Fminimax

Stewart Curry, Kristy Mitchell, Robert Smith, Jasmine West

April 9, 2013

1 Introduction

fminimax is an algorithm which solves constrained optimization problems of the form:

$$\begin{aligned} \min_x \max_{F_i}\{F_i(x)\} \text{ such that } c(x) &\leq 0 \\ ceq(x) &= 0 \\ Ax &\leq b \\ Aeq \cdot x &= beq \\ lb \leq x &\leq ub \end{aligned}$$

where $\{F_i(x)\}$ is a set of multivariable functions, starting at an initial estimate. b and beq are vectors, A and Aeq are matrices, and $c(x)$, $ceq(x)$ can be nonlinear functions. x, lb, and ub can be passed as vectors or matrices.

In other words, fminimax solves minimax problems, which seek to minimize the worst case value (or maximum loss) for a set of functions. For example, *fminimax* can be used to find the value of x that minimizes the maximum value of $f = [f_1(x), f_2(x), f_3(x), f_4(x), f_5(x)]$ where

$$\begin{aligned} f_1(x) &= 2x_1^2 + x_2^2 - 48x_1 - 40x_2 + 304 \\ f_2(x) &= -x_1^2 - 3x_2^2 \\ f_3(x) &= x_1 + 3x_2 - 18 \\ f_4(x) &= -x_1 - x_2 \\ f_5(x) &= x_1 + x_2 - 8 \end{aligned}$$

Using an initial estimate of $x0 = [0.1, 0.1]$, *fminimax* returns the solution $x = [4, 4]$. This means that $x = [4, 4]$ "minimizes the maximum value" of the vector f.

2 Visualization

A simple discrete example of a minimax problem can be seen in the following table, which sets up a game between Player A and Player B with the results for both players given in the form (Player A result, Player B result):

	Player B chooses B1	Player B chooses B2	Player B chooses B3
Player A chooses A1	(0,1)	(0,2)	(0,3)
Player A chooses A2	(1,1)	(2,2)	(3,2)
Player A chooses A3	(-10,1)	(20,2)	(30,1)

One strategy Player B might employ would be to minimize Player A's maximum gain. For Player B's choices of B1, B2, and B3, Player A's maximum outcomes are $+1$, $+20$, and $+30$, respectively. Thus, choice B1 would be a minimax choice for Player B because it minimizes Player A's maximum outcome. For a similar example, see the Kung-Fu Fighting handout.

In a very simple continuous example, consider the functions $f_1(x) = x$ and $f_2(x) = x^2$. The minimax solution can be thought of as the value for x that minimizes $g(x) = max[f_1(x), f_2(x)]$.

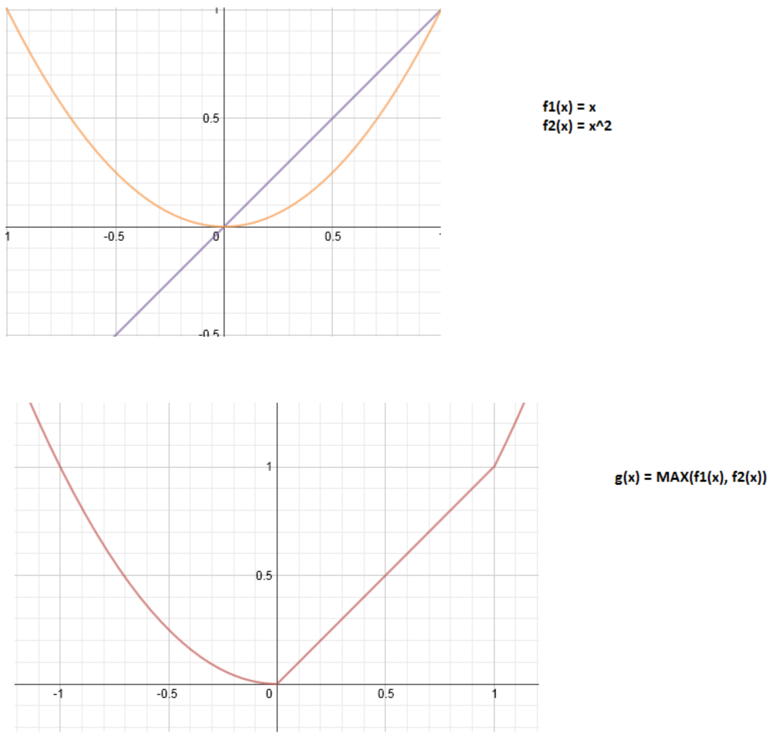

Figure 1 – *Continuous Example*

As you can see, the minimax solution in this example would be x = 0.

