Generation of Parametric Aircraft Models from a Cloud of Points

Pengcheng Jia*and John F. Dannenhoffer, III†
Syracuse University, Syracuse, NY, 13244, USA

With the advent of feature-based parametric models, many new design and analysis techniques have been developed to take advantage of the model's features and design parameters. Examples of this are new grid generation techniques for Cartesian and overset methods. Unfortunately, many legacy geometries exist for which feature-based representations are not available. Proposed herein is a new technique for creating a parametric model from a legacy geometry that is defined in terms of a cloud of points that each lie on the aircraft's surfaces. First, for the points associated with any component, a variant of the Levenberg-Marquardt gradient-based optimization method (LM) is used to find the set of model parameters that minimizes the least-square errors between the model and the points. The efficiency of the LM algorithm is greatly improved through the use of analytic geometric sensitivities and sparse matrix techniques. Second, for cases in which one does not know *a priori* the correspondences between points in the cloud and the aircraft's components, an efficient initialization and classification algorithm is introduced. The technique is first explained with super-ellipse and isolated wing/fuselage configurations; then it is applied to aircraft configurations such as a glider. The accuracy and efficiency of these new techniques are demonstrated in both two and three dimensions, for configurations with both single and multiple components.

Nomenclature

a, b	super-ellipse radii
$cycle$	one (re-)initialization followed by several iterations
$\vec{d}$	design parameter vector
e_i	element in objective function
f_x, f_y	function for generating geometry model
$\vec{g}$	gradient vector of objective function
H	Hessian matrix
$iter$	each step within the LM optimization algorithm
J	Jacobian matrix
m	number of points in cloud
n	number of design parameters
q	components of vector to be minimized
r	power in super-ellipse
S	objective function
(u_i, v_i)	parametric coordinates
(x_p, y_p, z_p)	coordinates of points in cloud
(x_t, y_t)	points on super-ellipse
$\vec{\beta}$	vector of variables to be changed by optimizer, $(d_1 \cdots d_n, u_1 \cdots u_m)$
θ	rotation angle along z axis
λ	damping factor in LM method
$\delta x, \delta y, \delta z$	components of translation

*Ph.D. Candidate, Department of Mechanical and Aerospace Engineering, AIAA Member
†Associate Professor, Department of Mechanical and Aerospace Engineering, AIAA Associate Fellow.

Subscript

i	index of points in cloud
l	index of elements in objective function
j, k	index of parametric model parameters

Superscript

(s)	index of iterations

I. Introduction

A. Background

Modern computer-aided design (CAD) systems generally model complex aerospace systems through the creation of parametric, feature-based solid models. The parameters associated with these models are the key to automatic design optimization efforts. Also, such models are often key to streamlining the downstream analysis processes, such as grid generation.

Unfortunately, many aircraft that are currently flying were designed before the introduction of these parametric, feature-based modeling systems. Their computer-based representations consist of groups of surfaces, that may or may not be stitched together into a solid body. If the surfaces are not stitched together, a user who needs a watertight representation often has to stitch the surfaces together by hand, which is a very laborious and error-prone process.

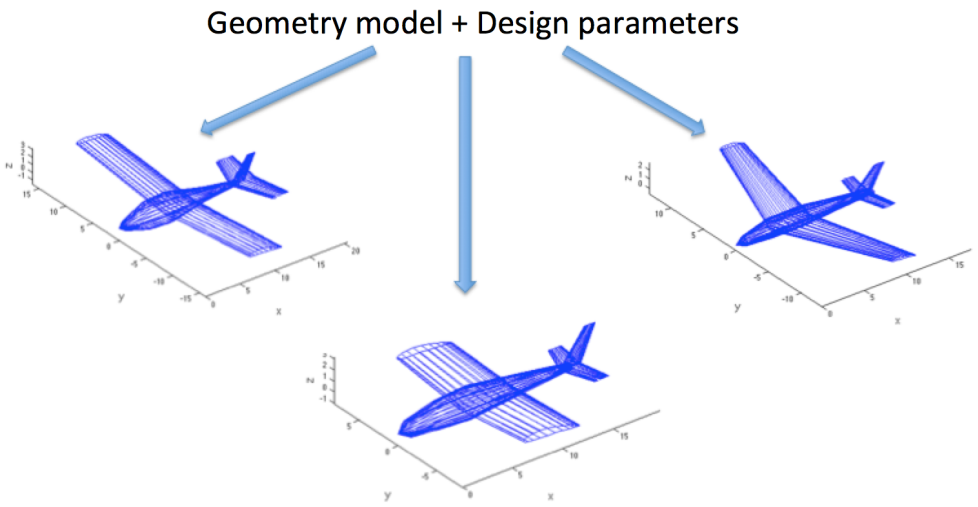
As a result, several researchers have developed techniques to convert the geometry into a cloud of points, and then fit the cloud; for example, freeform parametric surface (NUBRS) fitting has been developed. [1–6] However, use of this NUBRS-fitting technique lacks the connection back to the aircraft's real design parameters, such as wing sweep or aspect ratio. The fact that the representation is static means that it cannot be modified very easily, making it virtually unusable in design optimization.

Parametric models are also a key in reverse engineering, which is accomplished in three steps: part digitizing, feature extraction, and CAD modeling. [7–11] It plays an important role in reconstruction of a surface and significantly reduces the reconstruction time and the cost of part duplication [2, 12–15]. In reverse engineering, key research areas that still need further work include: improving data capture and calibration, coping with noise, merging views, coping with gaps in the data, reliable segmentation and fair surface fitting. [13] Roseline [11] presented an automatic and comprehensive retro-engineering process dedicated mainly to 3D meshes obtained initially by mechanical object discretization.

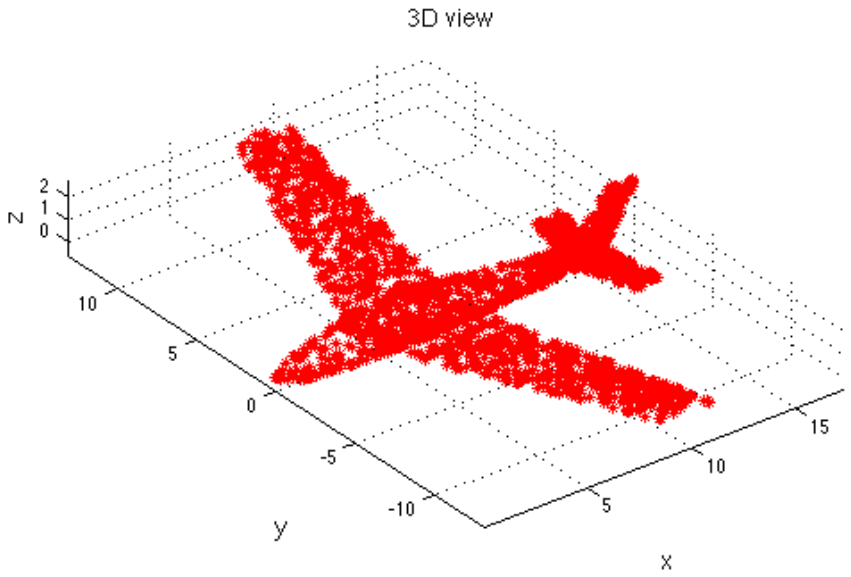
When parametric solid models are required, the first step is to generate a feature-based model that represents a whole class of similar configurations. An example of this is shown in Fig. 1a, which is a simple wing fuselage. Once the model has been developed, the next step is to find the set of parameters that "best matches" the legacy configuration (Fig. 1b).

To create a parametric model, McDonald [16] developed a technique that is effective for fitting a cloud of points to a configuration that has a single component; its use in general is limited by the effort associated with the classification process, which can be very inefficient and prone to errors.

The objective of this work is to generate parametric models efficiently and accurately from a cloud of points. This process uses the Levenberg-Marquardt (LM) method, which is a gradient optimization technique that is particularly well-suited to least-squares problems.



(a) Several candidate configurations made from the same feature tree but with different design parameters, $\vec{d}$



(b) Cloud of points

Figure 1: Overall problem of finding the model's parameters that best-fit a cloud of points

B. Outline

The paper begins by applying the LM method to a configuration that is composed of a single component. After describing the optimization problem to be solved, the paper describes the implementation of the LM method that is used. It turns out that a key to making the LM method robust is the need for clever initialization (and re-initialization) throughout the process; these are described next.

The paper then extends this work to configurations that are composed of multiple components. This

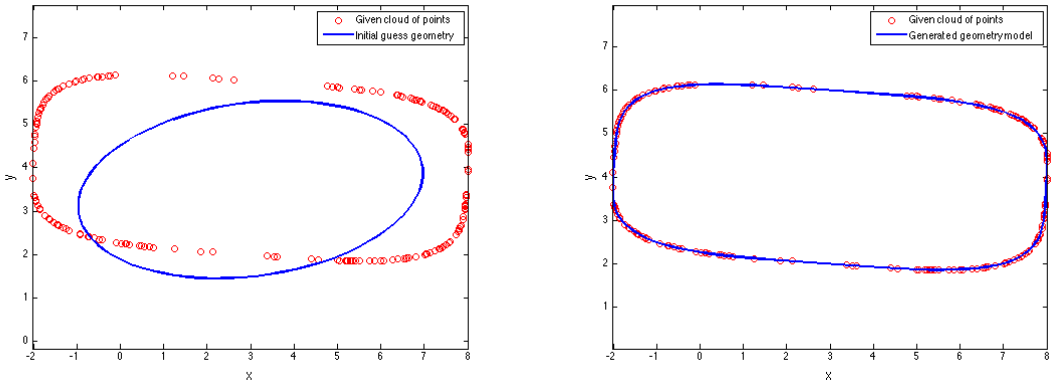
process involves two processes: the classification of points in the cloud that are associated with each component (such as wing, fuselage, ...), and the determination of the components' parameters that best fit the points in the cloud.

II. Single Component Configuration

To begin, the generation of a parametric model for a single component from a cloud of points is introduced. In two dimensions, the single component can be any parametrized shape, such as a circle, rectangle, super-ellipse, or airfoil. In three dimensions the component can be a box, super-ellipsoid, fuselage, or wing. For ease of understanding, the super-ellipse is used to demonstrated the technique in the following sections.

