Method of Moving Asymptotes

APPM 5630: Convex Optimization
28 April 2023

Grant Norman

1 Introduction

The Method of Moving Asymptotes (MMA) was originally introduced by Svanberg [1] as a Sequential Convex Programming (SCP) method for use in structural optimization problems. This class of optimization problems come with a few unique details.

1. Structural optimization problems are generally expensive forward solves, involving a finite element solve.
2. Usually there is a reasonable way to find gradients, often through the adjoint state method, although this is still expensive.
3. Further, it's common to have terms that are divided by the optimization variables. For instance, consider a maximum stress constraint for a bar with area A, which is the optimization variable:

$$\frac{|F|}{A} - \sigma_{\max} \leq 0.$$

Keeping specifics these in mind, the MMA performs a Taylor Expansion in the inverse of the optimization variable (e.g. $\frac{1}{A}$) and uses the function and gradient information to build the coefficients in this approximation. Then, a relatively expensive Primal-Dual Interior Point Method is used to solve this subproblem, but this expense should be dwarfed in comparison to the forward finite element solve.

Subsequent work by Svanberg [2], [3] introduced the Globally Convergent Method of Moving Asymptotes (GCMMA), which adds iterations to construct a conservative approximation of the functions, still having a similar inverse-of-the-optimization-variable structure.

1.1 Relation to Class

MMA is an SCP, which we began to touch on in class, although we only provided the examples of Sequential Linear Programming and Sequential Convex Programming. That is the major connection, explaining where this paper would chronologically fit into the curriculum of the class. Some other details included throughout this paper that relate to the class are the following:

- Use of slack variables / reformulating a general problem,
- Convexity arguments for the subproblem,
- Brief discussion of Lagrangian, KKT conditions, and an Interior Point Method for solving the subproblem,
- Numerical comparisons against unconstrained optimization methods (Adam, as a modification of gradient descent with momentum), and using automatic differentiation / gradient-based methods

2 General Setup

Consider the general "Nonlinear Program form" of an inequality constrained minimization problem, given by (P). Note that equality constraints can be written in this form by taking $h_i(\mathbf{x}) = 0$ as $f_{m+2i-1}(\mathbf{x}) = h_i(\mathbf{x}) \leq 0$ and $f_{m+2i}(\mathbf{x}) = -h_i(\mathbf{x}) \leq 0$.

$$(P)\begin{cases}\min\limits_{\mathbf{x}\in\mathbb{R}^n} & f_0(\mathbf{x}) \\ \text{s.t.} & f_i(\mathbf{x})\le 0, \quad i=1,\dots,m \\ & \mathbf{x}^{\min}\le\mathbf{x}\le\mathbf{x}^{\max}\end{cases}$$

If we formulate (P) with slack variables t_i on the f_i inequalities, we get the following:

$$(P')\begin{cases}\min\limits_{\mathbf{x}\in\mathbb{R}^n,\mathbf{t}\in\mathbb{R}^m} & f_0(\mathbf{x})+\sum_{i=1}^m c_i t_i \\ \text{s.t.} & f_i(\mathbf{x})+t_i\le 0, \quad i=1,\dots,m \\ & \mathbf{x}^{\min}\le\mathbf{x}\le\mathbf{x}^{\max} \\ & \mathbf{t}\ge\mathbf{0}\end{cases}$$

With $c_i=\infty$ in (P'), we recover (P), but in practice, for large enough c_i we may find $\mathbf{t}=\mathbf{0}$ and thus recover the original problem. We can make on more change to (P') allowing for more cases:

$$(P'')\begin{cases}\min\limits_{\mathbf{x}\in\mathbb{R}^n,\mathbf{t}\in\mathbb{R}^m,z\in\mathbb{R}} & f_0(\mathbf{x})+a_0z+\sum_{i=1}^m (c_i t_i+\frac{1}{2}d_it_i^2) \\ \text{s.t.} & f_i(\mathbf{x})-a_iz-t_i\le 0, \quad i=1,\dots,m \\ & \mathbf{x}^{\min}\le\mathbf{x}\le\mathbf{x}^{\max}, \\ & \mathbf{t}\ge\mathbf{0} \\ & z\ge 0,\end{cases}$$

Where we also require $a_0>0$, $a_i\ge 0$, $c_i\ge 0$, $d_i\ge 0$, $c_i+d_i\ge 0$, and $a_ic_i>a_0 \quad \forall i: a_i>0$. Rewriting the problem in the form of (P'') allows for dealing with "special cases" such as $f_0(\mathbf{x})=\max_{p\in\{1,2,\dots,P\}}[f_{0,p}(\mathbf{x})]$ (see 3 bar truss example in appendix) and least squares.

3 Method

MMA is a Sequential Convex Programming (SCP) method. In other words, a convex approximation of the problem (P''), $(\tilde{P}^{(k)})$, is formed and solved for each iteration, and the solution $(\mathbf{x}^{(k)},\mathbf{t}^{(k)},z^{(k)})$ is used as the initial guess for the next iteration. The main approximation done is constructing $\tilde{f}_i^{(k)}(\mathbf{x})$ for $f_i(\mathbf{x})$ $(i=0,1,2,...,m)$, as shown in Eq. 1. A minor adjustment is replacing $\mathbf{x}^{\min}$ and $\mathbf{x}^{\max}$ with $\boldsymbol{\alpha}^{(k)}$ and $\boldsymbol{\beta}^{(k)}$, respectively. Similar to other first order SCP methods, MMA uses information from the function evaluations $f_i(\mathbf{x}^{(k)})$ and their gradients $\frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(k)})$ to construct the approximating functions. However, MMA also uses additional parameters $\mathbf{u}^{(k)},\mathbf{l}^{(k)}\in\mathbb{R}^n$ (the so-called "moving asymptotes") to construct this approximation. These parameters are updated based on the previous iterations, similar to other methods like BFGS that use the optimization history to construct approximates. However, the moving asymptotes act as both a learning rate and a control of the box constraints (rather than an approximation of the Hessian).