Similarly, consider the funcitons $f_1(x) = x + y$, $f_2(x) = x - y$, $f_3(x) = y - x$ and $f_4(x) = -(x + y)$. Again, the minimax solution can be thought of as the vector x that minimizes $g(x) = max[f_1(x), f_2(x), f_3(x), f_4(x)]$

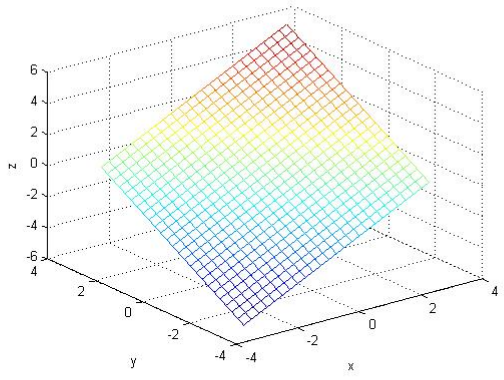

Figure 2 – $f_1 = x + y$

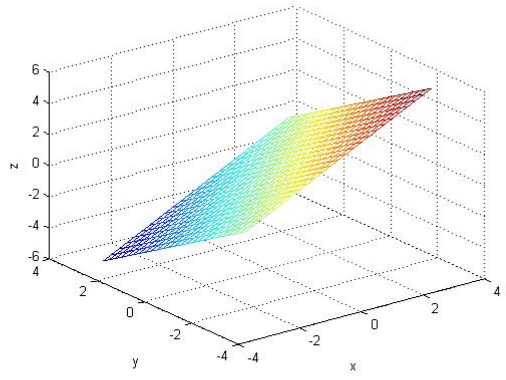

Figure 3 – $f_2 = x - y$

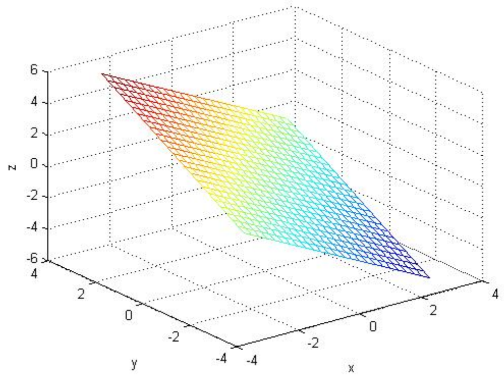

Figure 4 – $f_3 = y - x$

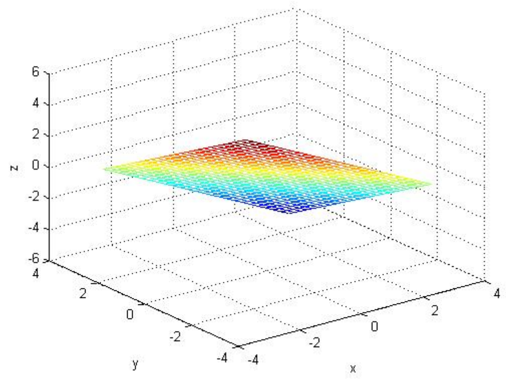

Figure 5 – $f_4 = -(x + y)$

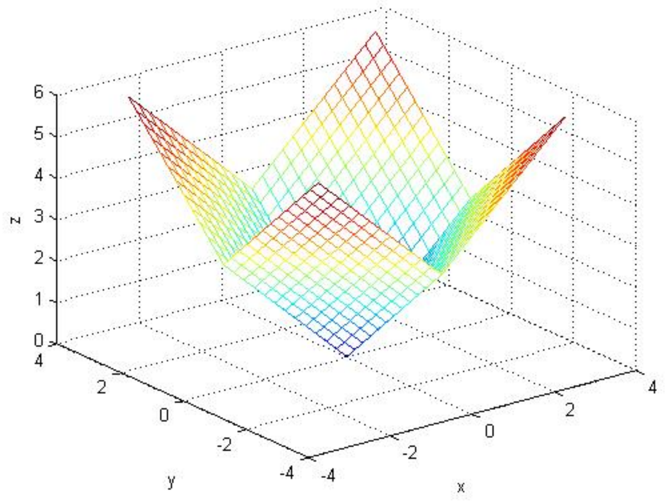

Figure 6 – $g(x) = \max\{f_1, f_2, f_3, f_4\}$

Here, the minimax solution would be the vector x = [0 0]