A. Sample Problem

The basic single component problem is demonstrated by a super-ellipse. Shown in Figure 2 is an example, where the blue line in (a) is the initial guess and the red points are the points in the cloud. The blue line in (b) is the final parametric model which is generated from the technique. Here, the model's parameters describe the shape of and location of the super-ellipse. The task at hand is to find a method that starts from (a) and ends at (b).



(a) Cloud of points (red) and initial guess (blue)

(b) Cloud of points (red) and final model (blue)

Figure 2: Sample problem, demonstrated with a super-ellipse.

1. *Design Parameters*

For the super-ellipse model, the elements of the design parameter vector, $\vec{d}$, are the major and minor radii, (a) and (b), the super-ellipse power, (r), the center point, $(\delta x, \delta y)$, and the rotation angle (θ). In the discussion that follows, the length of $\vec{d}$ is n; therefore $n = 6$ when $\vec{d} = (a, b, r, \delta x, \delta, \theta)$.

The defining function for a super-ellipse centered at the origin is

$$\begin{aligned} x_t &= |\cos u|^{2/r} \cdot a \cdot \text{sign}(\cos u) \\ y_t &= |\sin u|^{2/r} \cdot b \cdot \text{sign}(\sin u) \end{aligned} \tag{1}$$

where u is a parametric coordinate around the super-ellipse, with $0 \leq u \leq 2\pi$. These coordinates can then be transformed using homogeneous coordinates, as in

$$\begin{bmatrix} f_x \\ f_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} \tag{2}$$

2. Objective Function

The objective function, that is the quantity to be minimized, is the least-square distances between the points in the cloud and the surface of the parametric model. Therefore, the objective function is to simultaneously minimize $e_{(i)}$ for each point i in the cloud

$$e_{(i)} = (x_{p(i)} - f_x(\vec{d}, u_i))^2 + (y_{p(i)} - f_y(\vec{d}, u_i))^2 \tag{3}$$

where $x_{p(i)}$ and $y_{p(i)}$ are the x- and y-coordinates of the ith point in the cloud and $f_x(\vec{d}, u_i)$ and $f_y(\vec{d}, u_i)$ are the coordinates of a point on the super-ellipse associated with a parametric coordinate u_i, where there are m points in the cloud.

Since one does not know beforehand where on the surface of the super-ellipse each point in the cloud maps, m parametric coordinates, u_i, need to be added to the optimization's design variables; the elements of $\vec{d}$ and the u_i will be automatically adjusted during the optimization process.

Hence, the objective function is

$$S = \sum_{i=1}^{m} e_{(i)} = \sum_{i=1}^{m} (x_{p(i)} - f_x(\vec{d}, u_i))^2 + (y_{p(i)} - f_y(\vec{d}, u_i))^2 \tag{4}$$

Writing the objective function in standard least-square format, $e_{(i)}$ can be decomposed into parts associated with the x- and y-terms above, giving

$$S = \sum_{l=1}^{2m} q_l^2 \tag{5}$$

where

$$q_l = \begin{cases} x_{p(l)} - f_x(\vec{d}, u_l) & \text{for} \quad l <= m \\ y_{p(l-m)} - f_y(\vec{d}, u_{l-m}) & \text{for} \quad l > m \end{cases} \tag{6}$$

An example of a super-ellipse is shown in Figure 3, where the the cloud of points are shown in red, a super-ellipse as a green line, closest points in blue, and the distances to the surface in black. The objective minimizes the sum of the lengths of the black lines.

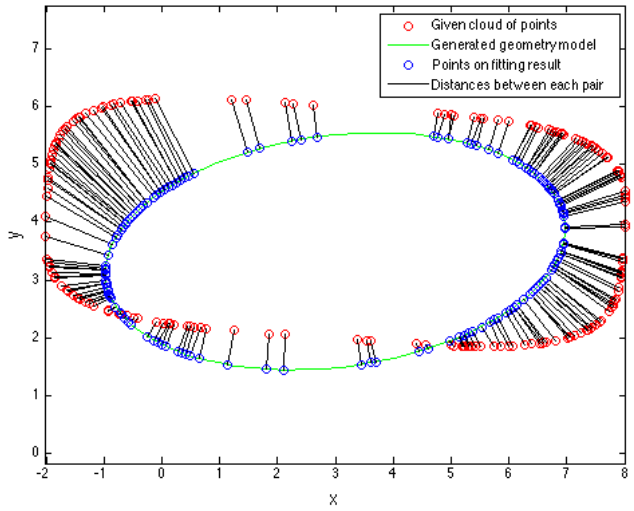


Figure 3: Cloud of points and sample super-ellipse.

3. Initialization

Given a set of design parameters $\vec{d}$, one needs to find the initial values for the u_i. Doing this well is important because if the initial guess is far from the correct answer, there is a high probability that the

optimization process will get stuck in a local minimum, as will be shown below. Even it does not get stuck at a local minimum, bad initializations will make the process very time consuming.

For the super-ellipse, it is reasonable to size and place it based upon the cloud of points. Specifically, the initialization technique used here is

$$\begin{aligned} a_{ini} &= |x_{pmax} - x_{pmin}|/2 \\ b_{ini} &= |y_{pmax} - y_{pmin}|/2 \end{aligned} \tag{7}$$

and

$$\begin{aligned} \delta x_{ini} &= \frac{(x_{pmax} + x_{pmin})}{2} \\ \delta y_{ini} &= \frac{(y_{pmax} + y_{pmin})}{2} \end{aligned} \tag{8}$$

where x_{pmin} and x_{pmax} are the minimum and maximum values of $x_{p(i)}$ in the cloud. The initial guess of the power r and rotation θ will not impact the location of the result very much, so r is chosen arbitrarily as 2 (a regular ellipse) and θ as 0.

To initialize the point parameters u_i, a discrete representation of the parametric model is created and then u_i is set to the u at the discrete point that is closest to point i in the cloud.

B. Optimization Technique

There are many approaches to optimization that can be applied. Fortunately, the problem here is defined analytically, and hence a gradient-based technique can be used.

1. Levenberg-Marquardt Method

The Levenberg-Marquardt method [17] is an optimization method which is used to minimize least square problems. It is a combination of Newton's method and steepest descent, making it both fast and robust.

Newton's method is an iterative method for finding the root of a differentiable function S. In optimization, Newton's method is applied to the derivative S' of a twice differentiable function S to find the root of S'. Once the root of S' is found, the maximum or minimum value of the objective function S is found.

Newton's method, to drive S' to zero, can be written as

$$\vec{\beta}^{(s+1)} - \vec{\beta}^{(s)} = -[H(\vec{\beta}^{(s)})]^{-1} \cdot \vec{g}(\vec{\beta}^{(s)}) \tag{9}$$

where $\vec{\beta}$ are the design variables, $\vec{g}$ is the first derivative of S in vector format, and H is the Hessian matrix of S. For the current problem, $\vec{\beta} = (d_1 ... d_n, u_1 ... u_m)$. For simplicity, it is useful to define

$$\vec{\delta} \equiv \vec{\beta}^{(s+1)} - \vec{\beta}^{(s)} \tag{10}$$

Note that $\vec{\delta}$ is $(m+n) \times 1$, $\vec{g}$ is $(m+n) \times 1$, and H is $(m+n) \times (m+n)$.

By taking the advantage of the least squares structure of objective function, one can write

$$g_j = 2 \sum_{l=1}^{2m} q_l \frac{\partial q_l}{\partial \beta_j} \tag{11}$$

$$H_{jk} = 2 \sum_{l=1}^{2m} \left(\frac{\partial q_l}{\partial \beta_j} \frac{\partial q_l}{\partial \beta_k} + q_l \frac{\partial^2 q_l}{\partial \beta_j \partial \beta_k} \right) \tag{12}$$

Now define $J \equiv \partial q_l / \partial \beta_j$, which is the Jacobian matrix of $\vec{q}$. (Recall the $\vec{g}$ is the gradient of q^2.) Marquardt assumed the second-order derivative terms of H could be ignored

$$H_{jk} = 2 \sum_{l=1}^{2m} J_{lj} J_{lk} \tag{13}$$

Combining these gives

$$\vec{\delta} = -(J^T J)^{-1} \cdot J^T \vec{q} \tag{14}$$

If the initial guess starts very far from the final minimum, Newton's method may not work; hence Marquardt suggested adding a term that employs ideas from the gradient descent method (since it only uses first derivatives). This is done by adding a damping parameter, λ, to the iteration function that can be changed based on the result in each step

$$\vec{\delta} = -(J^T J + \lambda I)^{-1} \cdot J^T \vec{q} \tag{15}$$

where I is the identity matrix.

When $\lambda \to 0$, the added term vanishes and the technique reverts to Newton's method. When λ becomes large, the scheme becomes the gradient descent method. This improves the robustness of the algorithm.

Marquardt suggested starting with a small value for λ. If the results of the previous step improves the objective function, (that is, $||\vec{e}^{(new)}|| < ||\vec{e}^{(old)}||$) $\vec{\beta}$ is incremented by $\vec{\delta}$ and the the value of λ is decreased (say by a factor of 2) and the method continues. If the method (unfortunately) increases the objective function, the step is discarded and λ is increased.

Marquardt also provided the insight that one can scale each component of the gradient according to the curvature so that there is larger movement along the directions where the gradient is smaller. This avoids slow convergence in the direction of small gradient. Therefore, the identity matrix I was replaced with the diagonal matrix consisting of the diagonal elements of $J^T J$, yielding

$$\vec{\delta} = -(J^T J + \lambda \cdot \mathrm{diag}(J^T J))^{-1} \cdot J^T \vec{q} \tag{16}$$

There are two stopping rules for interrupting the iteration. One is the number of iteration exceeds the maximum number of iterations, which is defined by the user. The other one is when the $||\vec{\delta}||$ becomes smaller than a specified tolerance, ϵ.

2. Improved Levenberg-Marquardt Method

Sometimes the optimization processes is slow, because the parameters have not been updated during many iterations; this means that many steps were rejected and λ was increased. The original LM method also has a high probability of getting stuck in a local minimum due to the monotonic updating rule.

To circumvent these problems, a modification to the LM method is introduced, which borrows some ideas from the simulated annealing method (SA). In SA, which is a stochastic optimization technique, candidate solutions are randomly generated and accepted if the new objective is "not too much worse" than the previous objective. The definition of "not too much worse" starts rather loosely and then tightens up as the method continues; this idea is inspired by the annealing process for metals and results in the SA method being less prone to getting stuck in a local minimum.

