MVEE Algorithm Derivation: Some real world data are high dimensional and approximately Gaussian distributed. Further, it is often useful to find the minimum volume affine hull that encloses these data. One natural way to do this is to compute the minimum volume enclosing ellipse, or MVEE. The goal is to compute the ϵ-proximal minimum volume affine hull of a finite set, $X \in \mathbb{R}^{d \times n}$. Here I will describe an existing MVEE algorithm developed previously [1] that is based on the founding work of Khachiyan [2] and others [3-4]. This repository provides a straightforward c++ implementation of it; I eventually plan to wrap this in a small header library, for now it just consists of a direct implementation.

The general form of an ellipsoid can be expressed as:

$$\varepsilon(E, \overline{x}) := \{x \in R^d : (x - \overline{x})^T E(x - \overline{x}) \leq d\} \tag{1}$$

where $\overline{x}$ is the centroid of the ellipsoid. The affine transformation E maps balls to ellipsoids and is symmetric (i.e., $E = E^T$) positive definite (i.e., $v^t E v > 0 : v \neq 0$). The volume of ε is:

$$Vol(\varepsilon) = v_0 \big(det(E)\big)^{-1/2} \tag{2}$$

where v_0 is the volume of the unit hypersphere. The MVEE optimization problem therefore aims to find the centroid $x_0 \in \mathbb{R}^d$ and transformation $E \in \mathbb{R}^{d \times d}$ that minimizes (2). The primal form is:

$$\begin{aligned} \min_{E,\overline{x}} \quad & \det(E^{-1}) \\ s.t. \quad & x_i^T E x_i \leq 1, \quad i = 1, 2, .., n, \\ & E > 0 \end{aligned} \tag{3}$$

which is not convex. Redefining (1) as $\varepsilon(E, \overline{x}) = \{x \in R^d : ||Ax - b||_2\}$, where $A = E^{1/2}$ and $b = E^{1/2} x_0$, we can rewrite this as

$$\begin{aligned} \min_{A,b} \quad & \text{-log } \det(A^{-1}) \\ s.t. \quad & ||Ax_i - b||_2 \leq 1, \quad i = 1, 2, .., n, \\ & A > 0 \end{aligned} \tag{4}$$

which is convex in the variables A and b. To solve this, it is desirable to restrict P to be centrosymmetric, but this constraint is of limited practical use. To solve this problem, Khachiyan used the projection $x \in \mathbb{R}^d \rightarrow q \in \mathbb{R}^{d+1}$ and solved the dual problem in R^{d+1}. We first rewrite (4) as

$$\begin{aligned} \min_{M} \quad & \text{-log } \det(M^{-1}) \\ s.t. \quad & q_i^T M q_i \leq 1, \quad i = 1, 2, .., n, \\ & A > 0 \end{aligned} \tag{5}$$

where $q_1, ..., q_n$ are the columns of $Q \in R^{(d+1) \times n}$, $Q^T = \big[X^T, \mathbf{1}\big]$, and $M \in R^{n \times n}$ is symmetric positive definite. Note that we still have an inverse in the log loss function. We therefore consider the Lagrangian of (5):

$$L(M, \lambda) = \text{-log } \det M + \sum_{i=1}^{n} \lambda_i \big(Q^T M Q - 1\big) \tag{6}$$

where $\lambda \in R^n$ is the Lagrange multiplier. Applying the KuhnTucker condition of optimality we get:

$$\begin{aligned} &\frac{\partial L}{\partial M} = -M^{-1} + Q\Lambda Q^T = 0 \\ &\therefore M = Q^T \Lambda Q \end{aligned} \tag{7}$$

where $\Lambda = \text{Diag}(\lambda)$. For clarity, we'll consider the simple variable substitution $u = \lambda/(d+1)$, yielding a tidy dual representation of (6).

