Methods for Constrained Optimization

Shuonan Dong
18.086 Project 2
Spring 2006

1 Introduction

This discussion focuses on the constrained optimization problem and looks into different methods for solving it. Constrained optimization is approached somewhat differently from unconstrained optimization because the goal is not to find the global optima. Often, constrained optimization methods use unconstrained optimization as a sub-step. In this paper, I first set up the constrained optimization problem, introduce several optimization methods, and apply it to a toy problem. Then I introduce the real-world application of data classification, which utilizes constrained optimization. I apply the applicable methods to a 2-D classification problem before demonstrating the full capability on handwritten numeral identification using real data.

2 Constrained Optimization Problem

The standard form of the constrained optimization problem is as follows:

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && g_j(\mathbf{x}) \le 0, \quad i = 1, \ldots, p \\ & && h_i(\mathbf{x}) = 0, \quad j = 1, \ldots, m \end{aligned}$$

where $\mathbf{x}$ has dimensions $n \times 1$, $f(\mathbf{x})$ is the objective function to be minimized, $\mathbf{g}(\mathbf{x})$ are a set of inequality constraints, and $\mathbf{h}(\mathbf{x})$ are a set of equality constraints. Inequality constraints of the form $\hat{\mathbf{g}}(\mathbf{x}) \ge 0$ can be rewritten as $\mathbf{g}(\mathbf{x}) = -\hat{\mathbf{g}}(\mathbf{x})$. To solve numerically and for ease of discussion, I restrict the functions $f(\mathbf{x})$, $\mathbf{g}(\mathbf{x})$, and $\mathbf{h}(\mathbf{x})$ to the following standard forms:

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c \\ \mathbf{g}(\mathbf{x}) &= \mathbf{D}\mathbf{x} - \mathbf{e} \\ \mathbf{h}(\mathbf{x}) &= \mathbf{C}\mathbf{x} - \mathbf{d} \end{aligned}$$

where $\mathbf{A}$ is an $n \times n$ matrix, $\mathbf{b}$ is an $n \times 1$ vector, c is a scalar, $\mathbf{D}$ is a $p \times n$ matrix, $\mathbf{e}$ is a $p \times 1$ vector, $\mathbf{C}$ is an $m \times n$ matrix, and $\mathbf{d}$ is an $m \times 1$ vector. These restrictions imply that I will only be considering quadratic objective functions with linear constraints. Some of the solution algorithms I will be discussing can also handle other types of objective functions and constraints, but for comparison, I imposed the above restrictions on all methods.

3 Methods for Solving Constrained Optimization Problems

Of the numerous solution methods for constrained optimization problems, I only discuss and implement a few. The methods that I have implemented include: the penalty function method, Lagrange multiplier method, augmented Lagrange multiplier for inequality constraints, quadratic programming, gradient projection method for equality constraints, and gradient projection for inequality constraints.

3.1 Penalty Function Method

The penalty function method can solve optimization problems with both equality and inequality constraints:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & g_j(\mathbf{x}) \le 0, \quad i = 1, \ldots, p \\ & h_i(\mathbf{x}) = 0, \quad j = 1, \ldots, m\,. \end{array}$$

The penalty function method applies an unconstrained optimization algorithm to a penalty function formulation of a constrained problem. The unconstrained optimization algorithm that I used is the `fminsearch` function in MATLAB, which applies the simplex search method of [Lagarias 1998]. The penalty function method as described in [Snyman 2005] is as follows:

$$\text{minimize} \quad P(\mathbf{x})$$

where

$$P(\mathbf{x}, \rho, \beta) = f(\mathbf{x}) + \sum_{j=1}^{m} \rho h_j^2(\mathbf{x}) + \sum_{j=1}^{p} \beta_j g_j^2(\mathbf{x})\,.$$

The penalty parameters ρ_j and β_j are given by

$$\rho >> 0$$

$$\beta_j = \begin{cases} 0 & if \;\; g_j(\mathbf{x}) \le 0 \\ \rho >> 0 & if \;\; g_j(\mathbf{x}) > 0 \end{cases}$$

Now the problem is formulated as unconstrained optimization. However, we cannot solve directly because large values of ρ can cause instability and inefficiency when deriving a solution with high accuracy. We can use the sequential unconstrained minimization technique (SUMT) to incrementally increase the penalty parameter as we derive the solution incrementally. The SUMT algorithm that I implemented is as follows:

1. Choose tolerances $\varepsilon_1 = \varepsilon_2 = 10^{-5}$, starting point $\mathbf{x}_0 = \mathbf{0}$, initial penalty parameter $\rho_0 = 1$.

2. Perform unconstrained optimization (`fminsearch`) on the penalty function $P(\mathbf{x}_0, \rho_k)$ to get $\mathbf{x}_k^*$.
3. Check the convergence criteria. If $\left\|\mathbf{x}_k^* - \mathbf{x}_{k-1}^*\right\| < \varepsilon_1$ and $\left|P(\mathbf{x}_k^*) - P(\mathbf{x}_{k-1}^*)\right| < \varepsilon_2$, then stop. Otherwise, set $\rho_{k+1} = 10\rho_k$, $\mathbf{x}_0 = \mathbf{x}_k^*$, and return to Step 2.

3.2 Lagrange Multiplier

The Lagrange multiplier method can solve optimization problems with equality constraints:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & h_i(\mathbf{x}) = 0, \quad j = 1, \ldots, m < n\,. \end{array}$$

The Lagrangian function is as follows:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{j=1}^{m} \lambda_j h_j(\mathbf{x}) = f(\mathbf{x}) + \lambda^T \mathbf{h}(\mathbf{x}),$$

where λ is an $m \times 1$ vector of Lagrange multipliers, one for each constraint. In general, we can set the partial derivatives to zero to find the minimum:

$$\frac{\partial L}{\partial x_i}(\mathbf{x}^*, \lambda^*) = 0, \quad i = 1, \ldots, n$$
$$\frac{\partial L}{\partial \lambda_j}(\mathbf{x}^*, \lambda^*) = 0, \quad j = 1, \ldots, m$$

where $\mathbf{x}^*$ is the minimum solution and λ^* is the set of associated Lagrange multipliers. In the case of quadratic objective function and linear constraints, we can directly find the partial derivatives:

$$L(\mathbf{x}, \lambda) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c + \lambda^T (\mathbf{C}\mathbf{x} - \mathbf{d})$$
$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b} + \mathbf{C}^T \lambda = 0$$
$$\frac{\partial L}{\partial \lambda} = \mathbf{C}\mathbf{x} - \mathbf{d} = 0$$

which produces the following linear system which can be solved directly:

$$\begin{bmatrix} \mathbf{A} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -\mathbf{b}^* \\ \mathbf{d} \end{bmatrix}.$$

3.3 Augmented Lagrange for Inequality Constraints

The augmented Lagrange method combines the classical Lagrange method with the penalty function method. I used the augmented Lagrange method to tackle inequality constraints for the problem:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & g_j(\mathbf{x}) \leq 0, \quad i = 1, \ldots, p \end{array}$$

One possible augmented Lagrangian function given by [Snyman 2005] is

$$L(\mathbf{x}, \lambda, \rho) = f(\mathbf{x}) + \sum_{i=1}^{p} \left[\max\left(\tfrac{1}{2}\lambda_i + \rho g_i(\mathbf{x}), 0\right)\right]^2$$

where λ are the Lagrange multipliers and ρ is an adjustable penalty parameter. [Greig 1980] gives a slightly different formulation, but retains the same concept. A comparison of the partial derivatives of the augmented Lagrange and the classical Lagrange functions produces the following iterative approximation for λ^*:

$$\lambda^* \cong \lambda_{k+1} = \max\left(\lambda_k + 2\rho_k \mathbf{g}(\mathbf{x}_k^*), 0\right).$$

The algorithm that I implemented is as follows:

1. Choose tolerance $\varepsilon = 10^{-5}$, starting point $\mathbf{x}_0 = \mathbf{0}$, initial penalty parameter $\rho_0 = 1$, and initial Lagrange multipliers $\lambda_0 = \mathbf{0}$.
2. Perform unconstrained optimization (`fminsearch`) on the augmented Lagrangian function $L(\mathbf{x}_0, \lambda_k, \rho_k)$ to get $\mathbf{x}_k^*$.
3. Update $\lambda_{k+1} = \max\left(\lambda_k + 2\rho_k \mathbf{g}(\mathbf{x}_k^*), 0\right)$
4. Increase $\rho_{k+1} = 2\rho_k$ if $\left\|\lambda_k - \lambda_{k-1}\right\| < 0.5$
5. Check the convergence criteria. If $\left\|\mathbf{x}_k^* - \mathbf{x}_{k-1}^*\right\| < \varepsilon$, then stop. Otherwise, set $\mathbf{x}_0 = \mathbf{x}_k^*$ and return to Step 2.