$$(\tilde{P}^{(k)})\begin{cases}\min\limits_{\mathbf{x}\in\mathbb{R}^n,\mathbf{t}\in\mathbb{R}^m,z\in\mathbb{R}} & \tilde{f}_0^{(k)}(\mathbf{x})+a_0z+\sum_{j=1}^m (c_j t_j+\frac{1}{2}d_jt_j^2) \\ \text{s.t.} & \tilde{f}_i^{(k)}(\mathbf{x})-a_iz-t_i\le 0, \quad i=1,\dots,m \\ & \boldsymbol{\alpha}^{(k)}\le\mathbf{x}\le\boldsymbol{\beta}^{(k)}, \\ & \mathbf{t}\ge\mathbf{0}, \\ & z\ge 0,\end{cases}$$

The approximates are given by

$$\tilde{f}_i^{(k)}(\mathbf{x}) = \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j} + \frac{q_{ij}^{(k)}}{x_j - u_j^{(k)}} \right) + r_i^{(k)}, \quad i = 0, 1, ..., m \tag{1}$$

Here, $p_{ij}^{(k)}$ and $q_{ij}^{(k)}$ contain the gradient information and are given by

$$p_{ij}^{(k)} = \underbrace{\left(u_j^{(k)} - x_j^{(k)}\right)^2}_{\geq 0} \left(\underbrace{1.001 \left(\frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(k)}) \right)^+}_{\geq 0} + \underbrace{0.001 \left(\frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(k)}) \right)^-}_{\geq 0} + \underbrace{\frac{10^{-5}}{x_j^{\max} - x_j^{\min}}}_{>0} \right) \tag{2}$$

$$q_{ij}^{(k)} = \left(x_j^{(k)} - l_j^{(k)}\right)^2 \left(0.001 \left(\frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(k)}) \right)^+ + 1.001 \left(\frac{\partial f_i}{\partial x_j}(\mathbf{x}^{(k)}) \right)^- + \frac{10^{-5}}{x_j^{\max} - x_j^{\min}} \right) \tag{3}$$

$$r_i^{(k)} = f_i(\mathbf{x}^{(k)}) - \sum_{j=1}^{n} \left(\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j^{(k)}} + \frac{q_{ij}^{(k)}}{x_j^{(k)} - l_j^{(k)}} \right) \tag{4}$$

Where we use the notation $(s)^+ := \max\{0, s\}$ and $(s)^- := \max\{0, -s\}$. Note that all terms of $p_{ij}^{(k)}$ are nonnegative and the last term is positive, making $p_{ij}^k > 0$. Also, consider $\frac{1}{u_j^{(k)} - x_j}$. If $x_j < u_j^{(k)}$, this function is convex, as $1/x$ is convex on $(0, \infty)$. Thus, the $\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j}$ terms in Eq. 1 are all strictly convex. By a similar argument, the $\frac{q_{ij}^{(k)}}{x_j - u_j^{(k)}}$ terms in Eq. 1 are also strictly convex. Further, $r_i^{(k)}$ is just a constant. Thus, because we are summing strictly convex functions, $\tilde{f}_i^{(k)}(\mathbf{x})$ is strictly convex for $\mathbf{l}^{(k)} < \mathbf{x} < \mathbf{u}^{(k)}$.

The modified bounds $\boldsymbol{\alpha}^{(k)}$ and $\boldsymbol{\beta}^{(k)}$ are chosen such that $\boldsymbol{\alpha}^{(k)} \leq \mathbf{x} \leq \boldsymbol{\beta}^{(k)}$ is equivalent to the following inequalities:

$$\begin{aligned} \mathbf{x}^{\min} \leq & \quad \mathbf{x} & \leq & \quad \mathbf{x}^{\max} \\ -0.9(\mathbf{x}^{(k)} - \mathbf{l}^{(k)}) \leq & \quad \mathbf{x} - \mathbf{x}^{(k)} & \leq & \quad 0.9(\mathbf{u}^{(k)} - \mathbf{x}^{(k)}) \\ -0.5(\mathbf{x}^{\max} - \mathbf{x}^{\min}) \leq & \quad \mathbf{x} - \mathbf{x}^{(k)} & \leq & \quad 0.5(\mathbf{x}^{\max} - \mathbf{x}^{\min}) \end{aligned} \tag{5}$$

Intuitively, this means that the $\boldsymbol{\alpha}^{(k)}$ and $\boldsymbol{\beta}^{(k)}$:

1. constrain $\mathbf{x}$ to be in the original box-constraint feasible region given by (P''),
2. keep the $u_j^{(k)} - x_j^{(k)}$ and $x_j^{(k)} - l_j^{(k)}$ from getting too small,
3. avoid becoming too far apart (relative to the original box-constraint) to keep the local approximation reasonable.