3 Sequential Quadratic Programming Method

The method that $fminimax$ uses to solve problems is known as the Sequential Quadratic Programming Method (SQP). SQP is an iterative method that is used in solving nonlinear optimization problems. SQP methods are only used on optimization problems for which the objective function along with the constraints are both twice continuously differentiable. SQP consists of several methods used to solve problems depending on what is given. For example, for unconstrained problems, the method reduces itself to Newton's method in order to find a point where the gradient of the objective vanishes. Now, if the problem only has equality constraints rather than inequality constraints then the SQP uses a method that is the same as applying Newton's method to the first order optimality conditions of the problem.

There are three main stages of this algorithm. First it updates the Hessian Matrix, then it finds a Quadratic Programming Solution and then uses the Line Search and Merit Function to find the iterates and ultimately solve the $fminimax$ problem.

3.1 Updating the Hessian Matrix

Let H, be a positive definite quasi-Newton approximation of the Hessian of the Lagrangian function. Also let $\lambda_i,\ i = 1, \ldots, m$ be the estimate of the Lagrange multipliers. In order to update the Hessian, SQP uses the following formula:

$$H_{k+1} = H_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{H_k^T s_k^T s_k H_k}{s_k^T H_k s_k}$$

where,

$$s_k = x_{k+1} - x_k \text{ and}$$
$$q_k = \left(\nabla f(x_{k+1}) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x_{k+1})\right) - \left(\nabla f(x_k) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x_k)\right)$$

The key to updating the Hessian matrix is to keep the Hessian positive definite as suggested in [1]. There are various ways of controlling $q_k^T s_k$ and keeping it positive in order to guarantee that the Hessian is positive definite. Sometimes, programs such as MATLAB will display a message that says "no update." This means that $q_k^T s_k$ is nearly zero and as a result one can conclude that the original problem was set up wrong or that the function being minimized is not continuous.

3.2 Quadratic Programming Solution

The Quadratic Programming solution(QPS) attempts to find a solution to the Quadratic programming subproblem which is explained below. For each major iteration of the SQP method, the quadratic programming solution looks as follows:

$$\begin{aligned} \min_{d \in \mathbb{R}^n} q(d) &= \frac{1}{2} d^T H d + C^T d \\ A_i d &= b_i,\ i = 1, \ldots, m_e \\ A_i d &\leq b_i,\ i = m_e + 1, \ldots, m \end{aligned}$$

where A_i means the ith row of matrix A, which is an $m \times n$ matrix. Now the QPS has two phases:

Phase 1 Calculation of a feasible point (if one exists)

Phase 2 Generation of iterative sequence of feasible points that converge to the solution.

For the sake of time, we will not go into details regarding these phases. However, they are used to complete the QPS.

3.3 Line Search and Merit Function

The solution to the Quadratic Programming subproblem talked about in the previous section, gives the vector d_k, which can now be used to form a new iterate, namely:

$$x_{k+1} = x_k + \alpha d_k$$

where α_k is the step length parameter. Note that α_k helps to create a sufficient decrease in a merit fuction. The merit function is as follows:

$$\Psi(x) = f(x) + \sum_{i=1}^{m_e} r_i g_i(x) + \sum_{i=m_e+1}^{m} r_i \cdot \max[0, g_i(x)]$$

where r_i is known as the penalty parameter and is defined as follows:

$$r_i = (r_{k+1})_i = \max_i \left\{ \lambda_i, \frac{(r_k)_i + \lambda_i}{2} \right\}, i = 1, \ldots, m$$

with the initial value of the penalty parameter defined as follows:

$$r_i = \frac{\|\nabla f(x)\|_2}{\|\nabla g_i(x)\|_2}$$

note that $\|\cdot\|_2$ is the $2-$norm.

4 Examples

4.1 Simple Example

The following is an example of a simple system of equations:

$$\begin{aligned} F(x,y) &= x+y \\ G(x,y) &= x-y \\ H(x,y) &= y-x \end{aligned}$$

In order to evaluate, we use the following code:

$[x, fx] = fminimax(@myfun1, x0)$

This code will evaluate $myfun1$, which contains this system, starting at the vector given by $x0$ to give the minimizer vector x which, when given as input to the system, results in the output vector fx.