3.4 Quadratic Programming

Quadratic programming (QP) can solve optimization problems with a quadratic objective function and linear constraints:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) = \dfrac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{b}^T \mathbf{x} + c \\ \text{subject to} & \mathbf{D}\mathbf{x} - \mathbf{e} \leq 0 \\ & \mathbf{C}\mathbf{x} - \mathbf{d} = 0\,. \end{array}$$

To derive the active set of inequality constraints, I use the method of Theil and Van de Panne [Snyman 2005], which iterates through all of the combinations of the inequality constraints to find the active set. The QP algorithm that I implemented is as follows:

1. Perform optimization with equality constraints using the Lagrange multiplier method to get $\mathbf{x}_0$.
2. Check if $\mathbf{x}_0$ satisfies the inequality constraints. If it does, we're done, i.e. $\mathbf{x}_{\min} = \mathbf{x}_0$. If not, continue.
3. Use the method of Theil and Van de Panne as discussed in [Snyman 2005], pg 78: find all the combinations of inequality constraints and attach to the equality constraints.
4. Iterate through each set of new constraints and use the Lagrange multiplier method on the new constraints to find a new $\mathbf{x}_0$.
5. Check if $\mathbf{x}_0$ satisfies the inequality constraints. If it does, add it to a list of potential $\mathbf{x}_{\min}$'s, and continue iteration. If it does not satisfy the inequality constraints, ignore it and continue iteration.
6. After the iteration is completed, find the $\mathbf{x}_{\min}$ in the list of potential $\mathbf{x}_{\min}$'s that gives the minimum $f(\mathbf{x})$.

3.5 Gradient Projection Method for Equality Constraints

The gradient projection method was originally proposed for optimization problems with linear equality constraints:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{Cx} - \mathbf{d} = 0\,. \end{array}$$

The basic idea is that we know some vector $\mathbf{x}'$ that satisfies the equality constraints. We find some direction **s**, which is the projection of the direction of steepest descent onto the constraint, so that it leads to another $\mathbf{x}'$ that also satisfies the equality constraints and is closest to the minimum.

From the derivations in [Snyman 2005] pg 81-84, which set up the problem as a Lagrangian of **s** with multipliers λ, the projected direction of steepest descent is chosen to be

$$\mathbf{s} = \frac{\nabla f(\mathbf{x}) + \mathbf{C}^T\lambda}{\left\|\nabla f(\mathbf{x}) + \mathbf{C}^T\lambda\right\|}$$

with

$$\lambda = -\left(\mathbf{CC}^T\right)^{-1}\mathbf{C}\nabla f(\mathbf{x})$$

and the gradient

$$\nabla f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\left(\mathbf{A}^T + \mathbf{A}\right) + \mathbf{b}^T\,.$$

The un-normalized gradient projection vector that is more often used in updating $\mathbf{x}'$ is simply:

$$\mathbf{u} = \nabla f(\mathbf{x}) + \mathbf{C}^T \lambda ,$$

so that

$$\mathbf{x}_{\min} = \mathbf{x}_0 + k\mathbf{u} ,$$

where k is the value that minimizes

$$F(k) = f(\mathbf{x}_0 + k\mathbf{u}).$$

We can confirm that $\mathbf{x}_{\min}$ is the minimum by checking that the gradient projection vector $\mathbf{u}(\mathbf{x}_{\min}) = 0$.

3.6 Gradient Projection Method for Inequality Constraints

The gradient projection method can also be extended to solve optimization problems with linear inequality constraints of the form:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{Dx} - \mathbf{e} \le 0 . \end{array}$$

The algorithm that I implemented for applying gradient projection to problems with inequality constraints is as follows:

1. Find the global minimum $\mathbf{x}_0$.
2. Check if $\mathbf{x}_0$ satisfies the inequality constraints. If it does, we're done, i.e. $\mathbf{x}_{\min} = \mathbf{x}_0$. If not, continue.
3. Find any vector $\mathbf{x}'$ that is on the active set of constraints, i.e. some boundary point that satisfies all constraints.
4. Given that the vector $\mathbf{x}'$ is on the active set, check if this vector is the minimum by testing if $\mathbf{u}(\mathbf{x}') \approx 0$. If it is, then we're done, i.e. $\mathbf{x}_{\min} = \mathbf{x}'$. If not, use gradient projection to update $\mathbf{x}'$.
5. If at some point the vector $\mathbf{x}'$ goes outside the constraints, i.e. the gradient projection reached beyond another constraint, then find the point of intersection of the new constraint and the old. Set this point as the new $\mathbf{x}'$, and add the new constraint to the active set. Make a new projection from this point and update $\mathbf{x}'$ to the projected vector.
6. If no such point of intersecting constraints was found, it indicates that $\mathbf{x}'$ was already on an optimal corner, so the old $\mathbf{x}'$ is optimal.
7. Return to Step 4.

4 Performance of Different Methods on Toy Problem

For the purpose of demonstration, I chose a toy problem of the form:

$$\text{minimize} \qquad f(\mathbf{x}) = 2x_1^2 + x_1 x_2 + 2x_2^2 - 6x_1 - 6x_2 + 15$$

subject to

$$\begin{aligned} x_1 + 2x_2 &\le 5 \\ 4x_1 &\le 7 \\ x_2 &\le 2 \\ -2x_1 + 2x_2 &= -1 \end{aligned}$$

which corresponds to the following matrix form:

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} -6 \\ -6 \end{bmatrix}, c = 15$$
$$\mathbf{C} = \begin{bmatrix} -2 & 2 \end{bmatrix}, \mathbf{d} = \begin{bmatrix} 5 \end{bmatrix}$$
$$\mathbf{D} = \begin{bmatrix} 1 & 2 \\ 4 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{e} = \begin{bmatrix} 5 \\ 7 \\ 2 \end{bmatrix}$$

Some methods can only handle equality constraints, some can only handle inequality constraints, and a couple can handle both equality and inequality constraints together. When testing the toy problem,

$$\mathbf{C} = \begin{bmatrix} 0 & 0 \end{bmatrix}, \mathbf{d} = \begin{bmatrix} 0 \end{bmatrix}$$

if the algorithm can only take inequality constraints, and

$$\mathbf{D} = \begin{bmatrix} 0 & 0 \end{bmatrix}, \mathbf{e} = \begin{bmatrix} 0 \end{bmatrix}$$

if the algorithm can only take equality ocnstraints.

4.1 Using Only Equality Constraints

There are four algorithms that I implemented which can handle optimization problems with equality constraints, including the penalty function method, Lagrange multiplier method, QP method, and gradient projection method. Figure 1 shows the results of each method on the toy problem excluding the inequality constraints. The small blue circles in the figure represent the iterations of $\mathbf{x}_k$ that the algorithm goes through.