Here the same idea is added to the LM method. Specifically, instead of accepting steps only if the objective improves, one can accept new results that are "not too much worse", or a step is accepted if

$$||\vec{e}^{(new)}|| < e^{||\vec{\delta}||} \, ||\vec{e}^{(old)}||$$

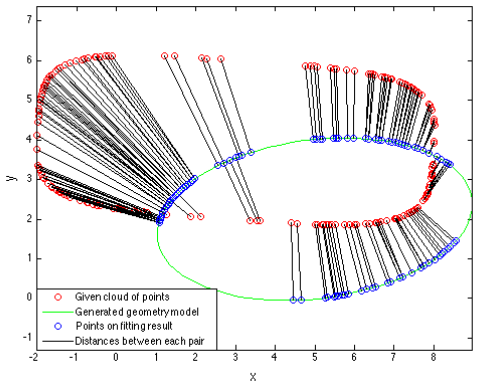
Since $||\vec{\delta}||$ approaches 0 as the solution is neared, the factor $e^{||\vec{\delta}||}$ approaches 1, and only better steps are accepted during the endgame (as was done in the SA method).

The fitting results based on different updating rules will be shown in the Result section in Figure 20.

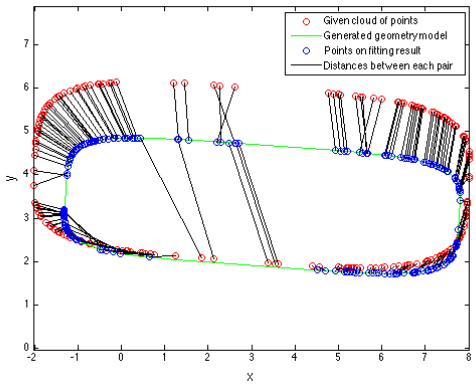
C. Reinitialization

After applying the technique introduced above to many cases, it was found that the optimizer got stuck at a local minimum in some cases, even with the SA-inspired improvements described above. This happens when the initial u_i happens to be on the wrong side of the body. In these cases, the optimizer will not change the u_i, since doing so locally would result in an objective function increase. Examples of this can be seen in Figure 4a, where the distances from each point to its closest point on the super-ellipse is shown. Figure 4b is the result after optimization based on the initialization, which shows that the optimizer got stuck at a local minimum.

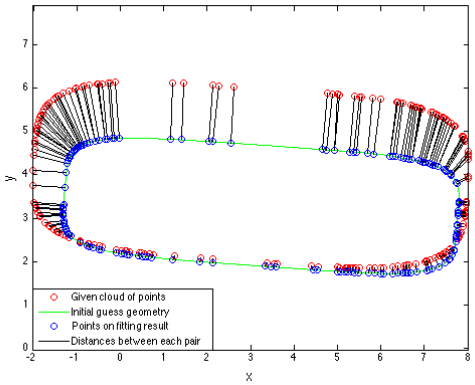
To solve this problem , one can simply reinitialize the u_i, while keeping the best design parameters $\vec{d}$. The reinitialization technique generates a new sequence of u_i. This will overcome the local minimum problem and the reinitializing result is shown in Figure 4c. The new optimization result is shown in Figure 4d



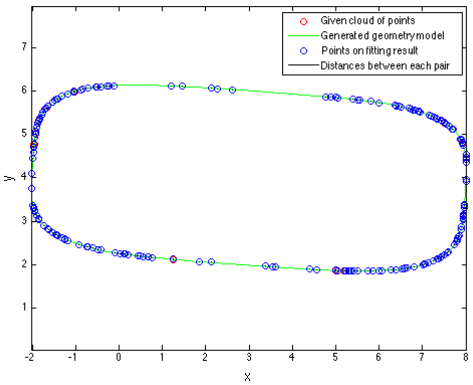
(a) Initialization



(b) Optimization stuck at local minimum



(c) Immediately after reinitialization



(d) Final optimization result

Figure 4: The effects of reinitialization on the fitting result

D. Sparse Technique

To improve the computational efficiency of the technique, one can see that the Jacobain matrix is very sparse, as shown in equation 17. For the first $2m \times n$ block, all the columns that correspond to the derivatives with respect to the design parameters, $\vec{d}$, are not generally zeros. But the second $2m \times m$ block of the Jacobain is comprised of two diagonal matrices, stacked one above the other, which correspond to the derivatives with respect to the parametric coordinate of that point, u_i.

$$J = \left[\begin{array}{ccc|cccc} \frac{\partial f_{x(1)}}{\partial d_1} & \cdots & \frac{\partial f_{x(1)}}{\partial d_n} & \frac{\partial f_{x(1)}}{\partial u_1} & 0 & \cdots & 0 \\ \frac{\partial f_{x(2)}}{\partial d_1} & \cdots & \frac{\partial f_{x(2)}}{\partial d_n} & 0 & \frac{\partial f_{x(2)}}{\partial u_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{x(m)}}{\partial d_1} & \cdots & \frac{\partial f_{x(m)}}{\partial d_n} & 0 & \cdots & \cdots & \frac{\partial f_{x(m)}}{\partial u_m} \\ \frac{\partial f_{y(1)}}{\partial d_1} & \cdots & \frac{\partial f_{y(1)}}{\partial d_n} & \frac{\partial f_{y(1)}}{\partial u_1} & 0 & \cdots & 0 \\ \frac{\partial f_{y(2)}}{\partial d_1} & \cdots & \frac{\partial f_{y(2)}}{\partial d_n} & 0 & \frac{\partial f_{y(2)}}{\partial u_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{y(m)}}{\partial d_1} & \cdots & \frac{\partial f_{y(m)}}{\partial d_n} & 0 & \cdots & \cdots & \frac{\partial f_{y(m)}}{\partial u_m} \end{array}\right] \tag{17}$$

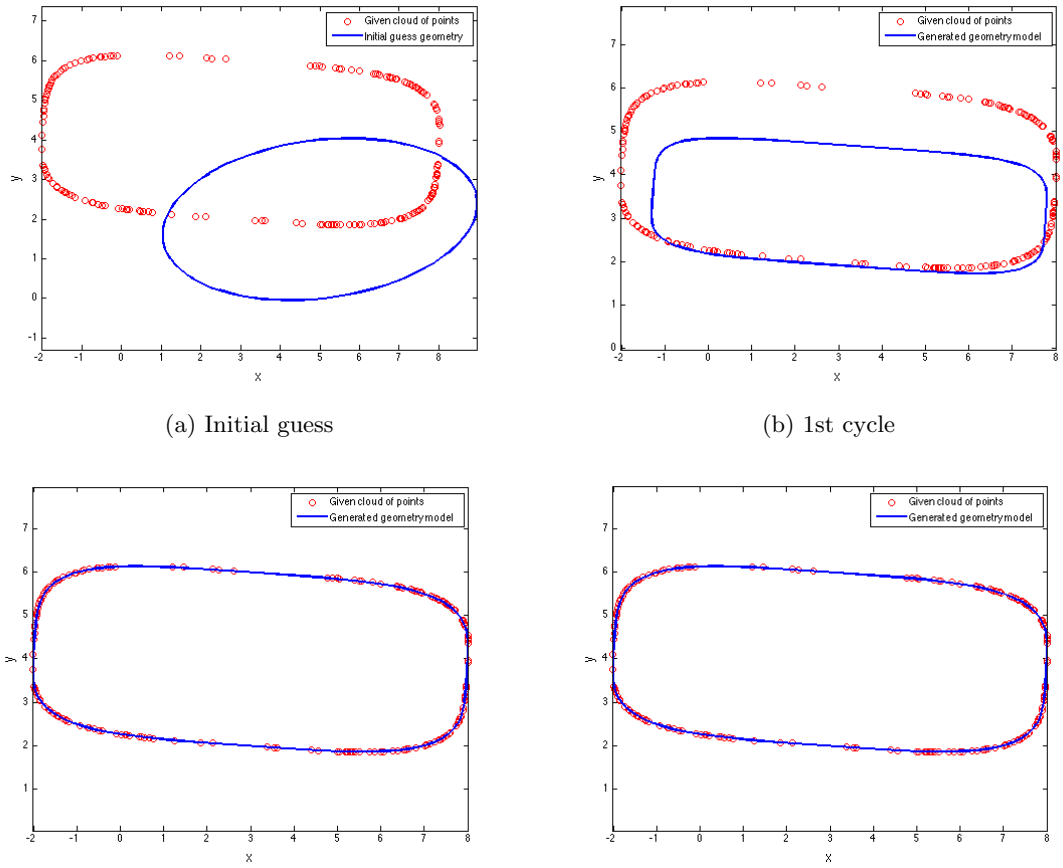
Hence, sparse matrix methods can be employed to great advantage. Additionally, since J only contains derivatives, if one computes the derivative analytically (for example, by using the techniques described in [18], additional time can be saved by not having to generate J via finite differences.

III. Single Component Configuration

A. Single Component in Two Dimensions

In the following discussion, "iteration" refers to one step of the improved LM method and "cycle" refers to a (re-)initialization followed by up to 20 "iterations".

The results after several cycles during the fitting of a super-ellipse are shown in Figure 5. The red points are the points in the cloud and the blue lines are the parametric super-ellipse model. Figure 5a is the super-ellipse which is generated by an initial guess of the design parameters. The figure shows the results for the first 3 cycle of up to 20 iterations each. From the figure, we can see that the correct parametric super-ellipse can be obtained after only 2 cycles.