For $x0 = [10, 1]$, our output looks as follows:

$$\begin{aligned} x &= [1.0e-06 * -0.1045, -0.0000] \\ fx &= [1.0e-06 * -0.1045, -0.1045, 0.1045] \end{aligned}$$

Note that, although we can see that the true minimizer is $[0, 0]$, the function meets default stopping criteria before actually reaching the true value.

4.2 Approximation

fminimax's most common use is in approximation. Suppose you have a system of simultaneous equations.

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= b_1 \\ f_2(x_1, x_2, \ldots, x_n) &= b_2 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= b_n \end{aligned}$$

In many applications, the system may lack a solution altogether, in which case the best that can be done is to find the input vector which comes closest to satisfying the equation. This is equivalent to minimizing the error, the absolute value of the difference between the given b_i and actual output of $f_i(x)$. This can be stated as the following *minimax* problem:

$$\begin{aligned} |f_1(x_1, x_2 \ldots x_n) &- b_1| \\ |f_2(x_1, x_2 \ldots x_n) &- b_2| \\ &\vdots \\ |f_n(x_1, x_2 \ldots x_n) &- b_n| \end{aligned}$$

We use the following code to get an approximation:

$[x, fx] = fminimax(@myfun2, x0, [\,], [\,], [\,], [\,], [\,], [\,], [\,], options)$
This gives the optimal approximation, here using an arbitrary, fixed system of equations. (Note that $myfun2$ does not include the absolute value. Instead, $options = optimset(\text{'}MinAbsMax', 20)$ sets $fminimax$ to minimize the absolute value of the first 20 equations, which is all of them in this example.)

4.3 Constrained Optimization Example

The above example is for unconstrained approximation. $fminimax$ is also capable of solving constrained optimization problems. For example,
$[x, fx] = fminimax(@myfun2, x0, ICE, ICN, [\,], [\,], [\,], [\,], [\,], options)$ uses the inequality coefficients vector ICE and inequality constants matrix ICN to define the inequality constraints listed below:

$$ICEi * xi <= ICNi$$

which, like all constraints in our example, will be chosen arbitrarily. Similarly,
$[x, fx] = fminimax(@myfun2, x0, [\,], [\,], ECE, ECN, [\,], [\,], [\,], options)$ defines equality constraints. This gives the solution:

$$ECEi * xi = ECNi$$

Note that MATLAB has a tolerance for constraint violation just as it does for other aspects of a problem, and will thus sometimes converge to solutions which do not exactly equal the provided value. This is why the solution is not necessarily $xi = ECEi \backslash ECNi$

$[x, fx] = fminimax(@myfun2, x0, [\,], [\,], [\,], [\,], LB, UB, [\,], options)$ defines boundary constraints. LB is a vector of lower bounds, while UB is a vector of upper bounds. The solution is then subject to:
$LBi <= xi <= UBi$

4.4 Nonlinear Constraints

The above have all consisted of linear constraints.
$[x, fx] = fminimax(@myfun2, x0, [\ ], [\ ], [\ ], [\ ], [\ ], [\ ], @myfun3, options)$ allows for nonlinear equality constraints, $ceq(x) = 0$, or inequality constraints,$c(x) <= 0$. In our example $(xi)^3+10 <= 0$ is a nonlinear constraint.

5 Conclusion

In conclusion, fminimax minimizes the maximum output of a set of multivariate functions, possibly subject to equality or inequality constraints. The algorithm accomplishes this task through a sequential quadratic programming method which solves the minimax program iteratively, starting at an initial estimate. Similar to minimax problems, which seek to minimize the maximum loss, maxmin problems seek to maximize the minimum gain. *fminimax* can be used to solve these problems as well, by using the identity:

$$\max_x \min_i F_i(x) = -\min_x \min_i(-F_i(x)).$$

Overall, fminimax, like other algorithms we have used this semester, is another tool built into MATLAB useful in solving optimization problems.

References

[1] Powell, M.J.D., "A Fast Algorithm for Nonlinearly Constrained Optimization Calculations," *Numerical Analysis*, G.A.Watson ed., Lecture Notes in Mathematics, Springer Verlag, Vol. 630, 1978.

[2] Mathworks. "Solve minimax constraint problem - MATLAB fminmax" Mathworks.com. http://www.mathworks.com/help/optim/ug/fminimax.html (accessed April 5, 2013).