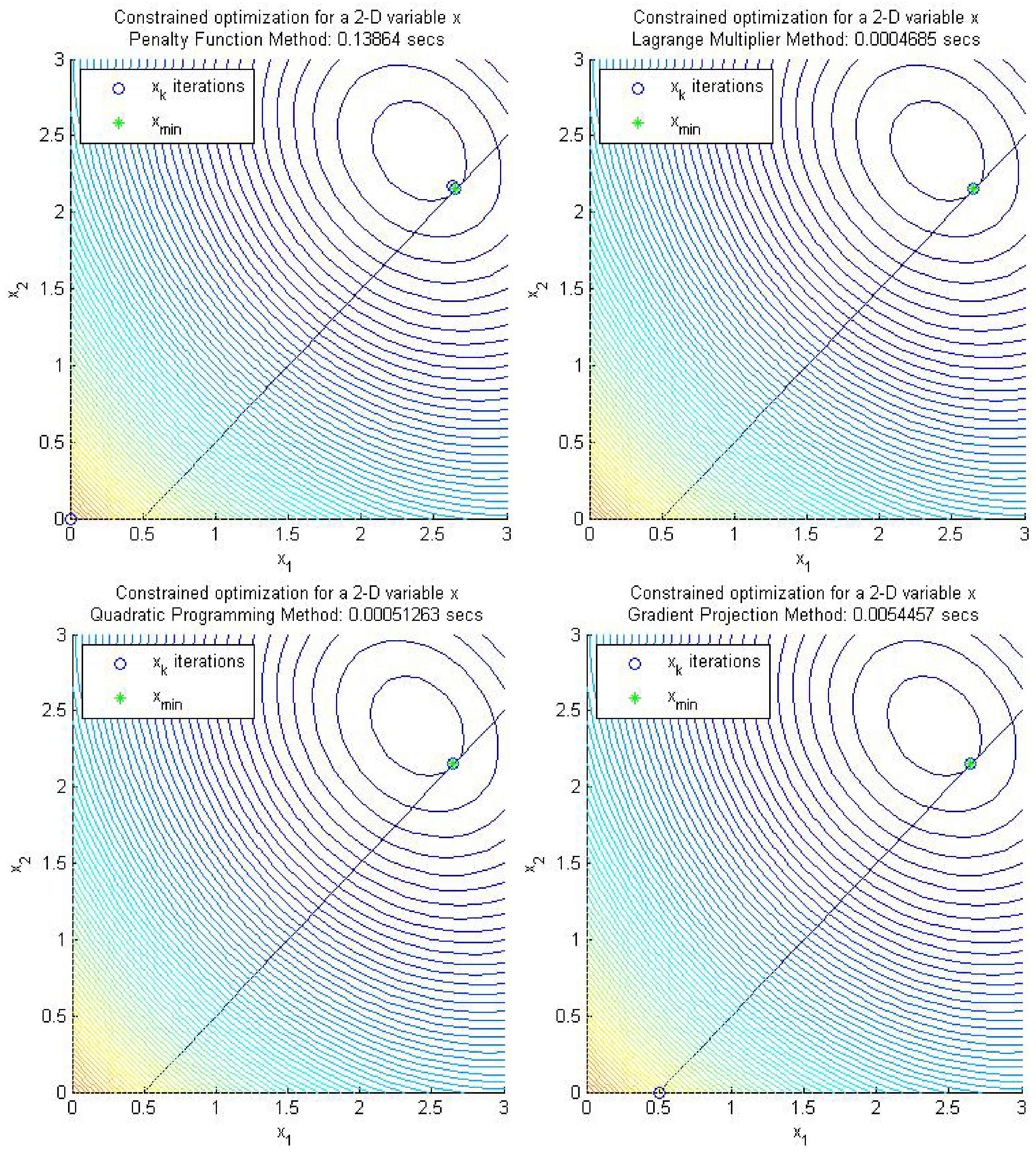


Figure 1. Results of different methods on toy optimization problem with only equality constraints

From Figure 1, we can see that the penalty function method goes through three iterations (starting from $\mathbf{x}_0 = [0,0]$) to settle on the solution, whereas the Lagrange multiplier and

QP methods only take one step to reach the solution. The gradient projection method starts from $\mathbf{x}_0 = [0.5, 0]$ on the constraint and makes a projection to the solution.

Figure 2 shows that the Lagrange multiplier and QP methods derive the solution slightly faster than the gradient projection method and much faster than the penalty function method. The Lagrange multiplier and QP methods are so fast because in my case of quadratic optimizer function and linear constraints, the partial derivatives are trivial to find. More complicated optimizer and constraint functions may cause these two methods to take more time in determining the partial derivatives. The scaling on Figure 2 may seem large, but that is because it is set for easy comparison with the time results for problems with inequality constraints in sections 4.2 and 4.3.

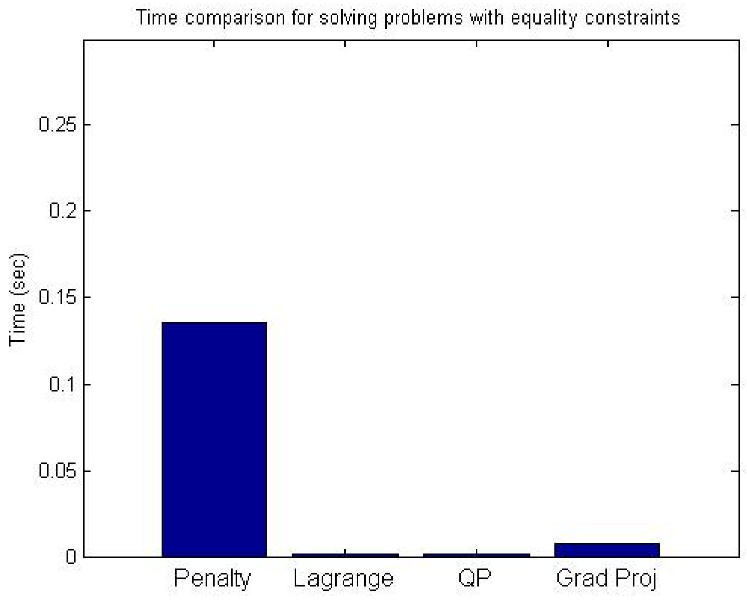


Figure 2. Comparison of run-time for methods handling equality constraints

4.2 Using Only Inequality Constraints

The algorithms that I implemented for optimization with inequality constraints include the penalty function method, augmented Lagrange method, QP method, and gradient projection method. Figure 3 shows the results of these methods on the example problem without equality constraints.