$$\begin{aligned} \max_u \quad & \text{-log det}(QUQ^T) \\ s.t. \quad & \mathbf{1}^T u = 1 \\ & u \geq 0 \end{aligned} \tag{8}$$

where $U = diag(u) \in R^{n \times n}$. The objective in (8) is strictly concave, so we can use coordinate ascent methods [1] to solve it. We first take the gradient of the objective in (6) w.r.t. u:

$$\Omega(u) = \nabla(\text{log det}(QUQ^T)) = \frac{\partial \text{log det} QUQ^T}{\partial u} = Q^T U^{-1} Q \tag{9}$$

and use it in our update rule. Let $\overline{u}$ be the current value of u; at each step compute $\Omega(\overline{u})$ and solve the linear optimization problem:

$$\begin{aligned} \max_u \quad & \Omega(\overline{u}) \\ s.t. \quad & \mathbf{1}^T u = 1 \\ & u \geq 0 \end{aligned} \tag{10}$$

in the vicinity of $\overline{u}$. The solution is $e_j \in R^n$, where $j = argmax_j \Omega(\overline{u})$. At each step, we are pushing u in the direction of $\triangle u = e_j - \overline{u}$. Khachiyan [1] showed that the step size, α, is given by $max_{\alpha \in [0,1]} \text{log det}(QU(\overline{u} + \alpha(e_j - \overline{u})))$, which has a closed form solution:

$$\alpha = \frac{\Omega_j(\overline{u}) - (d+1)}{(d+1)(\Omega_j(\overline{u}) - 1)}. \tag{11}$$

Provided an optimal solution $\hat{u}$, we seek the corresponding primal solution, $(\hat{M})$, from which we can extract the desired ellipsoid, $(E, \overline{x})$. To recap, our ellipsoid exists in three forms:

$$\begin{aligned} MVEE &= \{x \in R^d : (x - \overline{x})^T \hat{E} (x - \overline{x}) \leq 1\} \\ &= \{x \in R^d : Q^T \hat{M} Q \leq 1\} \\ &= \{x \in R^d : Q^T \left(QUQ^T\right)^{-1} Q \leq 1\} \end{aligned} \tag{12}$$

It can be shown that $Q^T \left(QUQ^T\right)^{-1} Q = (x - Xu)^T E (x - Xu)$ (I'm omitting the proof for brevity); this allows us to extract $(\hat{E}, \hat{\overline{x}})$ from the dual optimzation problem defined in (8):

$$\hat{E} = \frac{1}{d}\left(X\hat{U}X^T - X\hat{u}\left(X\hat{u}\right)^T\right)^{-1} \qquad \hat{\overline{x}} = X\hat{u} \tag{13}$$

MVEE Algorithm: Below I provide pseudocode for the MVEE algorithm described above (all of the variables are kept the same for readability). The complexity of this algorithm is:

$$Runtime(MVEE) = O\left(nd^2\left((1+\epsilon)^{\frac{2}{d+1}} - 1\right)^{-1} + \ln d\right) \tag{14}$$

Thus, for low dimensional problems, a good implementation of this algorithm will run fast, computation on higher-dimensional data comes at a cost, though. In this problem $d = 2$, and because the algorithm scales linearly in n, we can sample points from the ellipse at varying degrees of sparsity to achieve given levels of performance given time constraints and vice versa.

Algorithm 1 MVEE Algorithm

Require: $\mathrm{X} = x_1, ..., x_n \in \mathbb{R}^d,\ \epsilon > 0.$

$\alpha, m, j \leftarrow 0,\ err \leftarrow \infty,\ e \leftarrow \mathbf{1} \in R^n,\ u, \overline{u} \leftarrow (1/n)\mathbf{1} \in \mathbb{R}^n,\ Q \leftarrow \left[P^T, \mathbf{1}\right]^T,\ U \leftarrow Diag(u), g \in R^n$

While$(err > \epsilon)$ **do:**

$g \leftarrow \left(Q^T(QUQ^T)^{-1}Q\right).diag(),\ j \leftarrow argmax\ g,\ m \leftarrow max\ g.$

$\alpha \leftarrow (j - d - 1)/((d+1)(m-1)), \overline{u} \leftarrow (1-\alpha)u$

