Preface

This book is about Quadratic Programming (QP), Parametric Quadratic Programming (PQP), the theory of these two optimization problems, and solution algorithms for them. Throughout this text, we rely heavily on vector and matrix notation. Prime ($'$) will be used to denote matrix transposition. All unprimed vectors will be column vectors. The i–th component of the vector x_0 will be denoted by $(x_0)_i$. In a numerical example, we sometimes use the symbol times ($\times$) to denote multiplication. For 2–dimensional examples, we generally use the problem variables x_1 and x_2 with $x = (x_1, x_2)'$. In other situations, x_1 and x_2 may denote two n–vectors. The meaning should be clear from the context. The end of a proof is signified by an unfilled box ($\Box$) and the end of an example is signified by an unfilled diamond ($\Diamond$).

In Chapter 1, we consider several 2–dimensional examples of QP's. We teach the reader how to draw these problems and how to deduce an optimal solution using graphical arguments. We also deduce from the graphical representation algebraic conditions which are both necessary and sufficient for optimality.

Chapter 2 is devoted to the development of portfolio optimization. Portfolio optimization is concerned with combining a number of assets, each with known expected return and correlation with the other assets into a single portfolio. For the combined portfolio, with any level of expected return, no other portfolio should have a smaller variance. Such a portfolio is called efficient. Finding such a portfolio is equivalent to solving a certain QP. Finding all such portfolios is equivalent to solving a PQP. Thus Chapter 2 provides many examples of problems which are naturally QP's or PQP's.

Chapter 3 begins by looking at the problem of unconstrained quadratic minimization theory then development of a solution algorithm for it using conjugate directions (Algorithm 1). Although this problem can be solved directly, we use a conjugate direction method to solve it because it will be a building block for full QP algorithms developed subsequently. Chapter 3 continues by addressing the problem of quadratic minimization subject to linear equality constraints. Theoretical properties and a solution algorithm (Algorithm 2) are developed as generalizations of those for unconstrained quadratic minimization.

Whenever we develop an algorithm in Chapter 3 or subsequent chapters, we first give the informal development of the algorithm. Then we give a com-

plete and detailed formulation and prove its properties. We also give several numerical examples which are illustrated with detailed calculations. Then in a later section, a Matlab program is given to implement the algorithm. These programs are included on the accompanying CD. This computer program is run on the same data as the example to validate its results. In some cases, we illustrate the computer program by solving some problems which are too large for hand computation.

Chapter 4 is devoted to the theory of QP. Optimality conditions which are both necessary and sufficient for optimality are developed. Duality theory for QP may be regarded as a generalization of duality theory for LP. Indeed, the two types of duality share many of the same properties. All of the QP algorithms automatically compute an optimal solution for the dual. In some situations, obtaining the optimal solution for the dual may be equally important as finding the optimal solution for the given problem (the primal).

Sometimes the data for a QP or PQP is taken from physical measurements and is only known to a certain accuracy. In this case it is important to know how sensitive the (approximate) optimal solution is to small changes in the problem data. This area is called sensitivity analysis and is also developed in Chapter 4.

In Chapter 5, we generalize Algorithm 2 to solve a QP having both linear inequality and equality constraints. The resulting algorithm (Algorithm 3) will solve a QP provided a certain nondegeneracy assumption is satisfied. Algorithm 3 will also find an optimal solution for the dual QP. An initial feasible point is required to start Algorithm 3. We solve this problem by formulating an initial point problem (an LP) whose optimal solution will be feasible for the original problem. We also specialize Algorithm 3 to solve an LP (LP Algorithm 3) and this can be used to efficiently solve the initial point problem.

Chapter 5 continues by noting that when a new constraint becomes active, all of the conjugate direction built up by Algorithm 3 are destroyed and must be rebuilt. A Householder transformation is introduced which when applied to the existing conjugate directions will result in a new set of conjugate directions which will not be destroyed (Algorithm 4) and this is a very efficient and fast QP algorithm.

Both algorithms require a nondegeneracy assumption to guarantee finite termination. We complete Chapter 5 by giving a generalization of Bland's rules which ensures finite termination if degeneracy is encountered. Computer programs for Algorithms 3 and 4 are given.

Most QP algorithms are based on satisfying primal feasibility and complementary slackness at each iteration and iteratively working toward satisfying dual feasibility. In Chapter 6, we present a dual QP algorithm (Algorithm 5) which satisfies dual feasibility and complementary slackness at each iteration and iteratively works to satisfy primal feasibility. An example of where this is appropriate is the following. Suppose a QP has just been solved. But then it is realized that additional primal constraints are necessary. So the previously optimal solution satisfies dual feasibility and complementary slackness and so can be efficiently solved using a dual QP method.