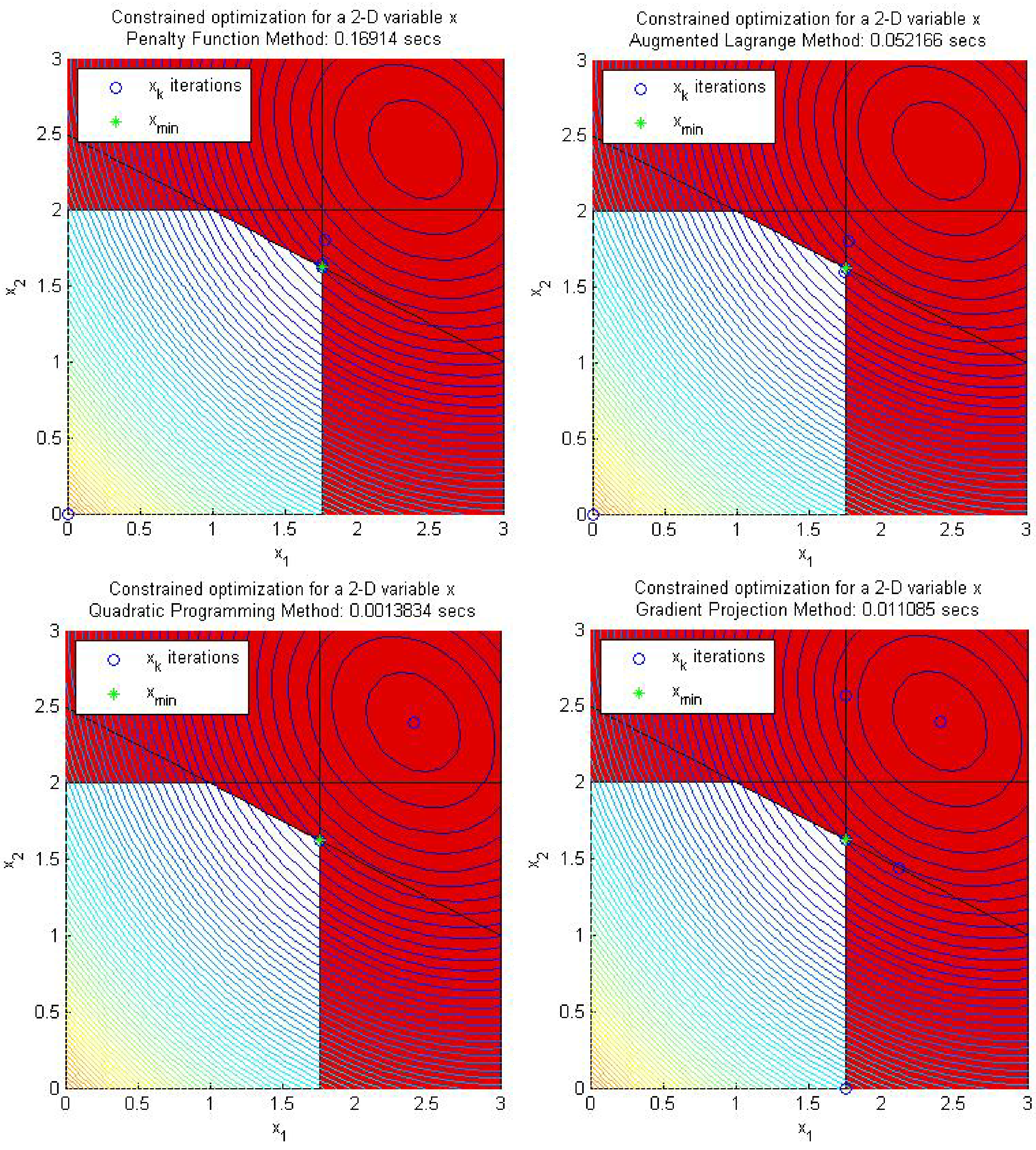


Figure 3. Results of different methods on toy optimization problem with only inequality constraints

From Figure 3, we see that the penalty function method performs in the same fashion as for equality constraints in section 4.1. The augmented Lagrange method performs more like the penalty function method than the Lagrange multiplier method for equality constraints because of the added penalty parameter. The iterative procedure in augmented

Lagrange also increased the run-time, as shown in Figure 4. Quadratic programming still performs very fast because it simply treats the active set of inequality constraints as equality constraints, and then runs a classical Lagrange multiplier procedure, which for quadratic optimizer and linear constraints, is quite a simple and fast task. The gradient projection method first checked if the global minimum satisfied the constraints, and since it did not, the algorithm chose some point on the active constraint set—in this case $\mathbf{x}' = [1.75, 0]$—and made a projection of the steepest path, which overshot another constraint. It found the point at which the two constraints intersected and made another projection from there. After detecting that the projection went in an infeasible direction, the algorithm concluded that the corner point must be the solution.

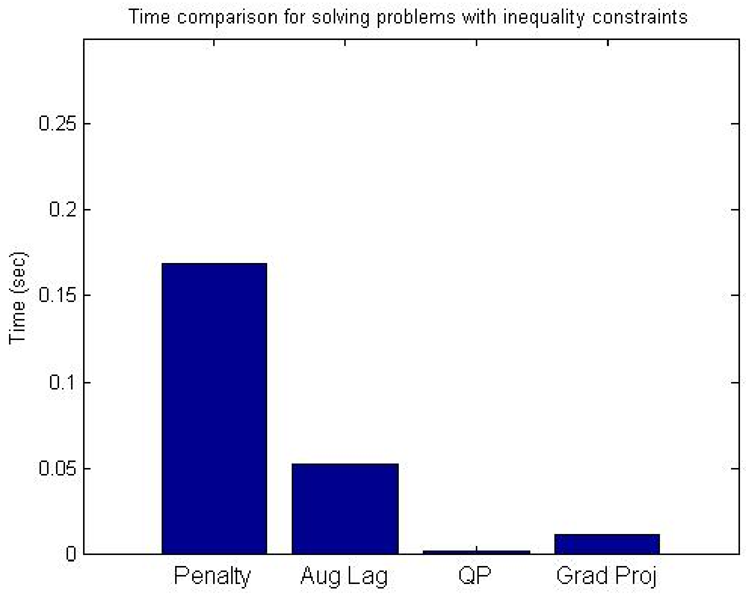


Figure 4. Comparison of run-time for methods handling inequality constraints

4.3 Using Both Equality and Inequality Constraints

I implemented two methods that can handle both equality and inequality constraints simultaneously, which are the penalty function method and the QP method. Figure 5 shows the results of these two methods on the full example problem.

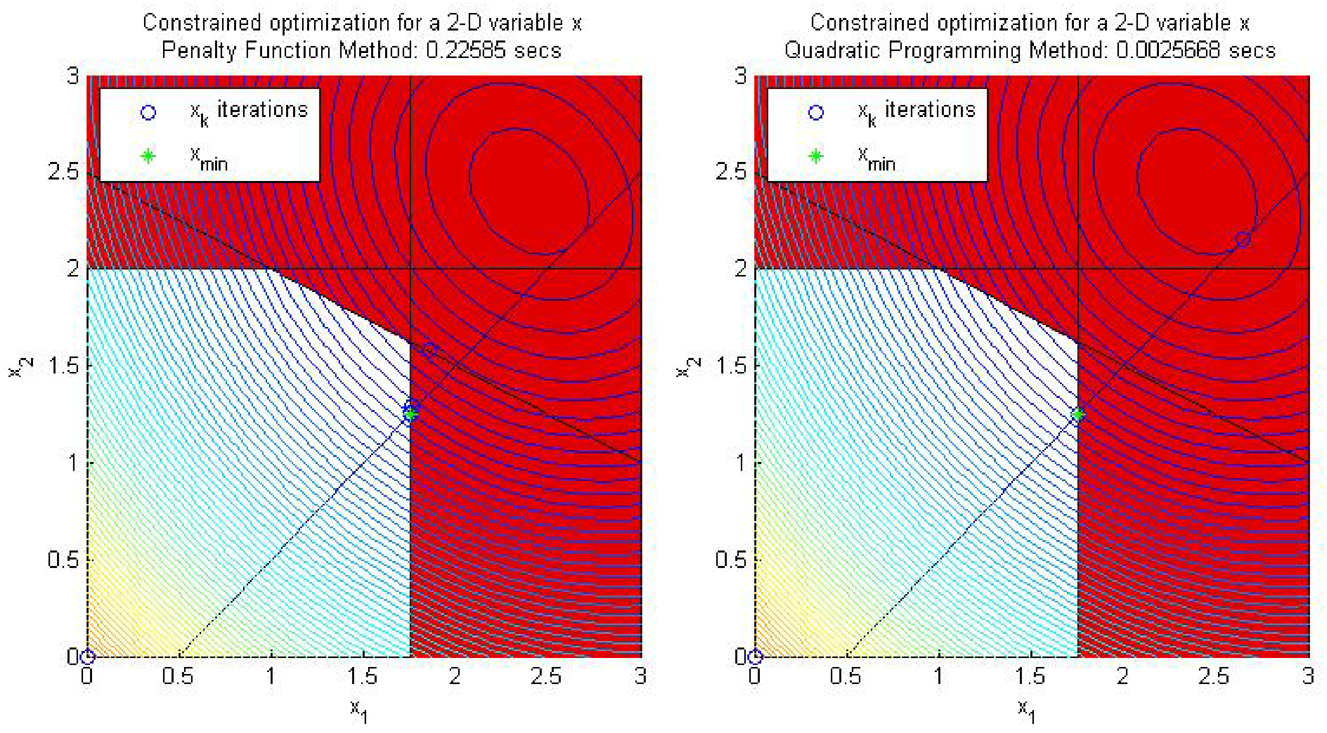


Figure 5. Results of penalty function and QP methods on toy optimization problem with equality and inequality constraints

The penalty function method took 6 iterations to solve this problem, whereas quadratic programming only looked at 3 possible values for **x**. The quadratic programming algorithm first finds the equality constrained optimum, checked that it does not satisfy the inequality constraints, then found all the combinations of inequality constraints that intersected with the equality constraint, and generated other possible optimum values. In this example, some combination of multiple inequality constraints with the equality constraint did not intersect, and thus generated the point [0,0]. The algorithm looks at all the possible optimal points and chooses the one that produces the smallest value for the optimizer function.

