 |
|  | 2.5 |
|  | 2.5 |
|  | 0.1 |
|  | 1 |
|  | 2 |
|  | 2 |

# Appendix B – Simplex method for strictly convex quadratic programming

Quadratic programming is the problem of optimizing a quadratic function of several variables subject to linear constraints on these variables. Consider a convex quadratic programming problem of the following form, minimizing with respect to the vector *x* of the following function showing in eq.(1).

|  |  |  |
| --- | --- | --- |
|  |  |  |

where *n* stands for the number of variables.  is a column of  variables, .  is a  symmetric positive definite matrix that . is a *n* dimension column vector. Subject to the linear inequality constraints as showing in eqs.(2) and (3).

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

where *m* stands for the number of constraints. *A* is an  matrix. is a given column vector of *m* elements. All the variables are constraint to be nonnegative. Combining the objective function as well as the constraints, the Lagrangian function for the quadratic program is formed as the following expression eq.(4).

|  |  |  |
| --- | --- | --- |
|  |  |  |

where  is the Lagrangian multipliers as a column vector of *m* dimensional, . Karush-Kuhn-Tucker conditions are first order necessary conditions for a solution in nonlinear programming to be optimal, provided that the following regularity conditions eqs.(5)~(8) are satisfied.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | then |  |
|  |  |  |
|  |  |  |

Thinking to solve the problem, slackness variables have been added in to switch the inequality constraints to equality constraints. As a result,  is the  slackness matrix for constraints and  is the  slackness matrix for objective functions. Then, the following eqs.(9)~(12) have been derived.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

where all the variables are nonnegative, , , , and . Eqs.(13)~(16) are derived based on eqs.(9)~(12), which works as an equivalent problem of the original quadratic programming to be solved.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

To solve the above problem, artificial variables  are generated for the initial basis composition.  and , as well as  and  are complementary slack variables. As a result, another coefficient vector as the Lagrangian multipliers is given in the objective function. The complementary slackness conditions are satisfied in nature. Then, the problem becomes the following optimization problem with the objective function of eq.(17) and the constraints of eqs.(18)~(20).

|  |  |  |
| --- | --- | --- |
|  | Minimize |  |
|  | Subject to |  |
|  |  |  |
|  |  |  |

where is an *n* dimension column vector. Here we set the coefficient of to be 1. Then,  is also an *n* dimension column vector that has been adjusted as the coefficient for accordingly.

In order to solve the modified quadratic programming problem, the Wolfe’s extended simplex method has been chosen which do not requires the gradient and Hessian computation. Accordingly, three major steps are taken as: 1) set up the initial tableau; 2) repeat the loop until the optimal solution found; and 3) return the final solution. Details are explained as below.

**Step 1. Set up the initial tableau**

According to the constraints eqs.(18) and (19), Table 1 has been set up for further computation. Variables  and are chosen as the initial bases.

Table B1. Step-up the initial tableau for the Simplex method

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Basic variables | Values basic variables  (1) | (n) | (m) | (n) | (m) | (n) |
| (m) | *b* | *A* |  |  | *I* |  |
| (n) | *- d* | *C* | *A'* | *- I* |  | *I* |

Since feasible values of  and  have not been reached yet, the following coefficients are set as a start for calculation simplification.

Table B2. Step-up the initial price value for the Simplex method

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variables |  |  |  |  |  |
| Values | 0 | 0 | 0 | 0 |  |

**Step 2. Repeat the loop until the optimal solution found**

There are three sub-steps to be conducted within the loop as: 1) search for pivot element; 2) perform pivoting; and 3) update price. When each time a new feasible solution is found through pivoting, the price table for variables is updated, in order to check whether the new combination could help to reduce the value of the objective function. The pivot element is the new variable selected to enter the basis, while the previous one is the one to leave. Two rules are followed by searching for the pivot element in the loop.

* *Rule 1. Search for the pivot column. Find the largest negative indicator in the price table after canonical transformation.*
* *Rule 2. Search for the pivot row. Find the smallest nonnegative indicator for the ratio.*

**Step 3. Return the final solution**

For the loop iteration, the stopping criteria is when the variable changes could no longer help to reduce the value of the objective function , which means the updated price values after canonical transformation are all positive.