3.1 Moving Asymptotes

The moving asymptotes, $\mathbf{l}^{(k)}$ and $\mathbf{u}^{(k)}$, depend upon the history of the optimization. If the previous iterations were successful, these parameters move farther away from each other, allowing for more aggressive step sizes. Before a history has been established ($k < 3$), they are initialized as the following:

$$\begin{aligned} \mathbf{l}^{(k)} &= \mathbf{x}^{(k)} - 0.5(\mathbf{x}^{\max} - \mathbf{x}^{\min}), \\ \mathbf{u}^{(k)} &= \mathbf{x}^{(k)} + 0.5(\mathbf{x}^{\max} - \mathbf{x}^{\min}). \end{aligned} \tag{6}$$

Then, they are adjusted based on the previous two iterations by the following, where $\odot$ is the Hadamard / element-wise product,

$$\begin{aligned} \mathbf{l}^{(k)} &= \mathbf{x}^{(k)} - \boldsymbol{\gamma}^{(k)} \odot (\mathbf{x}^{(k-1)} - \mathbf{l}^{(k-1)}), \\ \mathbf{u}^{(k)} &= \mathbf{x}^{(k)} + \boldsymbol{\gamma}^{(k)} \odot (\mathbf{u}^{(k-1)} - \mathbf{x}^{(k-1)}), \end{aligned} \tag{7}$$

and with

$$\gamma_j^{(k)} = \begin{cases} 0.7, & (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) < 0, \\ 1.2, & (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) > 0, \\ 1, & (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) = 0 \end{cases} \tag{8}$$

Here, $\gamma_j^{(k)}$ can be thought of as an expansion or contraction factor for a given state variable x_j. If in the past two iterations x_j moved in the same direction both times, expand the moving asymptotes so that future approximations allow x_j to move more. On the other hand, if the past two iterations moved x_j in different directions, then contract the corresponding asymptotes so that the approximation is more local. In other words, if there are oscillations, reduce the stepsize, but if there are no oscillations, increase the stepsize. In practice, we also restrict the moving asymptote updates to be a "medium" size based on $x_j^{\max} - x_j^{\min}$. If the update barely moves the asymptote, instead move the asymptote a fixed, lower amount, but if the update moves the asymptote a lot, restrict this to a separate fixed, upper amount. This idea is visualized later.

3.2 Solving the Convex Subproblem

While we approximate the original problem with $(\tilde{P}^{(k)})$, we still must solve this subproblem, for each iteration k. We can rewrite $\tilde{P}^{(k)}$ as P^{sub} for arbitrary k, simplifying the notation a little bit by removing 'k', replacing '$\tilde{f}_i$' with '$g_i - b_i$', for $b_i = -r_i$, and dropping the constant b_0 from the objective. The subproblem solved at each iteration is

$$(P^{\text{sub}}) \begin{cases} \displaystyle\min_{\mathbf{x}\in\mathbb{R}^n, \mathbf{t}\in\mathbb{R}^m, z\in\mathbb{R}} & g_0(\mathbf{x}) + a_0 z + \displaystyle\sum_{j=1}^{m} (c_j t_j + \frac{1}{2} d_j t_j^2) \\ \text{s.t.} & g_i(\mathbf{x}) - a_i z - t_i \le b_i, \quad i = 1, \dots, m \\ & \boldsymbol{\alpha} \le \mathbf{x} \le \boldsymbol{\beta}, \\ & \mathbf{t} \ge \mathbf{0} \\ & z \ge 0. \end{cases}$$

3.2.1 Convexity

Recall that $\tilde{f}_i(\mathbf{x})$ is convex for $\mathbf{l} < \mathbf{x} < \mathbf{u}$, so g_i is as well. The set $\{\mathbf{x} \in \mathbb{R}^n : \boldsymbol{\alpha} \le \mathbf{x} \le \boldsymbol{\beta}\}$ is the intersection of two halfspaces, which are convex, making the set convex. Similarly, $z \ge 0$ and $\mathbf{t} \ge \mathbf{0}$ are halfspaces and are convex. The function $g_i(\mathbf{x}) - a_i z - t_i$ is convex in $(\mathbf{x}, \mathbf{t}, z)$ jointly and taking the $g_i(\mathbf{x}) - a_i z - t_i \le b_i$ is the intersection of the epigraph of the function with a another convex set. Next, the intersection of all the constraints, which are all convex, is itself convex. Under the previous assumptions of nonnegative constants a_0, c_j, d_j, the objective function is the sum of convex functions with nonnegative coefficients and is a convex function. Thus, because the objective is a convex function and the constraint set is a convex set, the problem (P^{sub}) is a convex problem.

Alternatively, we can compute the diagonal of the Hessian matrix as

$$\frac{\partial^2 g_i}{\partial x_j^2} = \frac{2p_{ij}}{(u_j - x_j)^3} + \frac{2q_{ij}}{(x_j - l_j)^3} \tag{9}$$

Further, with $l_j < \alpha_j \le x_j \le \beta_j < u_j$, and $p_{ij} > 0$, $q_{ij} > 0$ from before, g_i is strictly convex, as its Hessian is positive definite. Without getting into the specifics, under sufficient conditions, satisfying the KKT conditions of (P^{sub}) is sufficient for a point to also solve (P^{sub}), which we will assume.

3.2.2 Dual Variables

The primal variables are

- $\mathbf{x} \in \mathbb{R}^n$ (the original optimization variables),

- $\mathbf{t} \in \mathbb{R}^n$ (the slack variables for the original inequalities),
- $z \in \mathbb{R}$ (a problem generalizability variable).

Next, the dual variables corresponding to all of the constraints in (P^{sub}) are

- $\boldsymbol{\lambda} \in \mathbb{R}^m$ (for the function-prescribed constraints $g_i - a_i z - y_i \leq b_i$ for $i = 1, \dots, m$),
- $\boldsymbol{\xi} \in \mathbb{R}^n$ (for the upper box constraints $\mathbf{x} \leq \boldsymbol{\alpha}$),
- $\boldsymbol{\eta} \in \mathbb{R}^n$ (for the lower box constraints $\boldsymbol{\beta} \leq \mathbf{x}$),
- $\boldsymbol{\mu} \in \mathbb{R}^m$ (for the slack variables $\mathbf{t} \geq 0$),
- $\zeta \in \mathbb{R}$ (for the problem generalizability variable $z \geq 0$).