In Chapter 7, we formulate general QP algorithms (Algorithm 6) and PQP algorithms (Algorithm 7). These methods are general in the sense that the method of solving certain linear equations is left unspecified. The resulting method just contains the QP and PQP logic. The linear equations may be sparse or contain structure like flow constraints. This structure may be utilized to formulate a more efficient solution method. One way to formulate an implementable version of these methods is to solve the relevant linear equations using the inverse of the coefficient matrix. Then there are 3 possible types of updates: add a row and column, delete a row and column and exchange a row and column. These 3 symmetric inverse updates are developed in detail.

The Simplex Method (Algorithm 8) and Parametric Simplex Method (Algorithm 9) are developed in Chapter 8. They both assume that the problem constraint structures are of the form $\{ x \mid Ax = b,\ x \geq 0\}$, which is identical to that of the Simplex Method for LP. The efficiency of this method is a consequence of that if the intermediate points constructed have many variables at zero, then the linear equations that must be solved are very small in number. This would be the case if the Hessian matrix for the problem has low rank.

In Chapter 9, we consider the problem of nonconvex quadratic programming. This type of problem may possess many local minima, a global minimum or be unbounded from below. We show that the first order Karush–Kuhn–Tucker conditions are necessary for a local minimum but not in general sufficient. We then develop the second order sufficiency conditions which with the first order necessary conditions will guarantee that a point is a strong local minimum.

With additional steps in Algorithm 4, we formulate a new method (Algorithm 10) which will determine a strong or weak local minimum (if such a point exists), or determine that the problem is unbounded from below.

Using the Included Matlab Programs

Every Matlab program shown in this book is included in the accompanying CD. A good way to access these programs is to first make a new folder and call it "MYQP". Then copy the entire contents of the CD into "MYQP". The following instructions apply to MATLAB R2016a but other releases of MATLAB should be similar. Start Matlab and press the "HOME" button and then the "Set Path" button on the "HOME" page toolbar. This brings up a window entitled "Set Path". Press "Add Folder..." and keep browsing until you locate the "MYQP" folder. Highlight "MYQP" and then press the "Select Folder". This will close the "Add Folder..." window and return you to the "Set Path" window. At the top of this window you will see the correct path to "MYQP". Now press "Save" and then "Close". All of this sets Matlab's path pointing to "MYQP". This only needs to be done once. Matlab will remember this setting when you restart Matlab.

Next, suppose you want to run eg3p3.m (Figure 3.8). Press the "EDITOR" button to show the drop down menu. Then press the "OPEN" button and then the second "Open" button. This will produce a new window containing all of the Matlab programs in the book. Scroll through the names until you come to eg3p3. Highlight this line and press "Open". The program eg3p3.m will now appear in the "Editor" window. The program may now be run by pressing the "EDITOR" button and pressing the "Run" button (green triangle icon). The output will appear in the "Command Window". Note that eg3p3.m references the functions checkdata, Alg1 and checkopt1. These will be loaded automatically because they are in the "MYQP" directory.

Next suppose we want to modify the data for eg3p3. It is good practice to make a copy of eg3p3.m. The original version of eg3p3.m thus remains intact and a copy of it can be made as follows. Press "EDITOR", then "New" then "Script". This will open a new empty window entitled "Untitledj" where j is some number. Press the tab eg3p3.m which will open the file eg3p3.m, highlight the contents (press Ctrl plus a), press Ctrl plus Ins, then open "Untitledj" and press Shift plus Ins. A copy of eg3p3.m will then appear in the window "Untitledj". Suppose we want to see the effect of changing c(2) from its present value of -8 to +8. In line 3 of the "Untitledj" window, just change the -8 to 8. Press the "EDITOR" button, choose "Save" then "Save As..." from the drop down menu. Enter some name like eg3p3mod.m. Now the modified example can be run by choosing "Editor", then "Run" as

above.

Another useful thing is to show the intermediate calculations of an algorithm. Suppose we wish to see the intermediate values of "x" in the iterations of Algorithm 1. Open Alg1.m and observe that "x" is updated on line 49. Remove the semi colon (;) from the end of that line, save the modified Alg1, open an example for Alg1.m and run it. Every time line 49 is executed, the new value of "x" will be printed. This can be done for any variable in Alg3.m.