In order to demonstrate the feasibility for solving quadratic programming problem, the following test case has been conducted:

|  |  |  |
| --- | --- | --- |
|  | Minimize |  |
|  | Subject to |  |
|  |  |  |
|  |  |  |

Accordingly, the input parameters are set asthe number of constraints *m* = 2; number of variables *n* = 2; and, , , , . Initial set-up tableau and price value have been set as Table 3 and 4 indicating respectively.

Table B3. Application of the simplex method for quadratic programming

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Basic variables | Values basic variables |  |  |  |  |  |  |  |  |  |  |
|  | 6.0 | 2.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
|  | 0.0 | 1.0 | -4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
|  | 4.0 | 2.0 | -2.0 | 2.0 | 1.0 | -1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
|  | 0.0 | -2.0 | 4.0 | 1.0 | -4.0 | 0.0 | -1.0 | 0.0 | 0.0 | 0.0 | 1.0 |

Table B4. Step-up the initial price value for the Simplex method

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Variables |  |  |  |  |  |  |  |  |  |  |
| Values | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |

Switching Table 4 to canonical form, the price value is shown as Table 5 below.

Table B5. Price value after canonical transformation

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Variables |  |  |  |  |  |  |  |  |  |  |
| Value | 0.0 | -2.0 | -3.0 | 3.0 | 1.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |

As a result, variable  is chosen as the pivot variable indicating the pivot column. The pivot row is selected from the smallest nonnegative elements of . Then, the pivot element based on the column and row is chosen to be 1.0. Accordingly, pivoting has been conducted and Table 6 is set as follows.

Table B6. Application of the simplex method for quadratic programming

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Basic variables | Values basic variables |  |  |  |  |  |  |  |  |  |  |
|  | 6.0 | 2.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
|  | 0.0 | 1.0 | -4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
|  | 4.0 | 6.0 | -10.0 | 0.0 | 9.0 | -1.0 | 0.0 | 0.0 | 0.0 | 1.0 | -2.0 |
|  | 0.0 | -2.0 | 4.0 | 1.0 | -4.0 | 0.0 | -1.0 | 0.0 | 0.0 | 0.0 | 1.0 |

From Table 6, , ,  and  built a feasible solution. Then the price values of the corresponding complementary variables are set. For instance, the complementary variable of  is . The updated price value for  is 6.0. Table 7 below indicates the details.

Table B7. Price value after canonical transformation

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Variables |  |  |  |  |  |  |  |  |  |  |
| Value | 0.0 | 0.0 | 6.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |

Repeating the process, the following tableau updating is demonstrated in Table 8.

Table B8. Application of the simplex method for quadratic programming

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iteration | Basic variables | Values basic variables |  |  |  |  |  |  |  |  |  |  |
| 3 |  | 6.0 | 2.5 | 0.0 | -0.25 | 1.0 | 0.0 | 0.25 | 1.0 | 0.0 | 0.0 | -0.25 |
|  | 0.0 | -1.0 | 0.0 | 1.0 | -4.0 | 0.0 | -1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
|  | 4.0 | 1.0 | 0.0 | 2.5 | -1.0 | -1.0 | -0.5 | 0.0 | 0.0 | 1.0 | 0.5 |
|  | 0.0 | -0.5 | 1.0 | 0.25 | -1.0 | 0.0 | -0.25 | 0.0 | 0.0 | 0.0 | 0.25 |
| 4 |  | 2.4 | 1.0 | 0.0 | -0.1 | 0.4 | 0.0 | 0.1 | 0.4 | 0.0 | 0.0 | -0.1 |
|  | 2.4 | 0.0 | 0.0 | 0.9 | -3.6 | 0.0 | -0.9 | 0.4 | 1.0 | 0.0 | 0.9 |
|  | 1.6 | 0.0 | 0.0 | 2.6 | -1.4 | -1.0 | -0.6 | -0.4 | 0.0 | 1.0 | 0.6 |
|  | 1.2 | 0.0 | 1.0 | 0.2 | -0.8 | 0.0 | -0.2 | 0.2 | 0.0 | 0.0 | 0.2 |
| 5 |  | 2.46 | 1.0 | 0.0 | 0.0 | 0.35 | -0.04 | 0.08 | 0.38 | 0.0 | 0.04 | -0.08 |
|  | 1.85 | 0.0 | 0.0 | 0.0 | -3.12 | 0.35 | -0.69 | 0.54 | 1.0 | -0.35 | 0.69 |
|  | 0.62 | 0.0 | 0.0 | 1.0 | -0.54 | -0.38 | -0.23 | -0.15 | 0.0 | 0.38 | 0.23 |
|  | 1.08 | 0.0 | 1.0 | 0.0 | -0.69 | 0.08 | -0.15 | 0.23 | 0.0 | -0.08 | 0.15 |