Thus, dualizing all constraints in problem (P^{sub}), the Lagrangian is

$$\begin{aligned}\mathscr{L}(\mathbf{x}, \mathbf{t}, z; \boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\mu}, \zeta) = & g_0(\mathbf{x}) + a_0 z + \sum_{j=1}^{m} (c_j t_j + \frac{1}{2} d_j t_j^2) \\ & + \sum_{i=1}^{m} \lambda_i \left(g_i(\mathbf{x}) - a_i z - y_i - b_i \right) \\ & + \langle \boldsymbol{\xi}, \boldsymbol{\alpha} - \mathbf{x} \rangle + \langle \boldsymbol{\eta}, \mathbf{x} - \boldsymbol{\beta} \rangle \\ & - \langle \mu, \mathbf{t} \rangle - \zeta z\end{aligned} \tag{10}$$

Let us group the functions into $\psi(\mathbf{x}, \boldsymbol{\lambda}) = g_0(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x})$. This is the only place involving explicit data from the original functions.

The list of the 18 KKT conditions are given by [3]. From Stationarity, there are three sets of equalities (one set each for $\mathbf{x}$, $\mathbf{t}$, and z). There are the same five sets of Primal Feasibility inequalities as in (P^{sub}), and the corresponding five Dual Feasibility and five Complementary Slackness sets of inequalities ($\boldsymbol{\lambda}, \boldsymbol{\xi}, \boldsymbol{\eta}, \boldsymbol{\mu}, \zeta$). These equations are solved with a Primal-Dual Interior Point Method (IPM), but could also be solved with another convex optimization method.

3.2.3 Primal-Dual Interior Point Method

Another set of slack variables, s_i, are introduced for the inequalities $g_i(\mathbf{x}) - a_i z - t_i \leq b_i$, while the rest of the inequalties (and now also that $s_i \geq 0$) are handled by the IPM. This can be thought of as loosening the equality (e.g. $\zeta z = 0 \to \zeta z = \varepsilon$, for $\varepsilon > 0$ and small). The IPM is applied to the original three stationarity equations, to the one new slack variable equation $g_i(\mathbf{x}) - a_i z - t_i + s_i - b_i = 0$, and then to the five perturbed complementary slackness equations, which have the ε IPM parameter on the right-hand-side, rather than 0. These nine sets of inequalities are then further simplified, as many gradients are constant and many variables are not coupled (there are lots of ones / diagonal blocks). The system reduces to only four sets of equations, for the $\mathbf{x}$, $\mathbf{t}$, z, and $\boldsymbol{\lambda}$ updates. The sparse solve gives a direction to step the many variables, and a linesearch is performed to avoid violating the remaining inequalities (primal and dual feasibility).

3.3 Globally Convergent Method of Moving Asymptotes

Getting into the theory and specifics of the Globally Convergent Method of Moving Asymptotes (GCMMA) is outside the scope of this project, but the general idea is to add an inner iteration to MMA in which the approximates are refined so that they are *conservative* approximations of the original functions, i.e. $f_i(\mathbf{x}^*) \leq \tilde{f}_i(\mathbf{x}^*)$ for $i = 0, 1, ..., m$, where $\mathbf{x}^*$ is the solution of the convex subproblem. Note that this *does* require an extra evaluation of the original function f_i at the new point $\mathbf{x}^*$. This information can theoretically be reused on the next iteration if that point is accepted, but this is not done in practice. However, we do not need to calculate gradients in these inner iterations, which is helpful for structural optimization problems where usually gradients are pretty expensive. The modification performed on these inner iterations is changing the 10^{-5} factor in the definitions of p_{ij} and q_{ij} (Eq. 2, 3). This positive factor gaurantees that the approximation functions are strictly convex, as shown before with $p_{ij} > 0$ and $q_{ij} > 0$, where the strict inequality depends on that 10^{-5} term for the case when the function gradient is zero.

4 Numerical Experiments and Demonstrations

We now investigate the performance of the MMA algorithm on a set of example problems.

4.1 Warmup

First, we will consider a trivial problem to demonstrate the method in a more intuitive and visual way than could be done for other applications.

4.1.1 Approximating Functions

As previously established, the approximates are convex, the strength of their convexity is given by their Hessian matrices (Eq. 9), which are diagonal. The smallest diagonal of this matrix gives the parameter, unless it is zero, then the approximates are not strongly convex. Further, as shown by the Hessian being diagonal, these approximates are separable, meaning that we can separately optimize the function over coordinates x_j and $x_{j'} \neq x_j$ equivalently to optimizing over the coordinates simultaneously.

As given in Eq. 9, the strength of the convexity of $\tilde{f}_i(\mathbf{x})$ depends on the original problem data (p_{ij}, q_{ij}) but also depend on the moving asymptotes u_j and l_j. Specifically, moving u_j and l_j has a cubic influence on the strength of convexity. On the other hand, moving the parameters farther apart makes the approximate closer to being affine (decreasing the strength of convexity). For the optimization, this results in the algorithm taking bigger steps. Visualizations of these approximates for $f(x) = x^2$ are given on the project Github. The first of the three gifs shows the default initialization of the moving asymptotes, which are both $\frac{x^{\max}-x^{\min}}{2}$ away from $x^{(0)}$ (Eq. 6). The second gif shows the approximating function for when both asymptotes are set at $\frac{x^{\max}-x^{\min}}{2\cdot 2}$ away from $x^{(0)}$, and the third gif uses $\frac{x^{\max}-x^{\min}}{2\cdot 5}$. We see that the first case is closer to being linear and would also give the most aggressive predictions for $x^{(1)}$. The approximates rely on the gradient information of $f(x)$. To establish a workflow for more numerous and more complicated functions, PyTorch is used, allowing for calculation of gradients via automatic differentiation.