(a) Initial guess

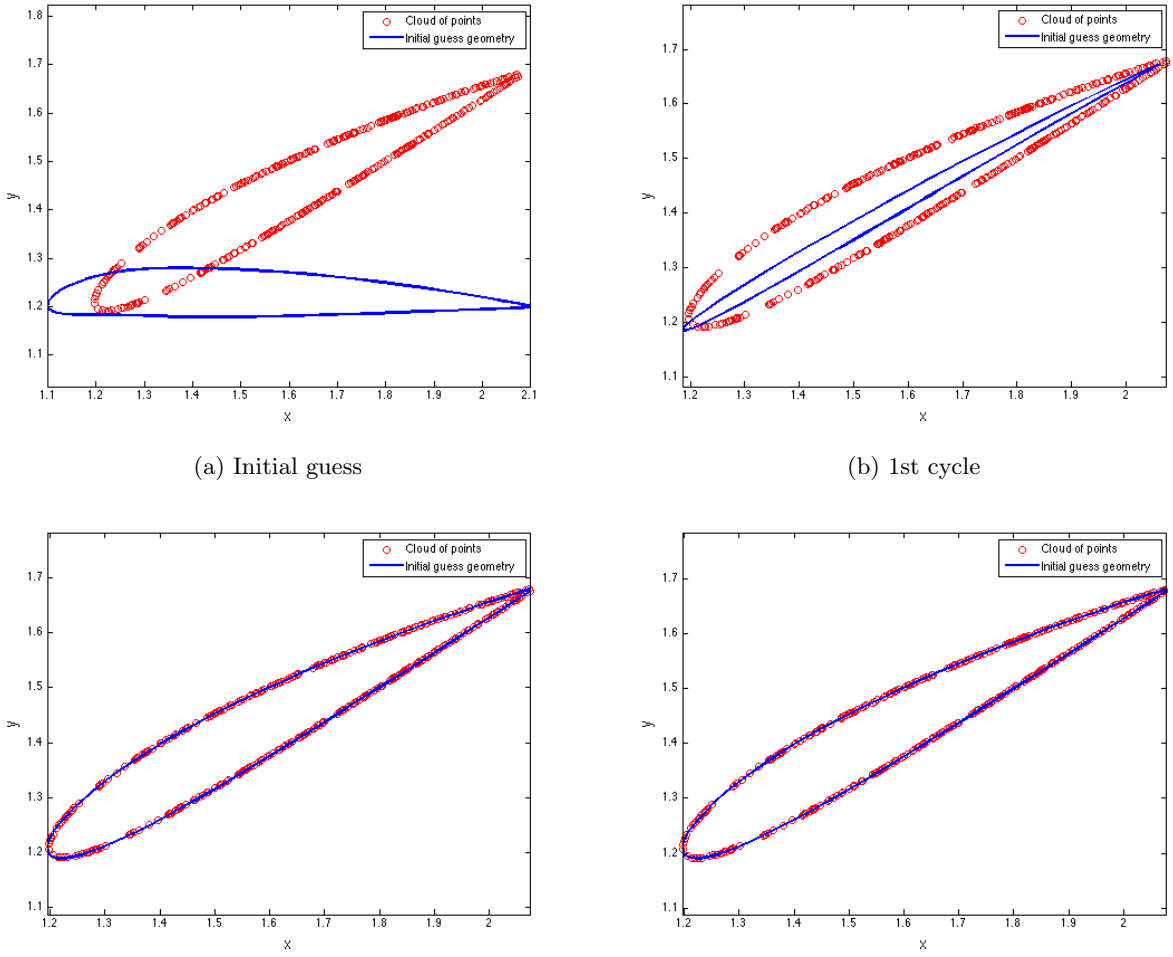
(b) 1st cycle

(c) 2nd cycle

(d) 3rd cycle

Figure 5: Generation of parametric model for 2D super-ellipse based on a cloud of points

A similar process can be applied to a model that consists of a single NACA airfoil, shown in Figure 6. The red points are the points in cloud and the blue lines are the parametric model. For the airfoil problem, the correct fitting also can be obtained in only 2 cycles. In this case, chord length is fixed at one. The design parameters chosen for the NACA airfoil generation are the maximum camber, location of maximum camber, maximum thickness, location, and the angle of attack.



(a) Initial guess

(b) 1st cycle

(c) 2nd cycle

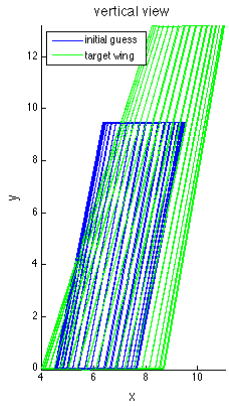
(d) 3rd cycle

Figure 6: Generation of parametric model for 2D airfoil based on a cloud of points

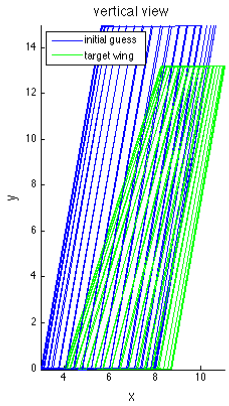
B. Single Component in Three Dimensions

A similar process can be applied in three dimensions, where it is tested on a three-dimensional isolated wing and fuselage. The 10 design parameters for the generation of a parametric wing are location in space, wing area, aspect ratio, taper ratio, twist angle, sweep angle, dihedral angle, maximum thickness, and maximum camber. The wing is generated by creating a ruled surface between airfoils generated at the root and tip. The major difference between the airfoils and wings is that there is another parametric coordinate, v_i, along the span direction.

Two problems can arise with models such as this. One occurs when the initial guess is smaller than the fitting target, and the other occurs when the initial guess is bigger than the cloud of point, as shown in Figure 7.



(a) Initial guess smaller than target wing



(b) Initial guess bigger than target wing

Figure 7: Two initializations that are troublesome

If the initial guess is smaller than the cloud of points, which is shown in Figure 7a, during the fitting iterations it is possible for one of the parametric coordinates (u or v) to increase beyond the range over which u and v are defined; when this happens, the equations that generate the body shape cannot be evaluated. To ensure that this does not happen, it is important to define the parametric coordinates to be periodic.

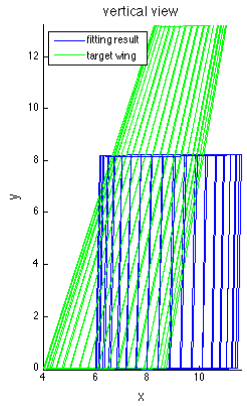
The other case occurs when the initial guess is larger than the cloud of points, as shown in Figure 7b. Here, there is a significant part of the model that is not associated with any points in the cloud. In general, the optimizer will not be able to detect and fix this since all the points in the cloud may be close to (a portion of) the configuration. To solve this, a penalty function that penalizes configurations that are too big can be added to the objective function. The penalty can be related to the surface area or volume of the configuration. When this is done, one must take care to gradually reduce the weight of the penalty so that the penalty for the final fit vanishes. At this time, a general way of doing this has not been found that does not require the use of user-specified values for the weight of the penalty term.

For the wing fitting, an alternative penalty function was defined. This penalty function is based on the idea that the points in the cloud should map to the full range of parametric coordinates. For the wing, the penalty is

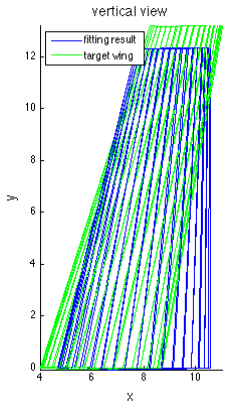
$$(1 - \max((1 - \cos u_i)/2))^2 + (1 - \max((1 - \cos v_i)/2))^2 \tag{18}$$

This form of the penalty term will reduce to zero during the optimization process and will not impact the final result. Figure 8 shows the fitting result of 3D wing on different cycles based on 1000 points. The green lines are the fitting target, which is the original wing model that was used to generate the cloud of points, and the blue lines show the optimization results. The convergence histories of these calculations are shown in Figure 9. In part (a) the RMS of the distances between the points in the cloud and the surface are shown as a function of iteration number; the abrupt rises in the RMS values occur during the reinitialization process (because the initial values of (u_i, v_i) are restricted to be one of the points in the discrete representation of the model).

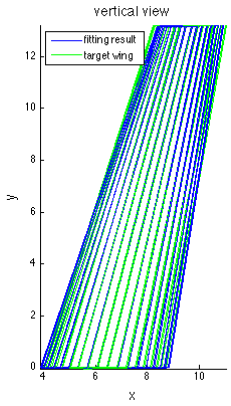
In part (b), the various design parameters, each normalized by its value that was used when the cloud of points were generated, are shown. Most of the optimized design parameters match their targets very well; the few that do not fit well (wing area, twist angle, and maximum thickness) turn out to have very little influence in this case and so their discrepancies are not significant.



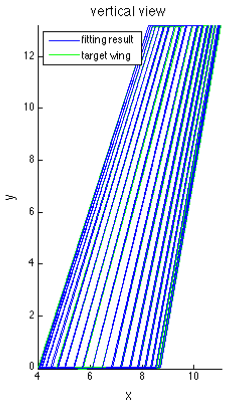
(a) Initial value of wing



(b) Fitting result after 1 cycle

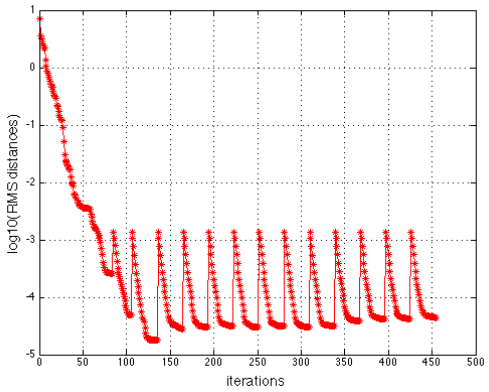


(c) Fitting result after 2 cycles

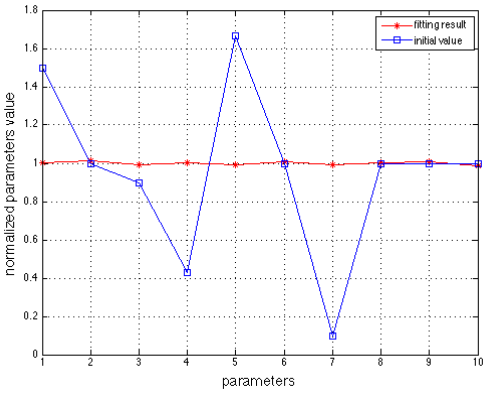


(d) Fitting result after 3 cycles

Figure 8: Progression of fitting results for 3D wing



(a) RMS of distances



(b) Normalized parameters

Figure 9: RMS distances and normalized parameters for 3D wing

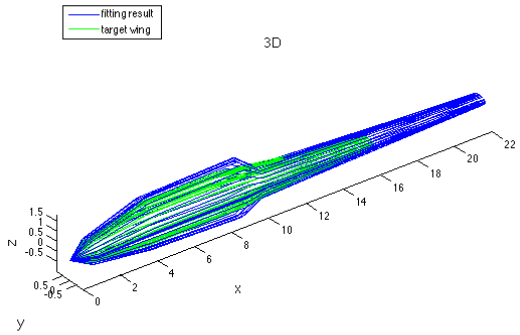
The second single-component configuration in three dimensions is a parametric fuselage, which is defined as the ruled surface between several super-elliptical cross-sections. Since there are 6 cross-sections, and each has 4 design parameters, this case contains a total of 24 design parameters, which are shown in Table 1. These are in addition to the $2m$ optimization variables (u_i, v_i) that are associated with the m points in the

cloud.