Figure 6 shows that the quadratic programming method performs much faster than the penalty function method on this problem. Aggregating Figures 2, 4, and 6, we find that quadratic programming generally outperforms all other available methods for problems with quadratic optimizer and linear constraints, because it uses this information to determine the form for partial derivatives a priori. If the problem were not limited to quadratic optimizer and linear constraints, some of the other methods might be more favorable.

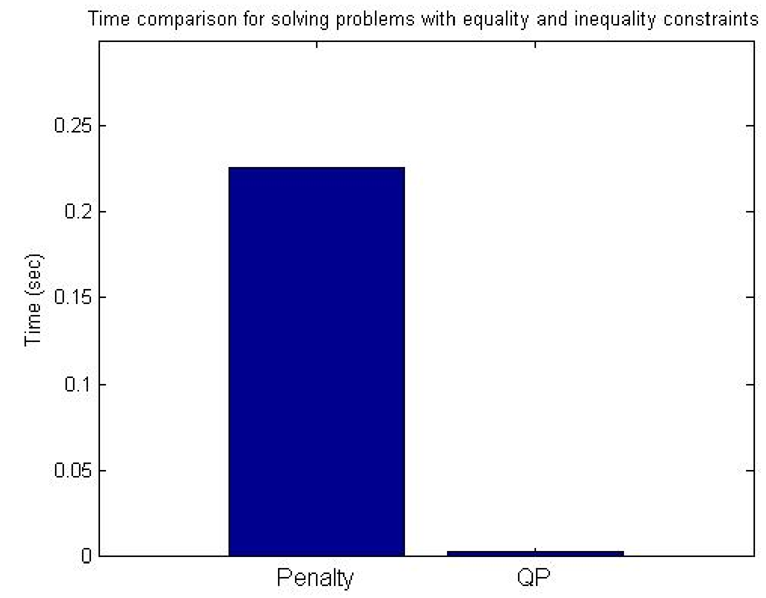


Figure 6. Comparison of run-time for methods handling equality and inequality constraints

5 Application of Constrained Optimization in Support Vector Machines for Data Classification

In the field of artificial intelligence and machine learning, classification is a useful tool for an intelligent agent to understand human intentions. One specific application is a smart pad that can recognize handwritten alphanumeric characters after learning from a training set. In the simplest case, there are only two different characters to distinguish. The machine learning algorithm that I use is called the support vector machine approach, first introduced by Vladimir Vapnik, and referenced in [Boser 1992].

5.1 SVM Problem Formulation

Assume the user draws a numeral on the smart pad to generate the training set. The smart pad, or the intelligent agent, selects a set of sample points from the user's input and stores them in a vector $\mathbf{x} = [x_1, y_1, x_2, y_2, \ldots]$. For training the algorithm, the user tells the agent the class that the vector belongs in, eg. the user's input corresponds to the numeral "0." We can consider a two class problem with class A and class B, eg. "0" vs. "1."

The training data consists of a set of *n*-dimensional vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_k, \ldots \mathbf{x}_p$. We label each $\mathbf{x}_k$ either +1 or –1 to associate it with a class: e.g. +1 for class A and –1 for class B. A training set containing *p* training points of vectors $\mathbf{x}_k$ and labels l_k would look like:

$$(\mathbf{x}_1, l_1), (\mathbf{x}_2, l_2), \ldots, (\mathbf{x}_k, l_k), \ldots, (\mathbf{x}_p, l_p), \quad \text{where } \begin{cases} l_k = 1 & \text{if } \mathbf{x}_k \in \text{class A} \\ l_k = -1 & \text{if } \mathbf{x}_k \in \text{class B} \end{cases}$$

5.1.1 Decision Function

Given the training data, the algorithm must derive a decision function (i.e. classifier) $D(\mathbf{x})$ to divide the two classes. The decision function may or may not be linear in **x** depending on the properties of the corpus and what kind of kernel is chosen. For our problem formulation, a decision value of $D(\mathbf{x})=0$ would be on the boundary between the classes. Any new, unknown vector **x** can be classified by determining its decision value: a positive decision value places **x** in class A, and a negative value places **x** in class B.

The decision function is a multidimensional surface that *maximizes the minimum margin* of the data in the two classes to the decision function. In our simple pedagogical example, *D* is some linear function of **x** = (*x, y*) that divides the two classes by maximizing the distance between itself and the nearest points in the corpus, as shown in Figure 7.

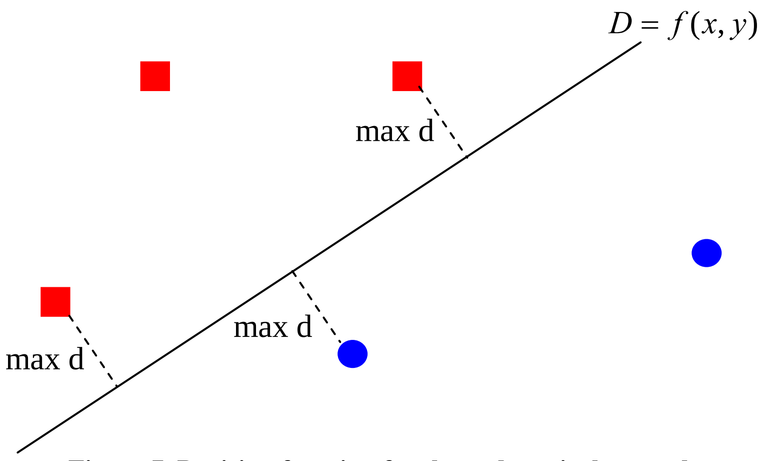


Figure 7. Decision function for the pedagogical example

The decision function of an unknown input vector **x** can be described in either primal space or dual space. In primal space, the representation of the decision function is:

$$D(\mathbf{x}) = \sum_{k=1}^{p} w_k \varphi_k(\mathbf{x}) + b\,,$$

where the φ_i are previously defined functions of **x,** and w_i and *b* are adjustable weight parameters of *D*. Here, *b* is the bias, i.e. offset of *D*. In dual space, the decision function is represented by:

$$D(\mathbf{x}) = \sum_{k=1}^{p} \alpha_k K(\mathbf{x}_k, \mathbf{x}) + b\,,$$

where the a_k and *b* are adjustable parameters, and $\mathbf{x}_k$ are the training patterns. The primal and dual space representations of *D*, i.e. Equations 2 and 3, are related to each other via the following relation:

$$w_i = \sum_{k=1}^{p} \alpha_k \varphi_i(\mathbf{x}_k).$$

K is a predefined kernel function of the following form for polynomial decision functions:

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^{\mathrm{T}} \cdot \mathbf{x}' + 1)^d .$$

5.1.2 1.1.3 Maximizing the Margin

Assume the training data is separable. Given n-dimensional vector $\mathbf{w}$ and bias b, $D(\mathbf{x}) / \|\mathbf{w}\|$ is the distance between the hypersurface D and pattern vector $\mathbf{x}$. Define M as the margin between data patterns and the decision boundary:

$$\frac{l_k D(\mathbf{x}_k)}{\|\mathbf{w}\|} \geq M$$

Maximizing the margin M produces the following minimax problem to derive the most optimal decision function:

$$\max_{\mathbf{w}, \|w\|=1} \min_k l_k D(\mathbf{x}_k) .$$

Figure 8 shows a visual representation of obtaining the optimal decision function.