4.1.2 MMA Iterations

The moving asymptotes expand and contract based on the success / oscillation of the previous two iterations. This is given by Eq. 7 and Eq. 8. If the two previous updates move x in the same direction, expand the moving asymptotes, giving a more aggressive approximation, both by the reduced convexity and by potentially expanding the move limits, corresponding to the α and β values used to enforce Eq. 5. At each iteration, the solution of the approximate problem, $(P)^{\text{sub}}$, is found and used as the next guess. This process is shown for a simple example problem below, with initial guess $x^{(0)} = -1.9$, on the project Github. Fig. 1 shows a summary of this optimization.

$$(P^{\text{ex}}) \begin{cases} \min\limits_{x \in \mathbb{R}} & x^2 \\ \text{s.t.} & -2 \leq x \leq 2 \end{cases}$$

Compared to other first order methods such as gradient descent, we seem to bounce around a lot, but the asymptotes do contract to decrease the magnitude of these oscillations. The subproblems are solved by a grid search, which is admissible for one dimension, certainly compared to implementing a Primal-Dual IPM. The behavior shown in Fig. 1 does not change if 100000 grid search points are used, rather than 100 grid search points. However, the behavior past iteration 20, (Fig. 2) changes substantially based on the number of grid search points. If we use 100, over the next 80 iterations the error climbs to close to $1e-2$. With 10000 grid search points, the error stays constant, not decreasing below $5e-5$, while the asymptotes oscillate back-and-forth every iteration. Lastly, using 100000 points, the error behaves the same as in the 10000 point

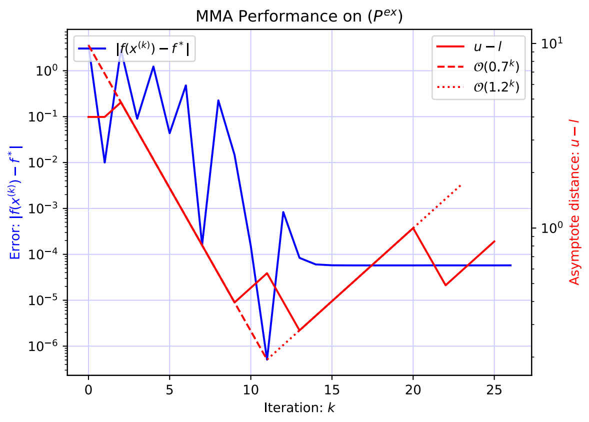

Figure 1: Performance of MMA on (P^{ex}), also showing the expansion / contraction of the moving asymptote parameters. Early in the optimization, oscillations are more pronounced, but as the asymptotes move to smaller sizes and as the guesses approach the solution, the oscillations are reduced.

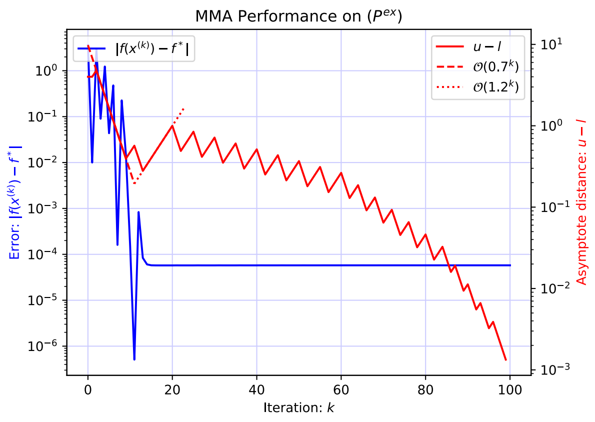

Figure 2: Performance of MMA on (P^{ex}), also showing the expansion / contraction of the moving asymptote parameters, over a larger number of iterations. For the subproblem solve, 100000 grid search points are used. Most importantly, we see that the error does not decrease past the level that was realized in the first 20 iterations.

case, but the asymptotes oscillate, but with different cycles (e.g. two iterations decreasing in size, then one iteration increasing), overall decreasing.

4.2 Neural Network Interpolation

Consider the task of fitting a neural network to some observations. A simple case of this is training a function $m(\xi; \mathbf{x}) = y$, $m : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$. The parameters of the neural network are given by $\mathbf{x} \in \mathbb{R}^n$, and the training data are $(\xi_l, y_l)_{l=1}^p$. The training data are artificially generated by $y = \sin(\pi \cdot \xi)$ for $p = 1000$ equispaced points $\xi \in [0, 1]$. We use a random 80% of this data for training and the other 20% for a final verification / test metric to evaluate overfitting. The optimization problem for training is the following:

$$(P^{\text{NN}}) \begin{cases} \min\limits_{\mathbf{x}\in\mathbb{R}^n} & \sum\limits_{l=1}^{0.80\cdot p} \left(m(\xi_l; \mathbf{x}) - y_l\right)^2 \\ \text{s.t.} & \mathbf{x}_{\min} \le \mathbf{x} \le \mathbf{x}_{\max} \\ & m(\xi_l; \mathbf{x}) + \varepsilon_{\min} \le 0 \quad l = 1, 2, ..., m \end{cases}$$

With slight abuse of notation, we take $m \le p$, and randomly select the data points $(\xi_l, y_l)_{l=1}^m$ where this (function) nonnegativity constraint is enforced. Note that we can also use z and a_i from (P'') to recast this Mean-Squared Error problem where the individual terms are added to the constraints, although this is too expensive for our exploratory investigation.