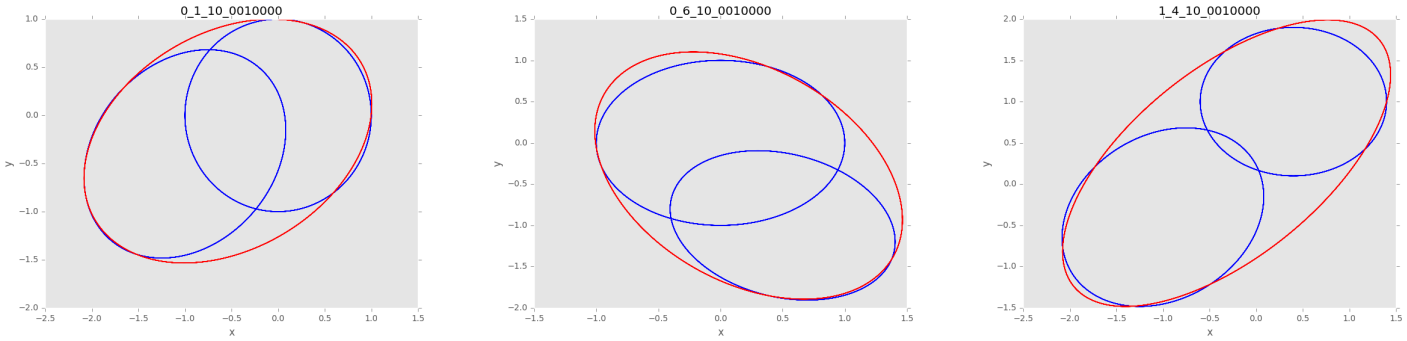
$\overline{u}(j) \leftarrow \overline{u}(j) + \alpha,\ err \leftarrow ||\overline{u} - u||_2, u \leftarrow \overline{u}$

End While

Return $(1/d)\left(\left(XUX^T - (XU)(XU)^T\right)^{-1}\right).$

Results: This algorithm works on a wide range of data. To demonstrate this, Figure 1 shows outputs of the algorithm when given points that define the locus (red colored) of two ellipses (blue colored) that I define in main.cpp. All of these were generated using $n = 20$ data points and with a tolerance of $\epsilon = 1e^{-2}$. The mean number of iterations to convergence for this parameter set was approximately 70, while the mean run time was approximately 5 milliseconds (2013 Intel Core i5). We can see that with one exception the results look very good. The poor fit on example 2-5-10 is most likely due to the low density of points being used. With only 10 points per ellipse, the likelihood of poles being excluded from the sample is higher than along the semi-major axis, which leads to this behavior. Overall, this algorithm works very well, even on the irregular data sampled from ellipse loci.

Figure 2 plots the run time (left) and number of iterations to convergence (right) versus number of points. Run time scales linearly with n, exactly as shown in (14). The number of iteration to convergence, on the other hand, scales logarithmically with n. Figure 3 plots the run time and number of iterations versus tolerance. We again find that run time follows (14), in this case it scales inversly with ϵ, as does the number of iterations to convergence.