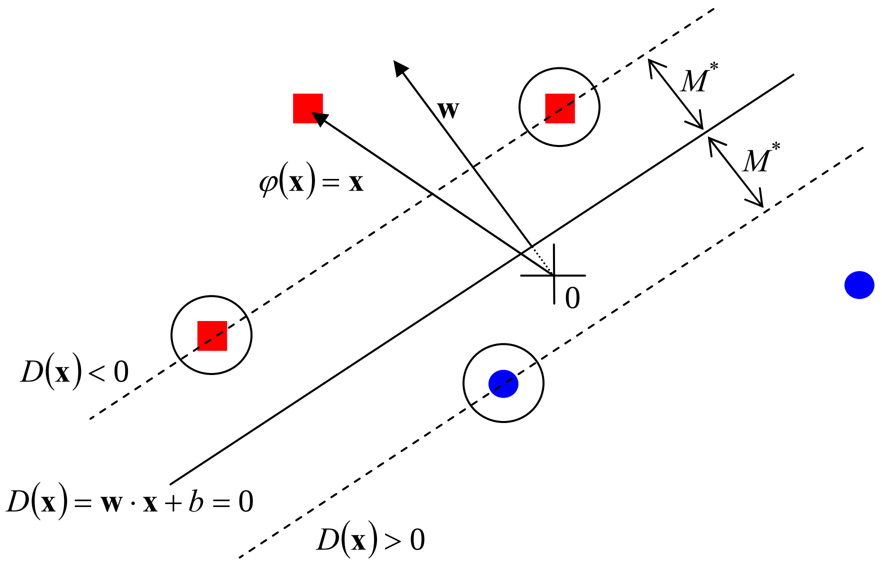

$\mathbf{w}$
M^*
M^*
$\varphi(\mathbf{x}) = \mathbf{x}$
0
$D(\mathbf{x}) < 0$
$D(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = 0$
$D(\mathbf{x}) > 0$

Figure 8. Finding the decision function. The decision function is obtained by determining the maximum margin *M. Encircled are the three support vectors.**

The following Lagrangian in the primal space can be used to solve the optimization:

$$\begin{aligned} &\text{minimize} && L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{k=1}^{p} \alpha_k \left[l_k D(\mathbf{x}_k) - 1 \right] \\ &\text{subject to} && \alpha_k \geq 0, \qquad k = 1, 2, \ldots, p \\ & && \alpha_k \left[l_k D(\mathbf{x}_k) - 1 \right] = 0, \qquad k = 1, 2, \ldots, p \end{aligned}$$

where the α_k are the Lagrange multipliers. Taking the partial derivatives of the Lagrangian with respect to **w** and *b* produces two equations:

$$\mathbf{w} = \sum_{k=1}^{p} \alpha_k l_k \varphi_k$$

and

$$\sum_{k=1}^{p} \alpha_k l_k = 0$$

Substituting these two equations back into the Lagrangian function gives the problem formulation in the dual space:

$$\begin{aligned} &\text{minimize} && J(\boldsymbol{\alpha}) = \sum_{k=1}^{p} \alpha_k - \frac{1}{2} \boldsymbol{\alpha} \cdot \mathbf{H} \cdot \boldsymbol{\alpha} \\ &\text{subject to} && \alpha_k \geq 0, \qquad k = 1, 2, \ldots, p \\ & && \sum_{k=1}^{p} \alpha_k l_k = 0 \end{aligned}$$

The kernel function *K* is contained in the $p \times p$ matrix **H**, which is defined by

$$H_{km} = l_k l_m K(\mathbf{x}_k, \mathbf{x}_m)$$

The optimal parameters **α*** can be found by maximizing the Lagrangian in the dual space using either the penalty function method or quadratic programming. The training data that corresponds to non-zero values in **α*** are the support vectors. There are usually fewer support vectors than the total number of data points. The equality constraint in the dual problem requires at least one support vector in each class. Assuming two arbitrary support vectors $\mathbf{x}_A \in \text{class A}$ and $\mathbf{x}_B \in \text{class B}$, the bias *b* can be found:

$$b^* = -\frac{1}{2} \sum_{k=1}^{p} l_k \alpha_k^* \left[K(\mathbf{x}_A, \mathbf{x}_k) + K(\mathbf{x}_B, \mathbf{x}_k) \right].$$

The decision function is therefore:

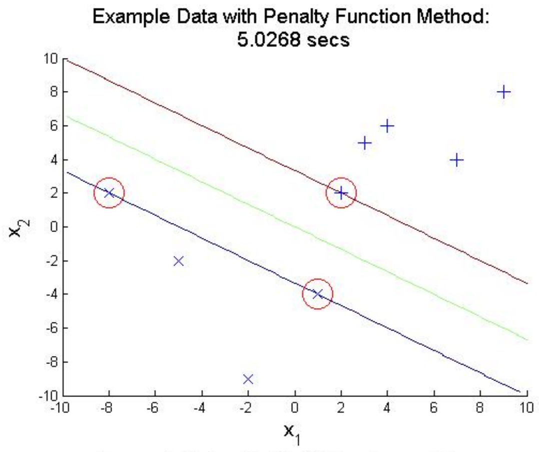
$$D(\mathbf{x}) = \sum_{k=1}^{p} l_k \alpha_k^* K(\mathbf{x}_k, \mathbf{x}) + b^* .$$

Each datum $\mathbf{x}$ is classified according to

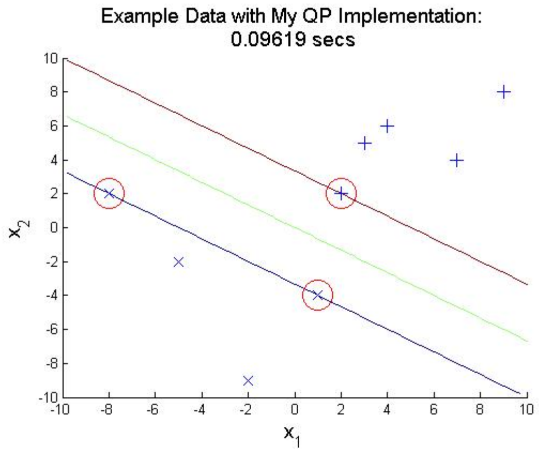
$$\operatorname{sgn}[D(\mathbf{x})].$$

6 Performance of SVM for Sample 2-D Data Classification Example

Since the optimization problem that we are solving has both equality and inequality constraints, we can use the methods discussed in section 4.3, i.e. the penalty function and quadratic programming methods. I first demonstrate that these methods work on a simple set of fabricated 2-D data in Figure 9. I also include the results from MATLAB's built-in quadratic programming function for comparison in Figure 10.

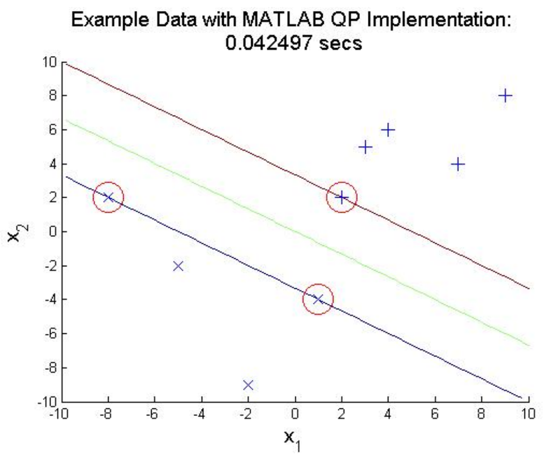


x_1	x_2	l	α
3	5	+1	0
4	6	+1	0
2	2	+1	0.065
7	4	+1	0
9	8	+1	0
-5	-2	–1	0
-8	2	–1	0.015
-2	-9	–1	0
1	-4	–1	0.05



x_1	x_2	l	α
3	5	+1	0
4	6	+1	0
2	2	+1	0.065
7	4	+1	0
9	8	+1	0
-5	-2	–1	0
-8	2	–1	0.015
-2	-9	–1	0
1	-4	–1	0.05

Figure 9. The linear decision function for example data is shown in green. Encircled are the support vectors. The data point values, labels, and optimal parameters *α* are shown on the right.



x_1	x_2	l	α
3	5	+1	0
4	6	+1	0
2	2	+1	0.065
7	4	+1	0
9	8	+1	0
-5	-2	–1	0
-8	2	–1	0.015
-2	-9	–1	0
1	-4	–1	0.05

Figure 10. MATLAB's results of the linear decision function.