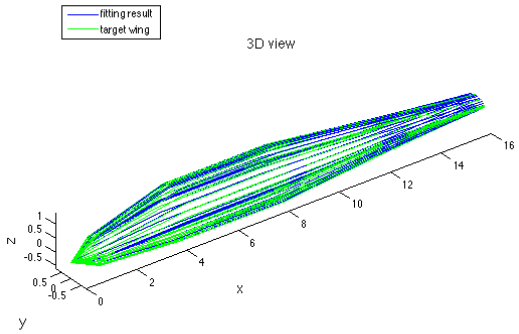
Table 1: Design Parameters of Fuselage

	section1	section2	section3	section4	section5	section6
x location	0	1	4	8	12	16
z location	0	0.1	0.4	0.4	0.3	0.2
width	0	1	1.6	1.6	1	0.8
height	0	1	2	2	1.2	0.4

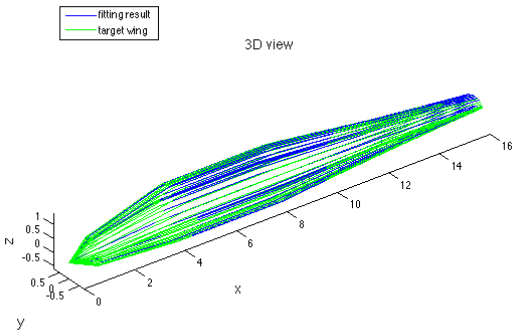
The iteration history, final fitting result, and normalized optimized parameters are shown in Figures 10-12. As can be seen from the Figure 10, the correct fitting result can be obtained after one cycle. For Figure 12, the discontinuities result from target values that are zero, for which a normalized result, is not defined.



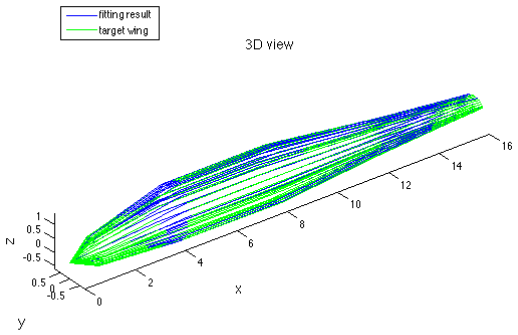
(a) Initial value of fuselage



(b) Fitting result after 1 cycle



(c) Fitting result after 2 cycles



(d) Fitting result after 3 cycles

Figure 10: Progression of fitting results for 3D fuselage

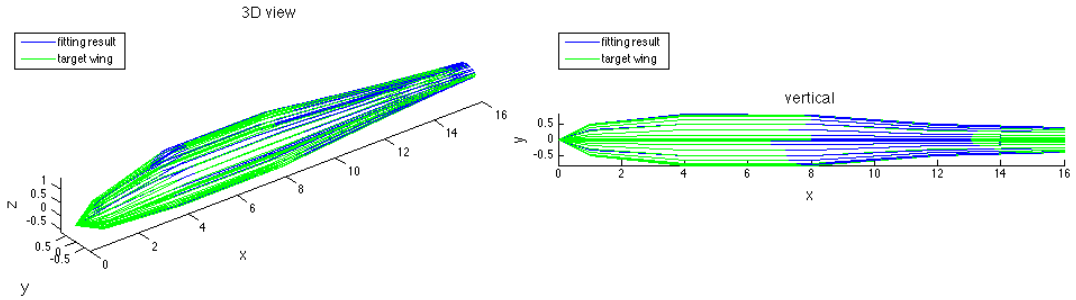
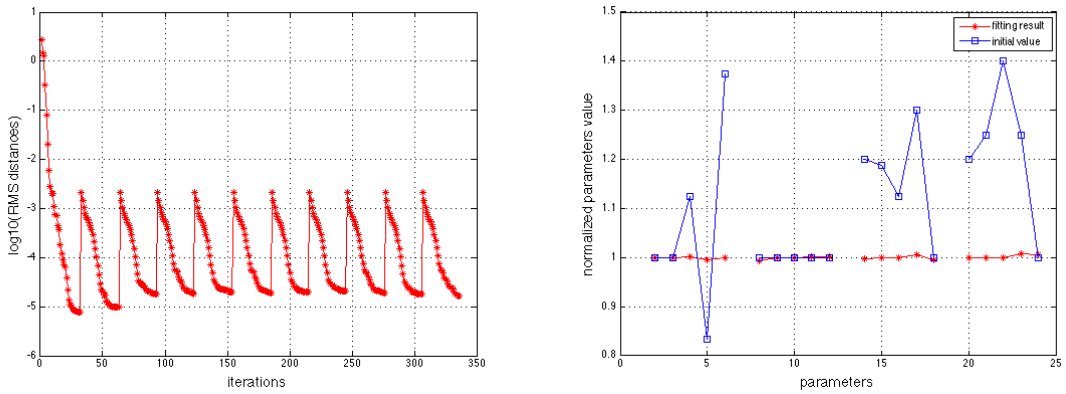


Figure 11: Final fitting result of 3D fuselage after using periodic (u, v)



(a) RMS of distances

(b) Normalized parameters

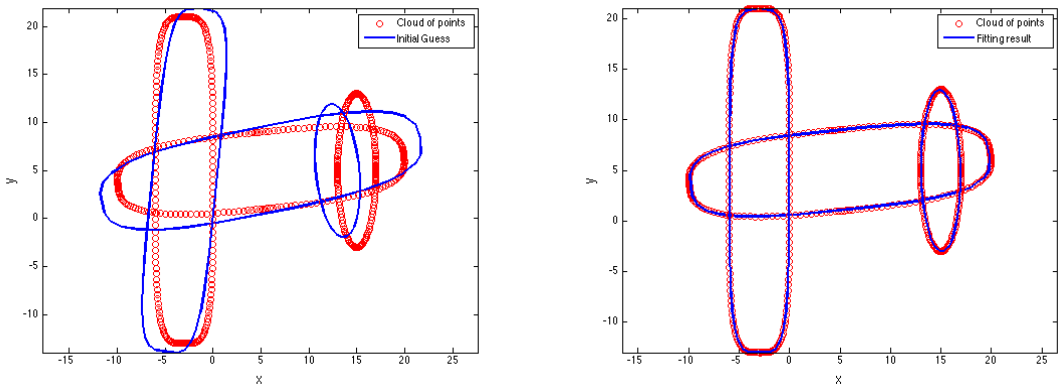
Figure 12: RMS distances and normalized parameters for 3D fuselage

IV. Multiple-Component Configuration

In the above, the fitting has been done for a single component. In this section, the generation of parametric models for multiple components based on a cloud of unclassified points is introduced. To ease the discussion, the technique will be explained in two dimensions with multiple super-ellipses; it will the be applied to a three-dimensional case of a glider.

A. Definition

The basic multiple-component problem is demonstrated by three super-ellipses. As seen in Figure 13, the blue line in (a) is the initial guess and the red points are the points in the cloud. The blue line in (b) is three parametric models which is generated from the technique. The objective here is to find a method that starts from (a) and ends at (b). This process is almost the same as the single component problem, but the multiple components problem needs one more technique to classify the different components and use the single component technique on each part after classification. Specifically, the classification need be finished at the same time as the objective function is minimized.



(a) Cloud of points and initial guess

(b) Fitting result

Figure 13: Sample classification problem for three super-ellipses

B. Classification

Recall that for the single component problem, the initialization technique found the (u_i, v_i) by finding the smallest distance from points in the cloud to the discrete points in the model. For multiple components, during the initialization process one also needs to record the identity of the component to which each point in the cloud belongs. Figure 14a shows the initial classification of the points in the cloud.

Classification correlates with the optimization process. If the initial classification is not good, the optimization will yield a bad result, which in turn will yield incorrect results in subsequent classifications. As can be seen in Figure 14a, there are many misclassified points near the intersections of the super-ellipses, which will adversely impact the optimization result. Therefore, a strategy has been adopted in which the points in the cloud that are near the intersections (as evidenced by the fact that they are equally close to two or more components) are temporarily ignored; the tolerance to determine if they are "equally close" is rather loose in early stages and is gradually tightened so that, in the end, nearly all points are classified. Note that classification is part of initialization, and that any time one needs to reinitialize, one also needs to reclassify.

C. Multiple Components in Two dimensions

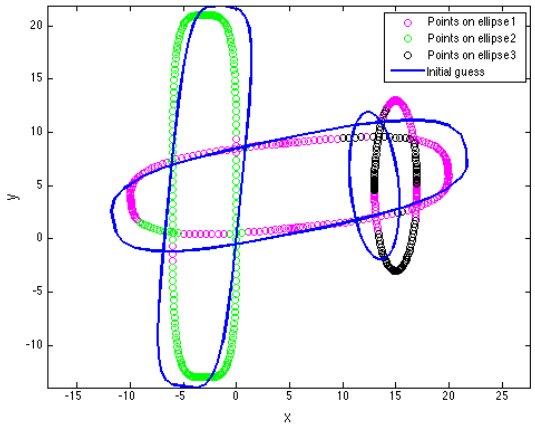
This new classification scheme has been applied to the case of three intersecting super-ellipses, as shown in Figure 14b.

Several cycles of the optimization are shown in Figure 15. The green points are the points classified to super-ellipse 1, the pink points are the points classified to super-ellipse 2, and the black points are the points classified to super-ellipse 3. The blue lines are the parametric super-ellipse model that is generated based on the cloud of points.