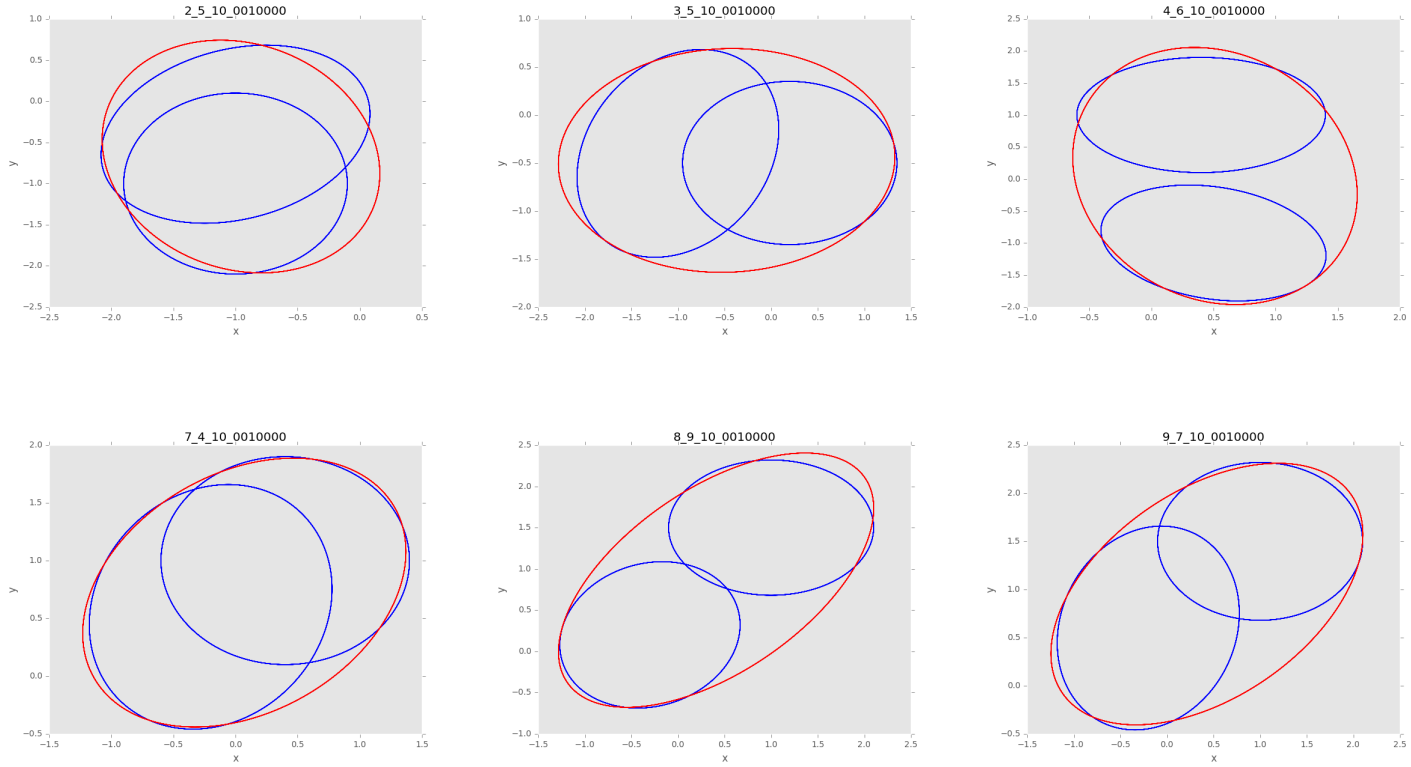


Figure 1: Example ellipses generated using the MVEE algorithm. The input ellipses are blue, bounding ellipse is red.