4.2.1 Training Experience

Using $m = 20$ (function) nonnegativity constraints, and a small 1 layer neural network with 10 units and tanh activations, the training is exceptionally slow. Some of this may be attributed to using PyTorch for the neural network evaluations and gradients then transferring to Numpy for the optimization parameter updates. However, there still is a very poor scaling in the number of constraints, m, although we would expect the transferring cost to only be linear in m. An example training is shown in Fig. 3 and 4.

The training is somewhat sensitive to the choices of $\mathbf{x}_{\min}$ and $\mathbf{x}_{\min}$. Taking $\mathbf{x}_{\min} = -\mathbf{x}_{\max}$ and varying

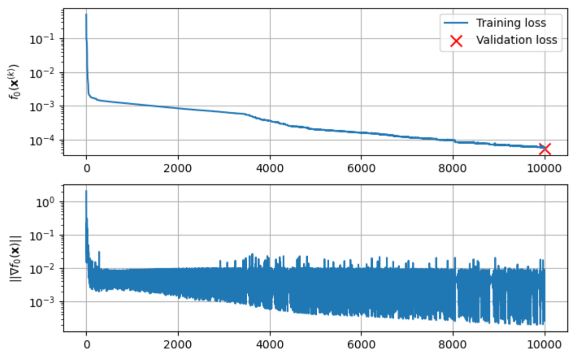

Figure 3: Using GCMMA for training a neural network on a 1D interpolation problem. The constraints are satisfied for all iterations, so that plot is omitted.

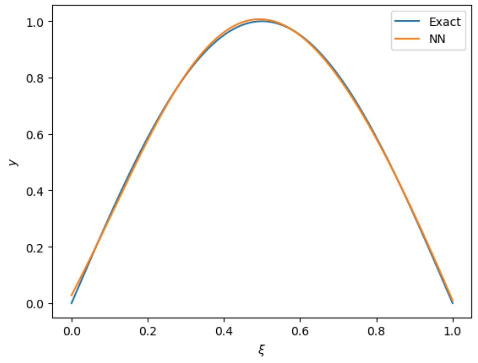

Figure 4: Resulting neural network trained from GCMMA. Despite many training iterations, the performance is still subpar, compared to Adam, which visually overlaps with the curve everywhere.

$\mathbf{x}_{\max}$ from 1 to 20 and training only a few hundred iterations shows a non-monotonic dependence on the box constraint bounds. Taking $\mathbf{x}_{\max} = 20$ gives worse performance than $\mathbf{x}_{\max} = 10$, but increasing from $\mathbf{x}_{\max} = 5$ to $\mathbf{x}_{\max} = 10$ provides a benefit for this problem.

We can also solve the unconstrained version of (P^{NN}) with a regular machine learning optimizer like Adam. Doing so, we quickly reach lower objective values than GCMMA. Then, continuing the training with GCMMA shows some interesting behavior. We would expect that if both the box and nonlinear constraints are satisfied, that continuing the training with GCMMA would remain at the low objective value. As expected, if either the box or nonlinear constraints have some violation, then the objective value increases orders of magnitude in a couple of GCMMA iterations. However, this also happens when there are many nonlinear constraints ($p = 1000$).

4.3 Other Examples

The appendix includes two other examples that were part of the presentation in class. The first example looks at a 3 bar truss (as done in the code associated with [3]). It serves as an example of a motivation for this method, and as a way to verify our PyTorch interface and gradient calculations. The second example is adversarial learning of an image. This is a good example for setting box constraints on our optimization variables (noise on the pixel values) and for using our PyTorch framework.

5 Conclusion

We motivated and presented the Method of Moving Asymptotes, which is a general first-order SCP, derived using Taylor Expansions in the inverse of the optimization variables. The moving asymptotes shift this inverse expansion left or right, also changing the convexity of the approximation. These moving asymptotes stabilize (accelerate) the optimization, based on the oscillations (monotonicity) in the past two iterations. We presented a handful of gifs demonstrating these concepts and the general logic behind MMA. We also briefly investigated the subproblem formed on each iteration, arguing that it is in fact convex and started on an explanation of how a Primal-Dual Interior Point Method is used to solve it. Then, we briefly gave the intuition behind the Globally Convergent of MMA (GCMMA), although omitted the technical details for the purpose of brevity. Using this method and Automatic Differentiation in PyTorch, we looked at three examples: training a Neural Network with positivity constraints, designing a 3 bar truss for minimum compliance, and training an adversarial layer of noise for fooling an image classifier. The structural optimization problem certainly justifies the motivation of the method and felt like the most "natural" use of MMA. However, using box constraints to restrict the value of the noise for the adversarial image also shows some promise.

Appendix

This section includes supplementary information, as a means to keep the paper length reasonable (for grading purposes). Lots of this was at least mentioned in the in-class presentation.

3 Bar Truss

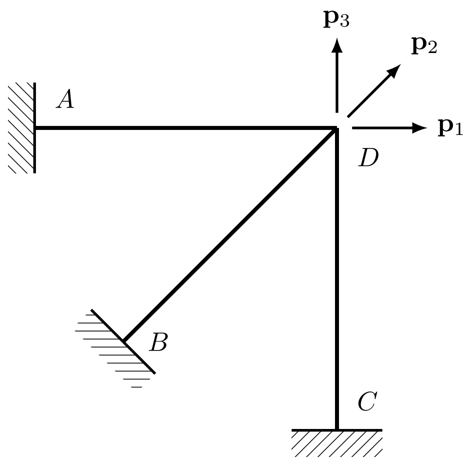

Figure 5: 3 Bar Truss Example configuration

As presented in [1], [3], we will consider a simple 3 Bar Truss. The Truss is given in Fig. 5 with three possible load configurations p_1, p_2, and p_3. This is a verification case from [3].