In Figures 9 and 10, there are nine data points. The four labeled × are in one class, and the five labeled + are in another. The support vector machine algorithm says to find the decision function which minimizes the maximum margin between the two classes. The support vectors that are found correspond to values for which $\alpha \geq 0$, and are circled in the plots. In this example, there are three support vectors although in general there could be fewer or more support vectors depending on the data set. Figures 9 and 10 show that the penalty function and quadratic programming methods both produce identical results for this simple example. Figure 11 shows the run-time of each method. Again we see that the penalty method takes much longer for these small dimension problems. My implementation of quadratic programming performs slightly slower than MATLAB's implementation, probably due to some minor inefficiencies.

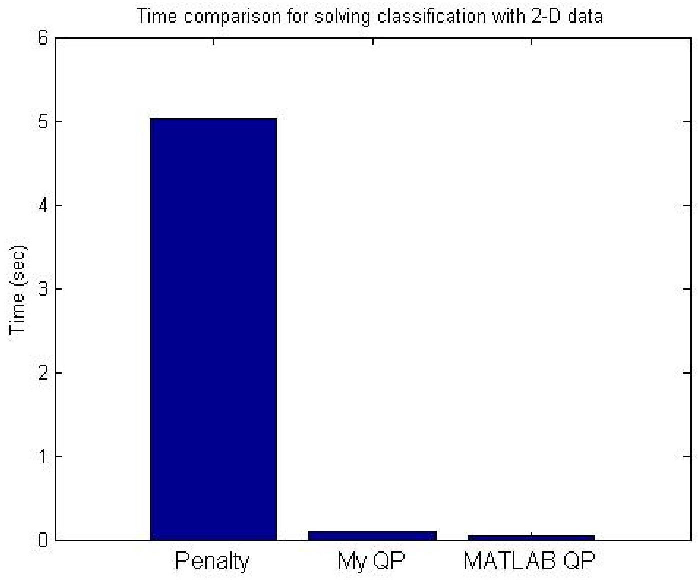
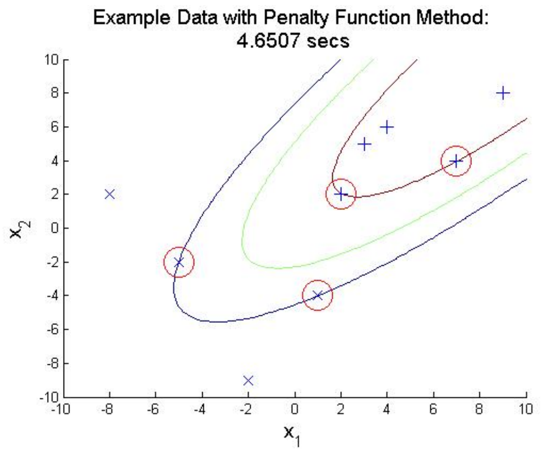
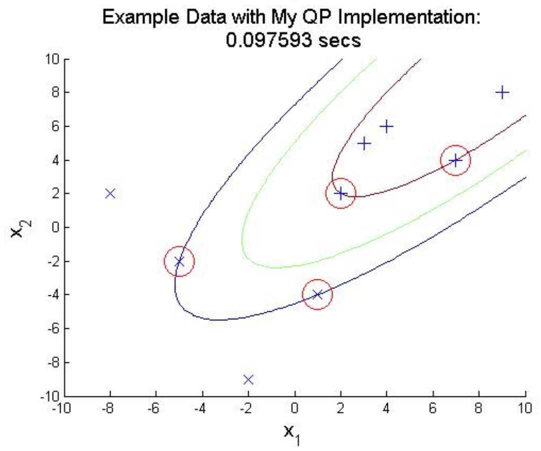


Figure 11. Comparison of run-time for methods solving the example 2-D data classification using first order kernel functions

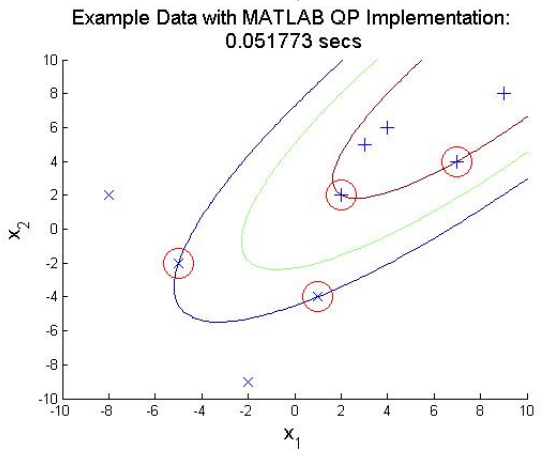
I also ran the support vector machine method with a second order kernel function, which produces a quadratic decision function. The results are shown in Figure 12. We can see that the penalty function method produces slightly different α values from quadratic programming due to the iteration cutoff error. The convergence criteria for the penalty function method was set to 10^{-8}. Making the convergence criteria tighter would help the penalty function method to produce more accurate results, but would increase run-time.



x_1	x_2	l	α
3	5	+1	0
4	6	+1	0
2	2	+1	0.0108
7	4	+1	0.0017
9	8	+1	0
-5	-2	–1	0.0070
-8	2	–1	0
-2	-9	–1	0
1	-4	–1	0.0055



x_1	x_2	l	α
3	5	+1	0
4	6	+1	0
2	2	+1	0.0109
7	4	+1	0.0017
9	8	+1	0
-5	-2	–1	0.0070
-8	2	–1	0
-2	-9	–1	0
1	-4	–1	0.0056



x_1	x_2	l	α
3	5	+1	0
4	6	+1	0
2	2	+1	0.0109
7	4	+1	0.0017
9	8	+1	0
-5	-2	–1	0.0070
-8	2	–1	0
-2	-9	–1	0
1	-4	–1	0.0056

Figure 12. The second order decision function for example data is shown in green. Encircled are the support vectors. The data point values, labels, and optimal parameters α are shown on the right.

Figure 13 compares the run-times of the different methods on this problem. The trend is similar to what we've seen in the first order kernel case: the penalty function method performs much slower than QP. It is interesting to note that the run-times for the second order kernel case are actually slightly faster than in the first order kernel case. This happened consistently when I ran both cases multiple times, and is probably caused by the positions of the data points.

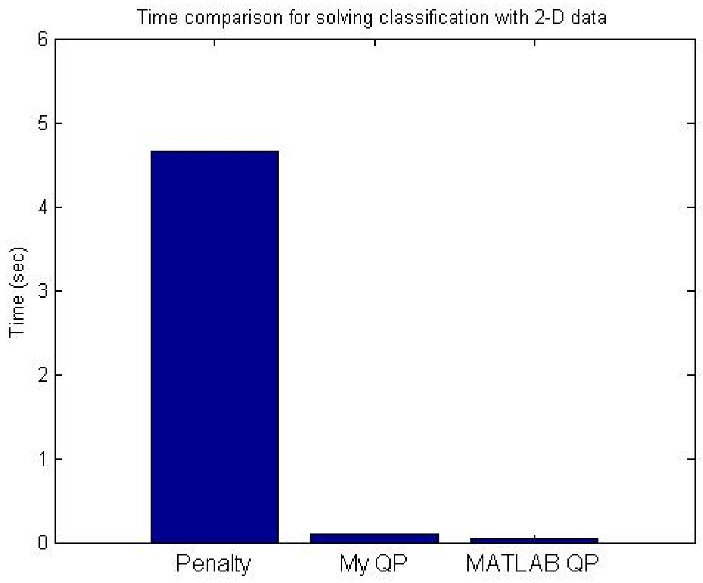


Figure 13. Comparison of run-time for methods solving the example 2-D data classification using second order kernel functions

7 Performance of SVM for Handwritten Numeral Recognition

Now the goal is to apply the constrained optimization methods to a real world application. This paper introduces the first steps of handwritten text recognition by trying to distinguish between just two handwritten numerals, in this case a "0" and a "1." The handwritten numerals data can be obtained from the UCI Repository of machine learning databases [Newman 1998], under the "pendigits" directory. E.Alpaydin and F. Alimoglu of Bogazici University in Istanbul, Turkey provided the database, which contains a total of over 10000 data points. Such a large collection of data, while theoretically able to reach very accurate results, is too much for most personal computers to handle—either it takes an unreasonable amount of time to solve the optimization or the machine runs out of memory. Moreover, very reasonable classifiers can be achieved with far fewer data. Therefore for demonstration purposes, I only use 20 data points for training, and about 700 for testing.