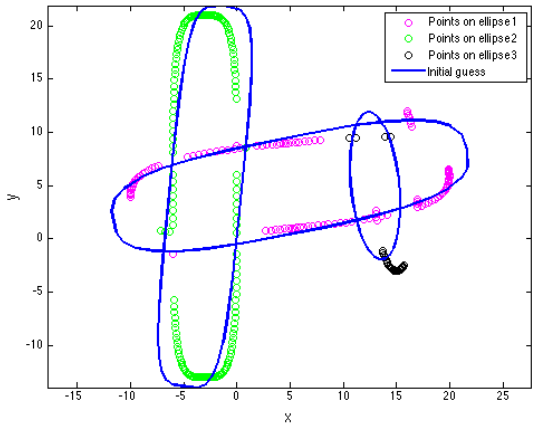
D. Multiple Components in Three Dimensions

Figures 16 to 19 show the results of fitting the glider to the cloud of points with the classification technique. Because the initial guess is used in the first classification, care must be take to create an initial guess that is somewhat close to the final configuration. (For example, an aircraft with a canard configuration should start with design parameters for a canard and not parameters for a conventional wing/tail configuration.) The initial guess and generated model for different cycles are shown in Figure 16. The green lines are the target model and blue lines are the generated glider model. The correct fitting result is obtained after 8 cycles.

Figure 17 shows the number of points associated with each component after classification cycle. Because the points in the vicinity of the intersections are be ignored at the beginning of optimization, the number of points associated with each component is much smaller than it should be. As the optimization progresses, and the shape of the model becomes closer to the correct result, the number of ignored points is automatically

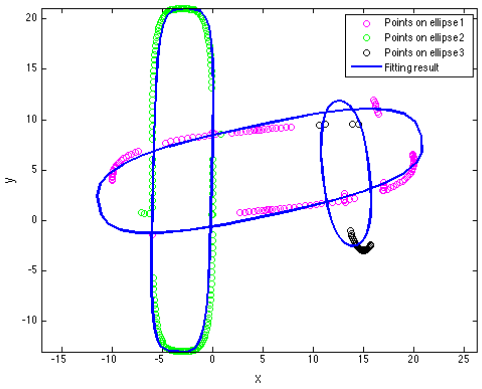


(a) Result of original classification

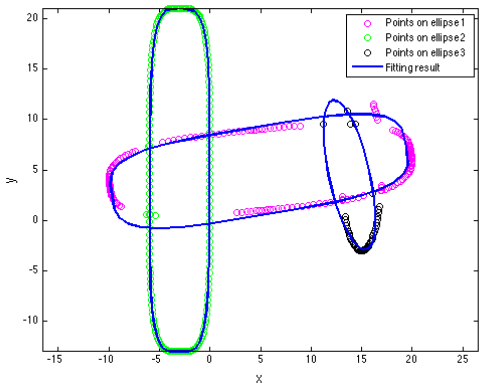


(b) Result of improved classification

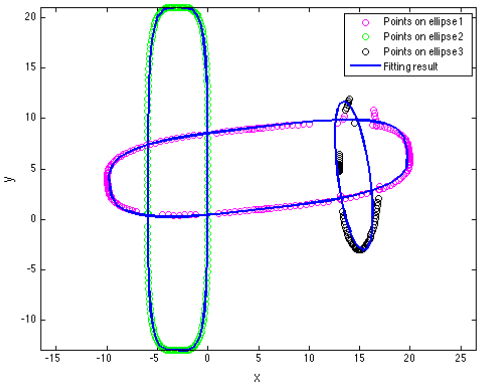
Figure 14: Initial classification result for three super-ellipses



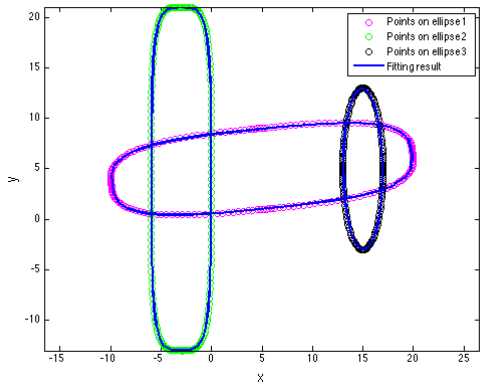
(a) Initial guess



(b) 3rd cycle

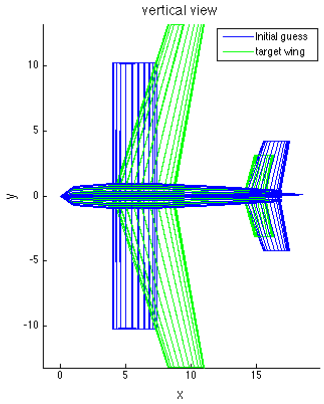


(c) 7th cycle

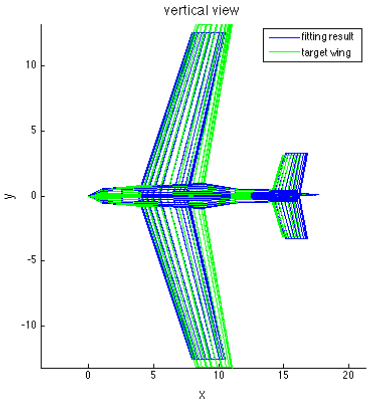


(d) 10th cycles

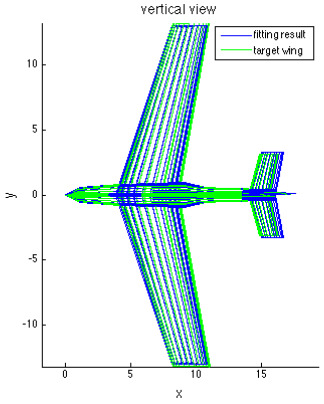
Figure 15: Progression of fitting results for three super-ellipses



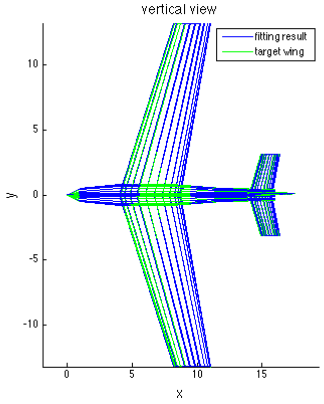
(a) Initial value



(b) Fitting result after 1st cycle



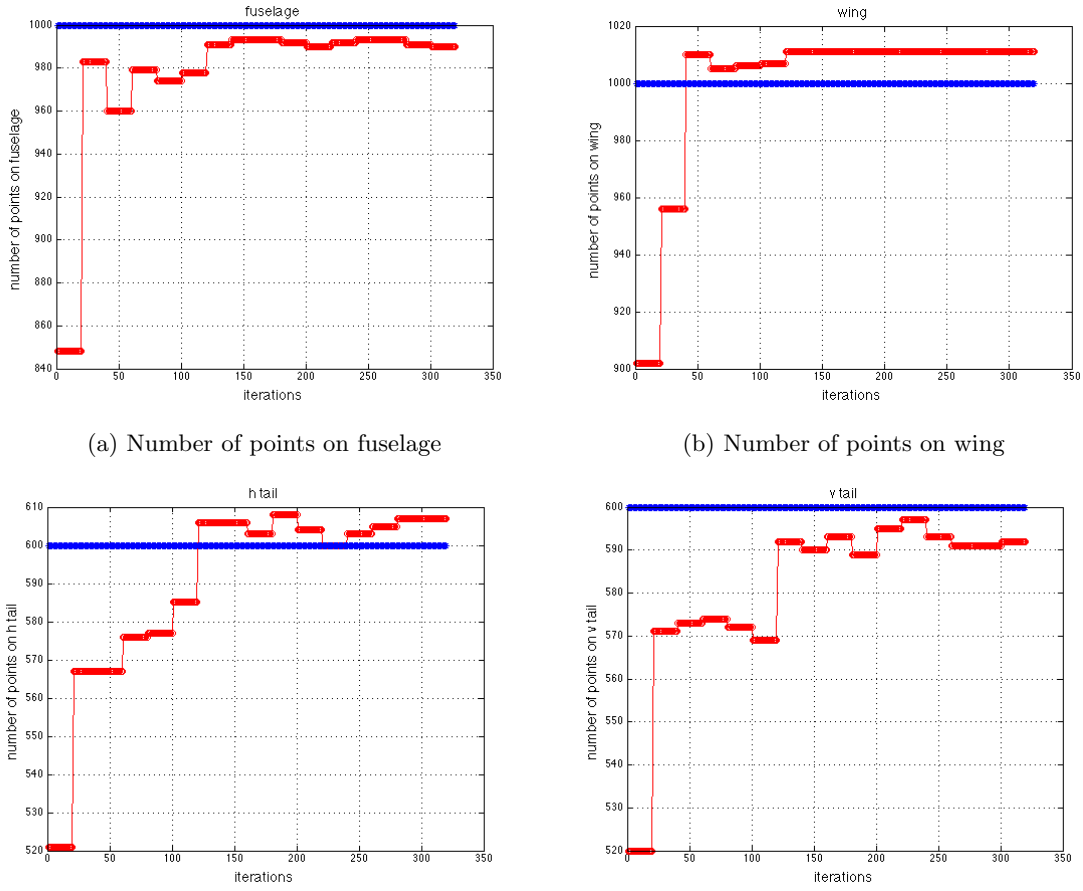
(c) Fitting result after 2nd cycle



(d) Fitting result after 8th cycle

Figure 16: Progression of fitting results for 3D glider

reduced. The final number of points associated with each component ends up being within 1% of the correct number.



(a) Number of points on fuselage

(b) Number of points on wing

(c) Number of points on horizontal tail

(d) Number of points on vertical tail

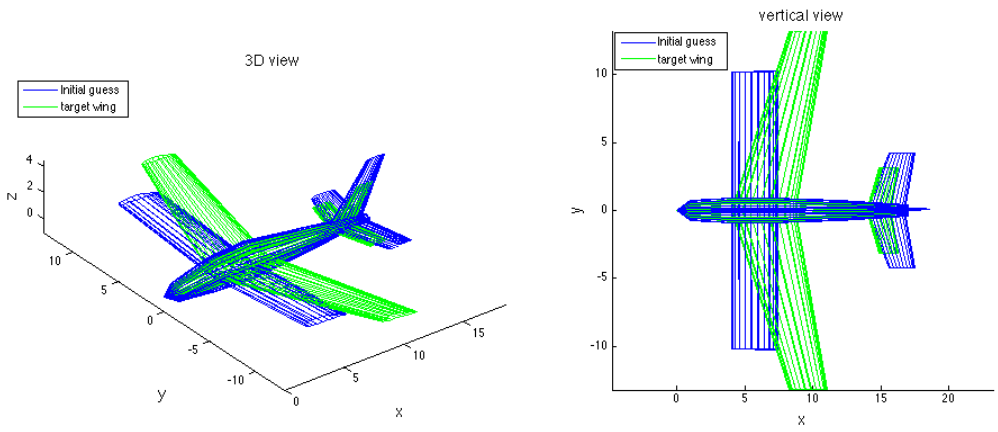
Figure 17: Number of points associated with each component for 3D glider

The final result adds the classification step to the glider, and the results are shown in Figure 18. The RMS of distance from points in cloud to the surface of the glider is shown in Figure 19a, where the system converges after 6 cycles. The normalized design parameters for the glider are shown in Figure 19b. As can be seen, the generated normalized parameters in the unclassified problem is not as good as in pre-classified problem. This is because 1% of the points are misclassified. However, the whole technique applied on the cloud of unclassified points still yields an acceptable result.