1. Bar 1 joins nodes A and D, with area x_1
2. Bar 2 joins nodes B and D, with area x_2
3. Bar 3 joins nodes C and D, with area x_3.

The objective is to find the areas $\mathbf{x}$ that minimize the maximum compliance over these three load cases, subject to individual area constraints ($x_i \geq 0.001$ for $i = 1, 2, 3$), and a joint mass constraint ($x_1+x_2+x_3 \leq 3$).

$$(P^{\text{Truss}}) \begin{cases} \min\limits_{x\in\mathbb{R}^3} & \sum\limits_{i=1}^{3} \mathbf{p}_l^T \mathbf{u}_l(\mathbf{x}) \\ \text{s.t.} & 0.001 \leq x \leq 3 \\ & \sum\limits_{i=1}^{3} x_i \leq 3 \end{cases}$$

$$\begin{aligned} \mathbf{K}(\mathbf{x})\mathbf{u}_l &= \mathbf{p}_i \quad l = 1, 2, 3 \\ \mathbf{K}(\mathbf{x}) &= \mathbf{R}(\text{diag}(\mathbf{x}))\mathbf{R}^T \end{aligned} \tag{11}$$

This example demonstrates what was alluded to in the introduction:

$$f_0(\mathbf{x}) \sim h\left(\frac{1}{x_1}, \frac{1}{x_2}, ..., \frac{1}{x_n}\right)$$

Adversarial Image Learning

As presented in [4], an imperceptible amount of noise can be added an image to fool neural networks. As we already have the framework set up for dealing with PyTorch models, this application is an interesting test problem, although unrelated to traditional applications of the algorithm or to concepts of convex optimization. However, adversarial image training does provide an excellent opportunity to use the special handling of box constraint provided by GCMMA. The optimization problem can be formulated as the following:

$$(P^{\text{Adv}}) \begin{cases} \min\limits_{x \in \mathbb{R}^{960 \cdot 1280 \cdot 3}} & -\operatorname{KL}\left(\sigma\left(\mathbf{m}(\mathbf{x}+\bar{\mathbf{v}})\right), \sigma\left(\mathbf{m}(\bar{\mathbf{v}})\right)\right) \\ \text{s.t.} & -0.1 \le x \le 0.1 \\ & \end{cases}$$

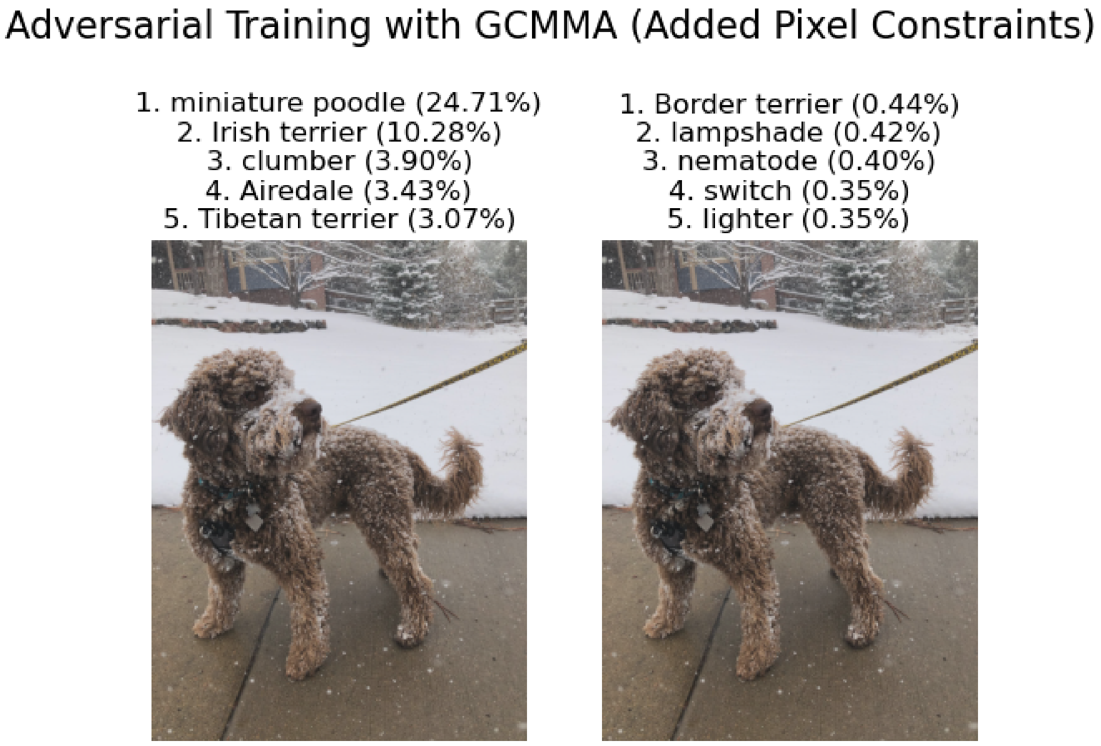
Adversarial Training with GCMMA (Added Pixel Constraints)
1. miniature poodle (24.71%)
2. Irish terrier (10.28%)
3. clumber (3.90%)
4. Airedale (3.43%)
5. Tibetan terrier (3.07%)
1. Border terrier (0.44%)
2. lampshade (0.42%)
3. nematode (0.40%)
4. switch (0.35%)
5. lighter (0.35%)

Figure 6: Training of adversarial noise on image, using GCMMA. GCMMA allows for the pixel values to be constrained. Dante is a good Italian Romagnolo. He is the best fetch player that we have ever witnessed.