Each data holds the (x, y) position information of 8 sample points within a handwritten numeral, so the dimension of the data is 16. All of the data are normalized to within (0, 0) and (100, 100). Figure 14 shows some typical user input data.

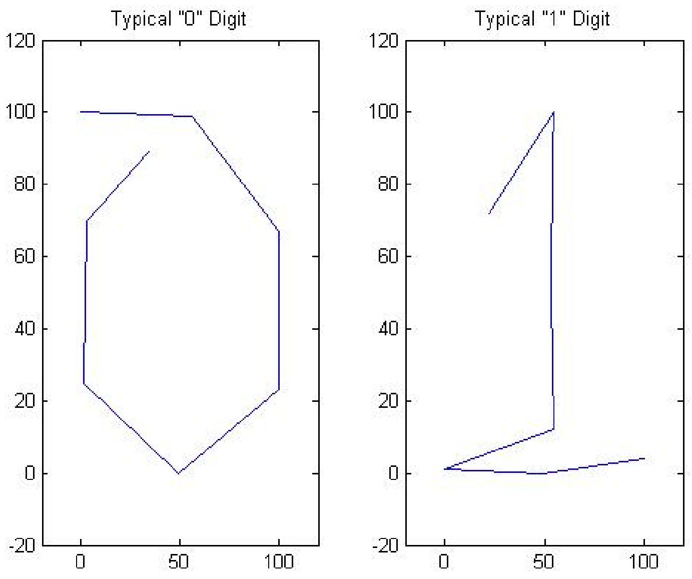


Figure 14. Typical user input vectors

Because each $\mathbf{x}_k$ vector is of dimension 16, the problem becomes very complicated. My implementation of the quadratic programming method uses the method of Theil and Van de Panne, which iterates through all the different combinations of inequality constraints to find the active set. This method grows factorially with the number of inequality constraints. In this problem, the inequality constraints are $0 \le \alpha_k \le Const,\ k = 1,2,\ldots,p$ to allow for misclassifications (see [Cortes 1993] for more detail on soft margin classifiers), and thus there are $p \times 2 = 32$ inequality constraints. The 32 inequality constraints can form over a billion different combinations to form the active set of constraints. Thus it is infeasible for my implementation of the QP method to perform optimization on this problem.

Since the penalty function method is iterative, it does not scale with the number of constraints, and thus still performs the optimization in a reasonably short amount of time.

The MATLAB implementation of quadratic programming with inequality constraints actually applies an active set method that uses projections, which is a much leaner way of finding the active constraints. Implementing this more advanced version of the quadratic programming method is beyond the scope of this paper, but for future reference, this method should be considered for further investigation.

At any rate, I performed classification on the 20 handwritten numerals training data using both the penalty function optimizer and MATLAB's QP, and tested the classifier on over 700 test data. The performance results are shown in Figure 15. The penalty function method misclassified about 7% of the test data, and MATLAB's QP misclassified about 4.5% of the test data. Again, reducing the penalty function's convergence criteria may help reduce the misclassification error.

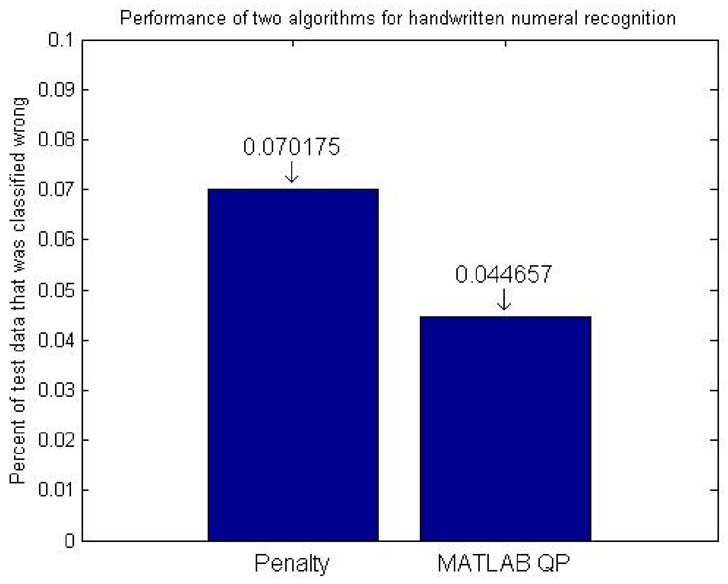


Figure 15. Percent of misclassified test data for handwritten numeral recognition

Figure 16 shows the run-times for the two methods. The penalty function method takes over 10 seconds whereas MATLAB's quadratic programming method takes less than a tenth of a second.

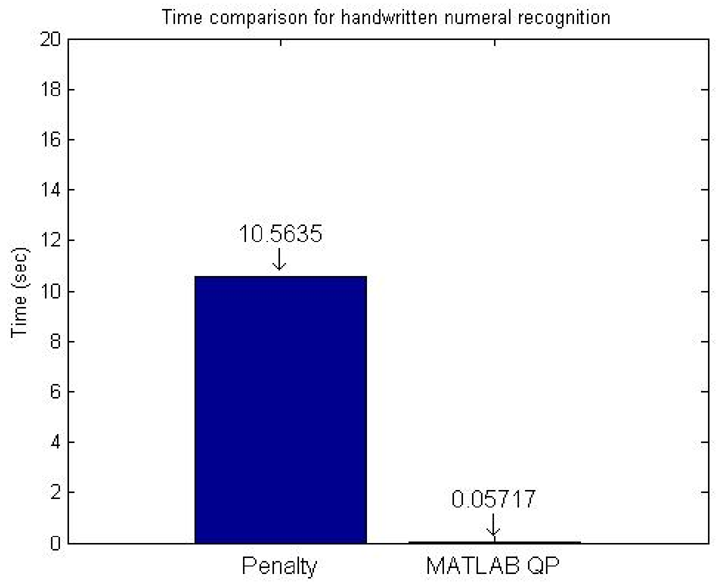


Figure 16. Comparison of run-times for using the penalty function method and MATLAB's QP method on handwritten numeral recognition

8 Conclusion and Future Work

In this paper, I have introduced the constrained optimization problem, and implemented several optimization methods, including the penalty function method, Lagrange multiplier method for equality constraints, augmented Lagrange multiplier for inequality constraints, quadratic programming, gradient projection method for equality constraints, and gradient projection for inequality constraints. I performed the optimization techniques on a simple toy problem and demonstrated in general how each method works. I used the applicable methods on the real-world application of data classification, specifically in handwritten numerals recognition.

One major extension to what I have done here is to expand the constraint optimization problem to more than a quadratic optimizer with linear constraints. Also, the optimization methods that I implemented are the more common techniques. Future work can be done to investigate some more sophisticated methods such as the new gradient based methods mentioned in [Snyman 2005] or the QP implementation that MATLAB uses. Finally, the SVM method can be extended to perform classification with multiple classes.

9 References

B. E. Boser, I. Guyon, and V. N. Vapnik, “A Training Algorithm for Optimal Margin Classifiers”, *In the Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pps 144-152, Pittsburgh 1992.

E.K.P. Chong and S.H. Zak. *An Introduction to Optimization*. John Wiley & Sons, Inc. New York: 1996.

C. Cortes and V. Vapnik. 1993 . *The soft margin classifier*. Technical memorandum 11359-931209-18TM, AT&T Bell Labs, Holmdel, NJ.

L.R. Foulds. *Optimization Techniques*. Springer-Verlag New York Inc. New York: 1981.

D.M. Greig. *Optimisation*. Longman Group Limited. London: 1980.

J.C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions," *SIAM Journal of Optimization*, Vol. 9, Number 1, pp.112-147, 1998.

D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. *UCI Repository of machine learning databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

J.A. Snyman. *Practical Mathematical Optimization*. Springer Science+Business Media, Inc. New York: 2005.