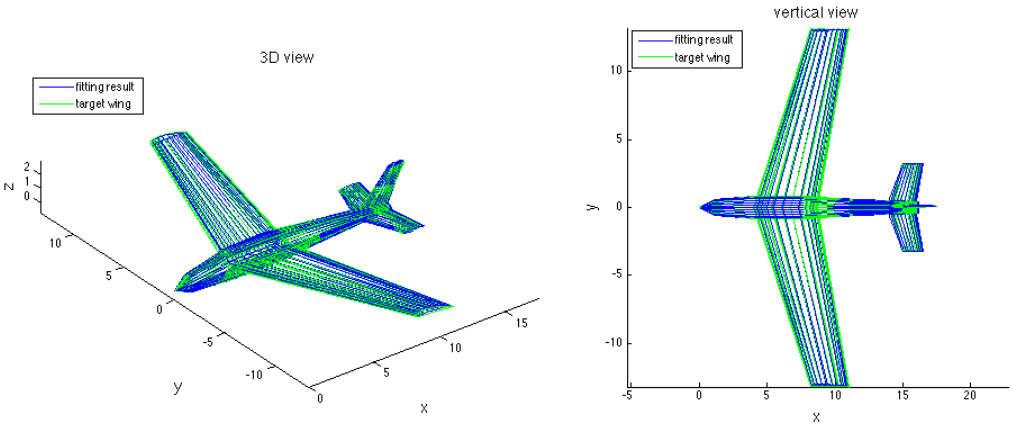
E. Effect of Improvements in LM Method

The fitting results after 10 iterations based on different updating rules are shown in Figure 20. The green lines are the target for fitting, and the blue lines are the result of the optimization process. As can be seen, using the simulated annealing idea for updating the iteration result can increasing the convergence speed. After 10 iterations, the shape of parametric model in Figure 20d is closer to the fitting target comparing with the shape in Figure 20b or in Figure 20c (which corresponds to a scheme in which trial solutions are accepted based upon a random number, as is done in simulated annealing alone). The corresponding results are shown in Figure 21.

Figure 22a shows the RMS distances between points in the cloud and generated glider. After relaxing the condition of updating the iterations' result, the RMS distance reduced more quickly. It can get the correct geometry model after 5 cycles, as opposed to 8 cycles during using original LM method. Also, the normalized parameters are all nearly one, which are shown in Figure 22b.

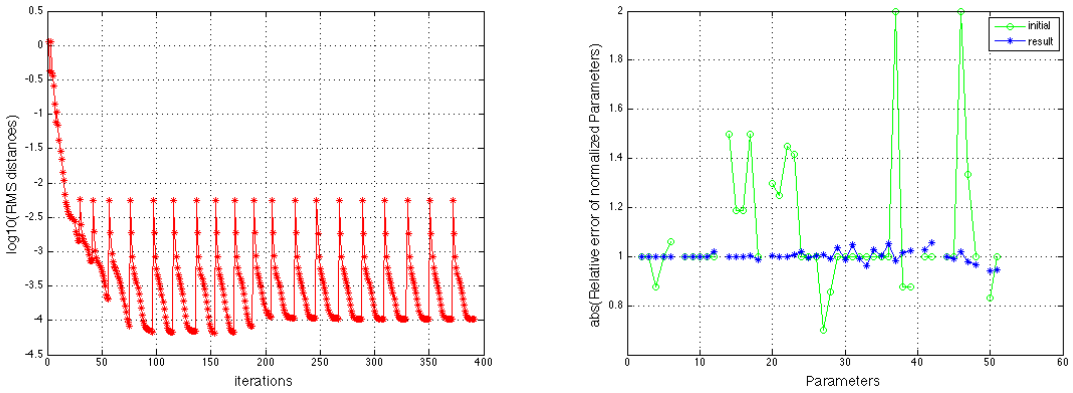


(a) Initial configuration



(b) Final configuration

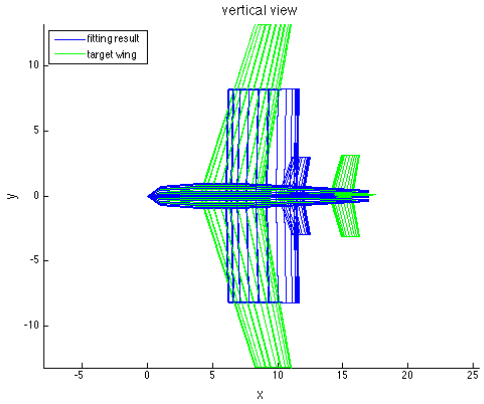
Figure 18: Final fitting result for 3D glider



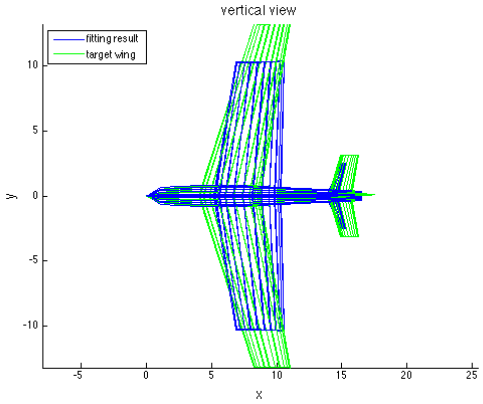
(a) RMS distances

(b) Normalized parameters

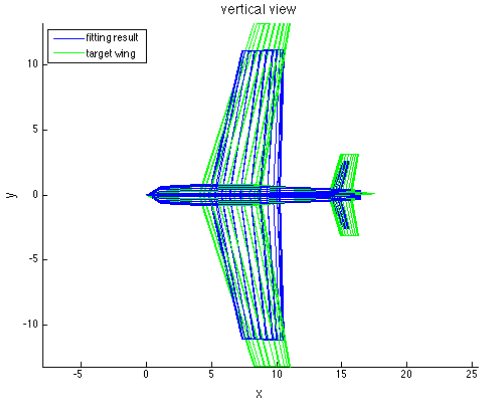
Figure 19: RMS distances and normalized parameters for 3D glider