Finally, optimal solutions are reached in the fifth iteration that , , .

Reference:

Wolfe, Philip. "The simplex method for quadratic programming." *Econometrica: Journal of the Econometric Society* (1959): 382-398.

Van de Panne, Cornelis, and Andrew Whinston. "The simplex and the dual method for quadratic programming." *OR* (1964): 355-388.

# Appendix C – Solving unconstrained quadratic programming

Solve the similar problem as:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Gradient would be:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Pseudo inverse:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

# Appendix D – Inputs and outputs for JFSQP class in program

|  |  |  |
| --- | --- | --- |
| Class | Inputs | Outputs |
| JFSQP.java | A random point;  Number of variables;  Objective functions;  Number of objective functions;  Boundary constraints;  Number of boundary constraints;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints;  Linear equality constraints;  Number of linear equality constraints; | An optimal point satisfying all listed constraints |
| Initial.java | A random point;  Number of variables;  Boundary constraints;  Number of boundary constraints;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints;  Linear equality constraints;  Number of linear equality constraints; | A feasible point fitting all constraints |
| MiniMax.java | An initial feasible point;  Number of variables;  Objective functions;  Number of objective functions;  Boundary constraints;  Number of boundary constraints;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints;  Linear equality constraints;  Number of linear equality constraints; | An optimal point fitting all listed constraints |
| Direction\_d0.java | Current point;  Number of variables;  Approximated Hessian Matrix;  Penalty parameters for nonlinear equality constraints;  Objective functions;  Number of objective functions;  Boundary constraints;  Number of boundary constraints;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints;  Linear equality constraints;  Number of linear equality constraints; | A vector of the direction d0 |
| Direction\_d1.java | Current point;  Number of variables;  Approximated Hessian Matrix;  Penalty parameters for nonlinear equality constraints;  Objective functions;  Number of objective functions;  Boundary constraints;  Number of boundary constraints;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints;  Linear equality constraints;  Number of linear equality constraints; | A vector of the direction d1 |
| Linesearch.java | Current point;  Number of variables;  Direction;  Penalty parameters for nonlinear equality constraints;  Objective functions;  Number of objective functions;  Boundary constraints;  Number of boundary constraints;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints;  Linear equality constraints;  Number of linear equality constraints; | A distance |
| BFGSPowell.java | Current point;  Updated point;  Approximated Hessian Matrix;  Objective functions;  Number of objective functions;  Multiplier of objective functions;  Boundary constraints;  Number of boundary constraints;  Multiplier of boundary constraints;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Multiplier of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Multiplier of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints;  Multiplier of linear inequality constraints;  Linear equality constraints;  Number of linear equality constraints;  Multiplier of linear equality constraints; | An updated Hessian Matrix |
| KKT.java | Derived direction d0;  Number of variables;  Quadratic part of the objective function;  Linear part of the objective function;  Variable coefficient of inequality constraints;  Coefficient of inequality constraints;  Variable coefficient of equality constraints;  Objective functions;  Number of objective functions;  Boundary constraints;  Number of boundary constraints;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints;  Linear equality constraints;  Number of linear equality constraints; | A multiplier vector fitting for all constraints |
| Check.java | Current point;  Nonlinear inequality constraints;  Number of nonlinear inequality constraints;  Nonlinear equality constraints;  Number of nonlinear equality constraints;  Linear inequality constraints;  Number of linear inequality constraints; | 0 - all constraints are satisfied  1 - at least one of the constraints are not satisfied |