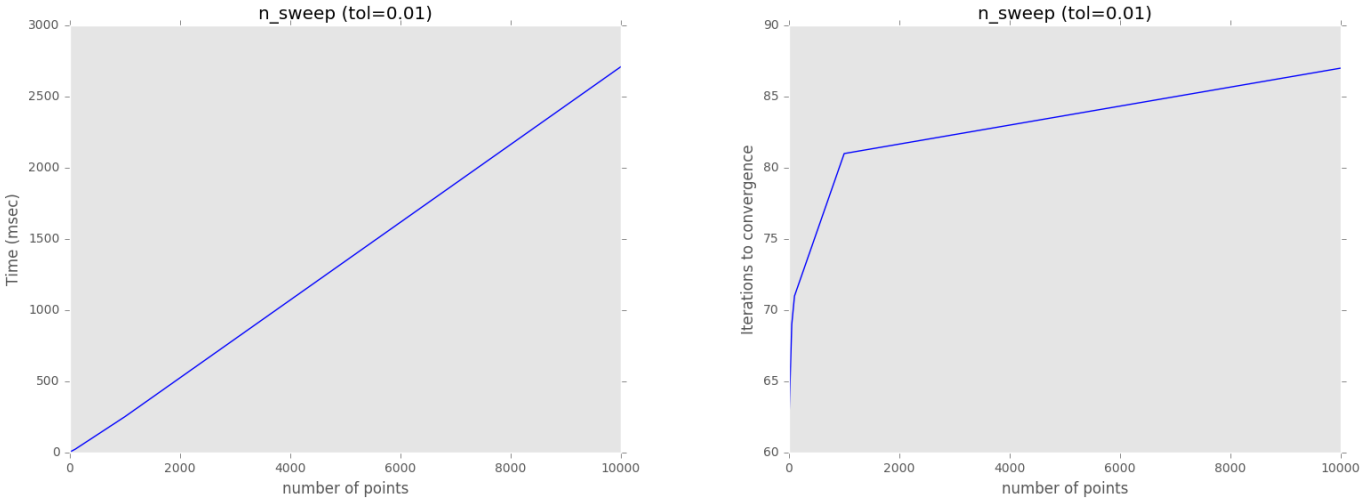


Figure 2: Plot of run time (left) and number of iteration (right) versus total number of points. We find that the runtime is linear in n, as expected per (14), while the number of iterations increases logarithmically with n.

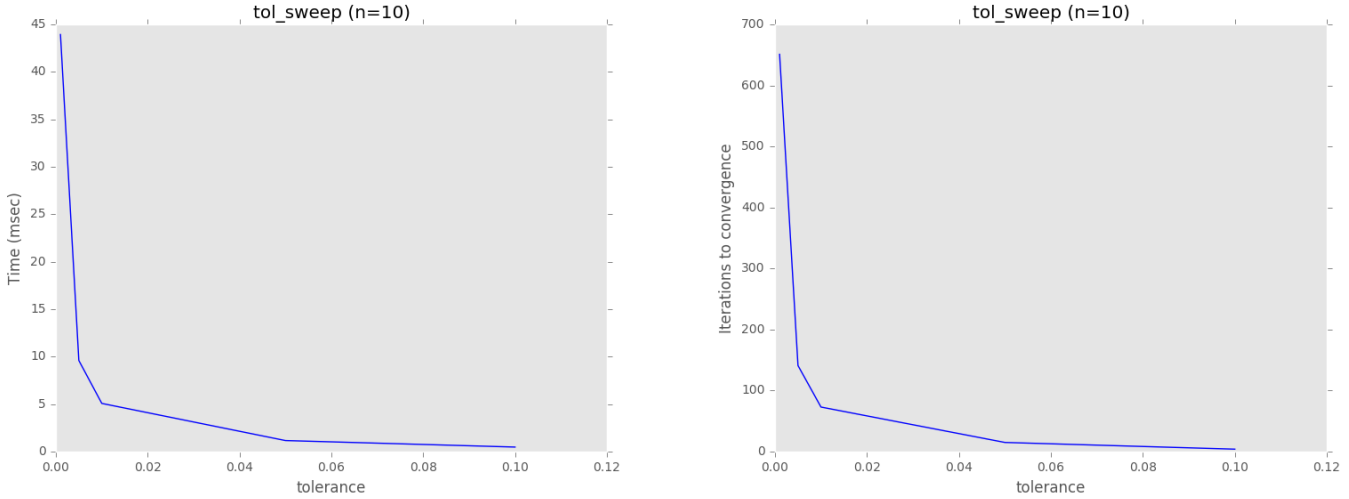


Figure 3: Plot of run time (left) and number of iteration (right) versus tolerance. The runtime scales inversly with ϵ, consistent with (14), while the number of iterations decreases inversely with higher tolerance.

References

[1] Khachiyan, Rounding of polytopes in the real number model of computation, Math. of Op. Res. (1996).

[2] Kumar et al., Mimimum volume enclosing ellipsoids and core sets (2005).

[3] Moshtagh, "Minimum volume enclosing ellipsoid." Convex Optimization (2005).

[4] Peng et al., Computation of minimum volume covering ellipsoids, Op. Res. (2004).