Where σ is the elementwise softmax function, $\mathbf{m} : \mathbb{R}^{960 \cdot 1280 \cdot 3} \to \mathbb{R}^{1000}$ is the pretrained ResNet-50 Model for the ImageNet-1000 dataset, the original image is $\mathbf{v} \in [0,1]^{960 \cdot 1280 \cdot 3}$, and

$$\bar{\mathbf{v}} = 0.8\mathbf{v} + 0.1.$$

This shift is necessary, as $\max \mathbf{v} = 1$ and $\min \mathbf{v} = 0$, although based on results, the shift of 0.1 in each direction is unnecessarily large. Here we think of $\mathbf{x}$ as a layer of noise that we add to the original image. We are instead maximizing the loss that the model training would minimize. Using GCMMA on this optimization problem gives the results shown in Fig. 6.

Alternatively, we can use Adam as our optimizer, but doing this we drop the constraints on the pixel values from (P^{Adv}), and thus do not alter the original image $\mathbf{v}$. There are likely other ways of formulating the problem so that the pixel values stay in $[0,1]$, but this was not implemented. Fig. 7 shows the results of the Adam training. Note that we do not clip the inputs to $\mathbf{m}$, which is asking the model to extrapolate beyond the pixel values it was trained on (and beyond what are considered acceptable pixel values). Thus, these results are not a realistic comparison. The Adam noise layer has a min / max of -0.13 / 0.24, and the GCMMA noise layer has a min / max of -0.0059 / 0.0061. This shows that our shift of 0.1 could have been smaller, and that the Adam training is not really a fair comparison.

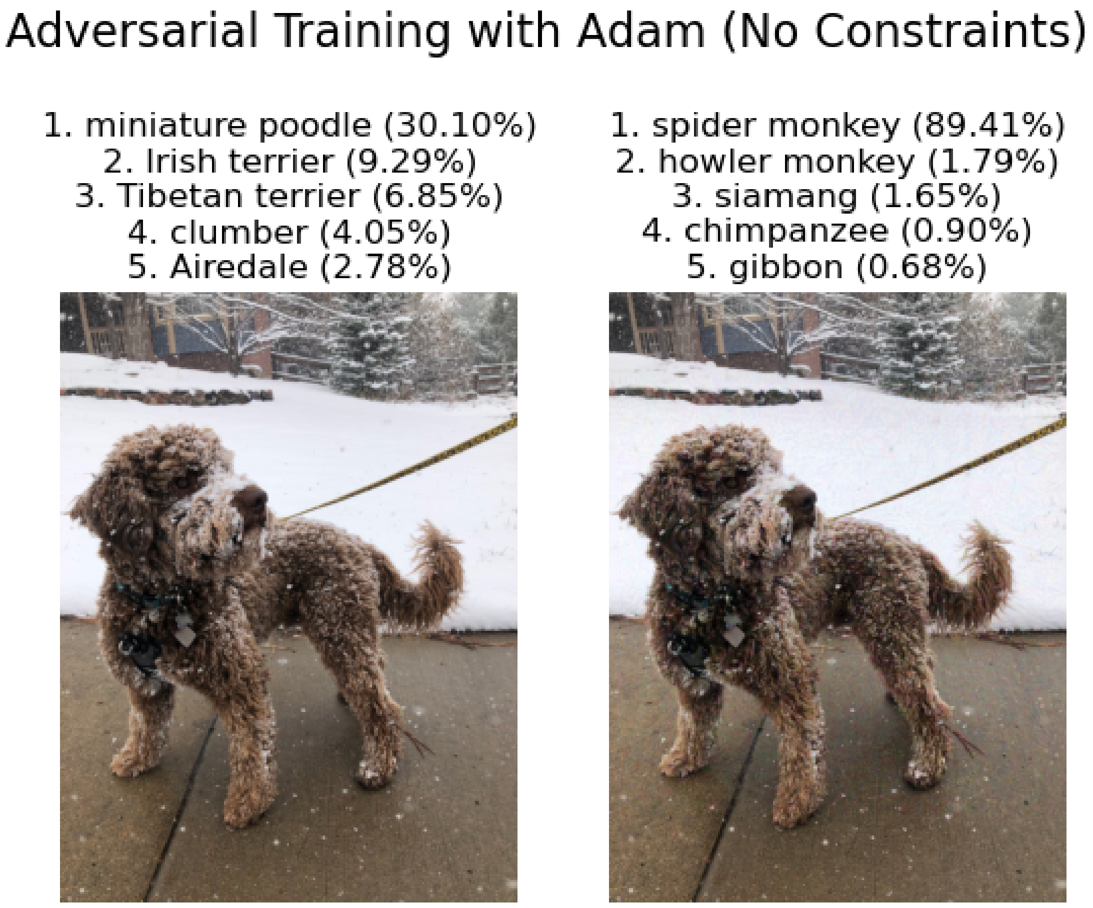

Figure 7: Training of adversarial noise on image, using Adam, with no constraints.

References

[1] Krister Svanberg. "The method of moving asymptotes—a new method for structural optimization". In: *International Journal for Numerical Methods in Engineering* 24 (2 1987), pp. 359–373. ISSN: 10970207. DOI: `10.1002/NME.1620240207`.

[2] Krister Svanberg. "A CLASS OF GLOBALLY CONVERGENT OPTIMIZATION METHODS BASED ON CONSERVATIVE CONVEX SEPARABLE APPROXIMATIONS". In: *Society for Industrial and Applied Mathematics* 12 (2 2002), pp. 555–573. URL: `http://www.siam.org/journals/siopt/12-2/36282.html`.

[3] Krister Svanberg. *MMA and GCMMA-two methods for nonlinear optimization.* 2007.

[4] Christian Szegedy et al. "Intriguing properties of neural networks". In: (2014).