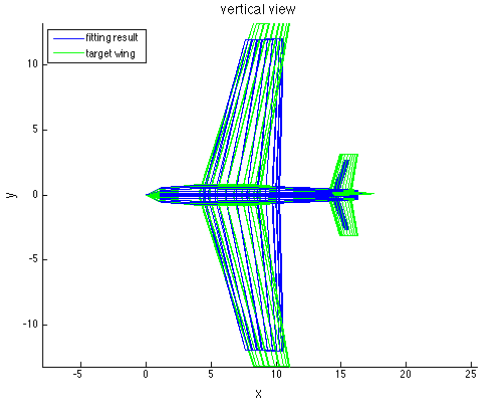
(a) Initial configuration



(b) LM method

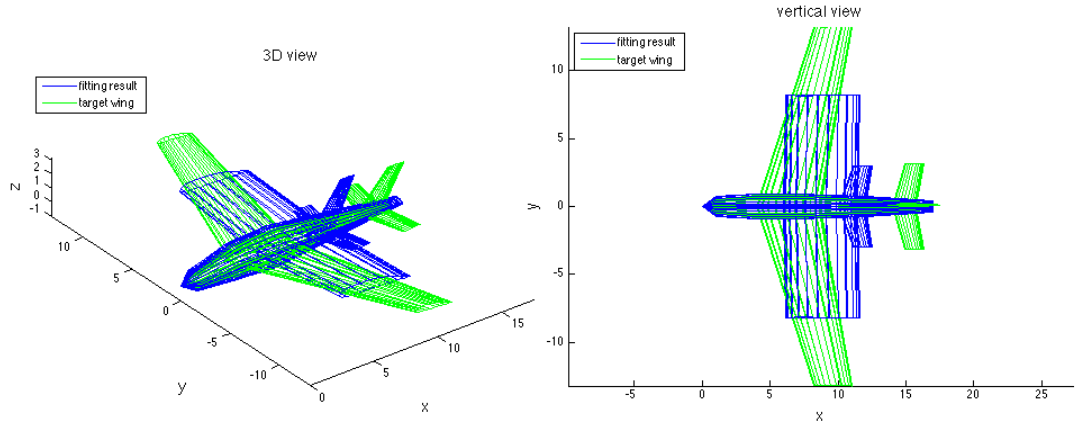


(c) LM coupling SA method

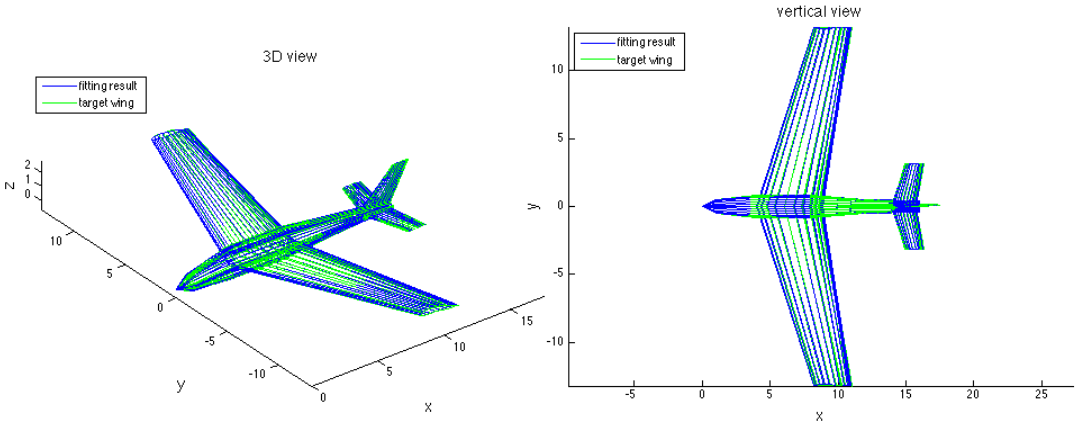


(d) LM coupling improved SA method

Figure 20: Comparison of different variants of LM method after 10 iterations

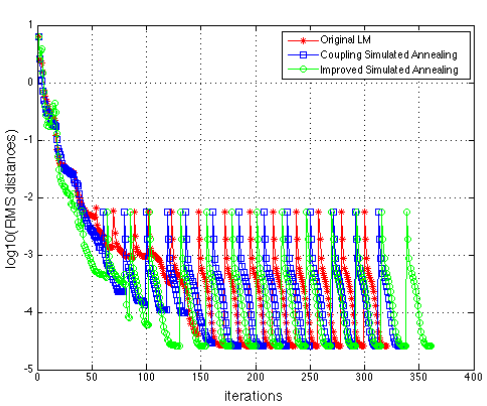


(a) Initial configuration

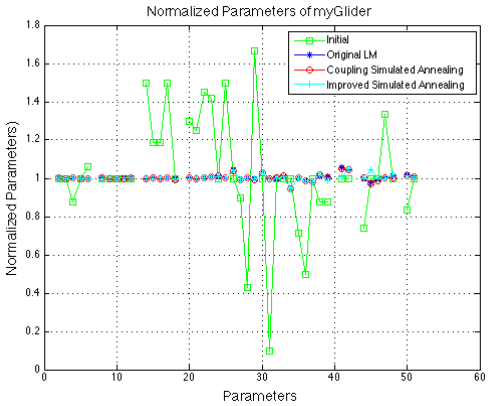


(b) Final fit configuration

Figure 21: Initial and final results for 3D glider



(a) RMS distances



(b) Normalized parameters

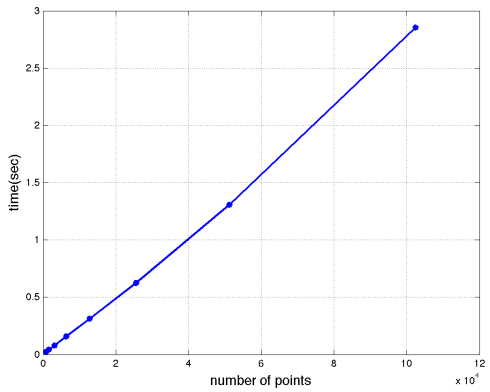
Figure 22: RMS distances and normalized parameters for 3D glider

F. Running Time

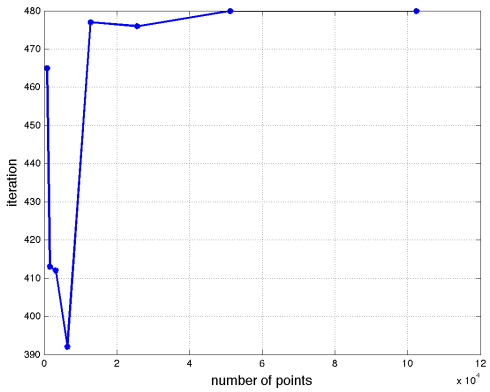
The running time for the glider problem is analyzed in this section.

In order to determine the influence of the number of points in the cloud on the running time, a series of cases was developed in which the number of points in the cloud is doubled each time. The number of points that are used to reinitialize is fixed at 14000. One can see from Figure 23a that the average running time for each iteration is increasing linearly with the number of points in cloud. This result predicts that the current scheme can be applied to a very complex problem in a reasonable time, even when the number of points in the cloud is very large.

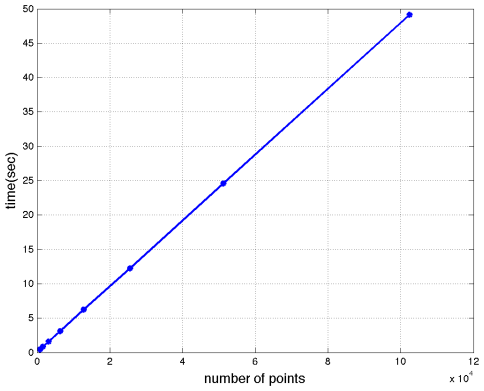
The plot of running time and iterations are shown in Figure 23. The running time increase linearly with the number of points in cloud. The fitting process can be finished in 40 minutes for 10^5 points in the cloud (in MATLAB of a MacBook Pro).



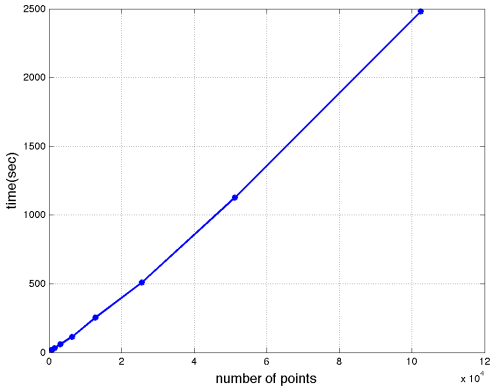
(a) Time per iteration



(b) Number of iterations required



(c) Time per reinitialization



(d) Total running time

Figure 23: Effect of problem size for 3D glider

In Figure 23c, the running time of each reinitialization is also increased linearly, which means the sum of each iteration time is linearly increased. Moreover, the average of the total iteration time is increased linearly by m, which is shown in Figure 23d. This is because the number of iterations is almost kept constant when the number of points in cloud increases.

Table 2: Running Time Analysis for 3D glider without Classification

Improved Levenberg-Marquardt with sparse Jacobain matrix								
30 maximum iterations and 16 cycles								
number of cloud of points	800	1600	3200	6400	12800	25600	51200	102400
number of points for reinitialize	14000	14000	14000	14000	14000	14000	14000	14000
average time for each iteration	0.023	0.042	0.079	0.159	0.312	0.625	1.307	2.854
number of iterations	465	413	412	392	477	476	480	480
total time	19	32	60	114	255	509	1127	2481
reinitialize time	0.469	0.840	1.60	3.14	6.25	12.2	24.6	49.1

V. Conclusion

The generation of a parametric geometry model based on a cloud of points can be solved by a gradient optimization method such as the Levenberg-Marquardt (LM) algorithm; this is used because the LM method takes advantage of the least squares property of the objective function. The LM method is modified with ideas borrowed from simulated annealing to help it avoid getting stuck in a local minimum; interestingly, this modification also improves the optimizer's rate of convergence.

For problems such as this, it is essential to take advantage of the form of the various operations performed during the optimization. Here, the sparse matrix techniques significantly improves the running speed during the optimization.

In order to improve the robustness of the technique, reinitialization (and reclassification for the multiple-component case) are used. This significantly reduces the likelihood that the optimizer will get stuck at a local minimum.

The running time for each iteration of the whole algorithm is shown to increase linearly with the number of points in cloud. Also, the number of iteration remains constant with the number of points.

Acknowledgments

This work was performed as part of the CAPS project, which was funded under AFRL Contract FA8050-14-C-2472: "CAPS: Computational Aircraft Prototype Syntheses"; Ed Alyanak is the Technical Monitor.

References

[1]Bo, P., Ling, R., and Wang, W., "A revisit to fitting parametric surfaces to point clouds," *Computers and Graphics*, Vol. 36, No. 5, 2012, pp. 534 – 540, Shape Modeling International (SMI) Conference 2012.

[2]Yoshihara, H., Yoshii, T., Shibutani, T., and Maekawa, T., "Topologically robust B-spline surface reconstruction from point clouds using level set methods and iterative geometric fitting algorithms," *Computer Aided Geometric Design*, Vol. 29, No. 7, 2012, pp. 422 – 434, Geometric Modeling and Processing 2012.

[3]Kang, H., Chen, F., Li, Y., Deng, J., and Yang, Z., "Knot calculation for spline fitting via sparse optimization," *Computer-Aided Design*, Vol. 58, No. 0, 2015, pp. 179 – 188, Solid and Physical Modeling 2014.

[4]Galvez, A. and Iglesias, A., "Particle swarm optimization for non-uniform rational B-spline surface reconstruction from clouds of 3D data points," *Information Sciences*, Vol. 192, No. 0, 2012, pp. 174 – 192, Swarm Intelligence and Its Applications.

[5]Galvez, A. and Iglesias, A., "A new iterative mutually coupled hybrid GAPSO approach for curve fitting in manufacturing," *Applied Soft Computing*, Vol. 13, No. 3, 2013, pp. 1491 – 1504, Hybrid evolutionary systems for manufacturing processes.

[6]Kineri, Y., Wang, M., Lin, H., and Maekawa, T., "B-spline surface fitting by iterative geometric interpolation/approximation algorithms," *Computer-Aided Design*, Vol. 44, No. 7, 2012, pp. 697 – 708.

[7]Motavalli, S., "Review of reverse engineering approaches," *Computers and Industrial Engineering*, Vol. 35, No. 12, 1998, pp. 25 – 28.

[8]Zhao, X., Zhang, C., Xu, L., Yang, B., and Feng, Z., "IGA-based point cloud fitting using B-spline surfaces for reverse engineering," *Information Sciences*, Vol. 245, No. 0, 2013, pp. 276 – 289, Statistics with Imperfect Data.

[9]Wang, J. and Yu, Z., "Quadratic curve and surface fitting via squared distance minimization," *Computers and Graphics*, Vol. 35, No. 6, 2011, pp. 1035 – 1050.

[10]Pottmann, H., Leopoldseder, S., Hofer, M., Steiner, T., and Wang, W., "Industrial geometry: recent advances and applications in CAD," *Computer-Aided Design*, Vol. 37, No. 7, 2005, pp. 751 – 766.

[11]Beniere, R., Subsol, G., Gesquiere, G., Breton, F. L., and Puech, W., "A comprehensive process of reverse engineering from 3D meshes to CAD models," *Computer-Aided Design*, Vol. 45, No. 11, 2013, pp. 1382 – 1393.

[12]Min, Z., "A New Approach of Composite Surface Reconstruction Based on Reverse Engineering," *Procedia Engineering*, Vol. 23, No. 0, 2011, pp. 594 – 599, {PEEA} 2011.

[13]Varady, T., Martin, R. R., and Cox, J., "Reverse engineering of geometric models–an introduction," *Computer-Aided Design*, Vol. 29, No. 4, 1997, pp. 255 – 268, Reverse Engineering of Geometric Models.

[14]Stamati, V., Antonopoulos, G., Azariadis, P., and Fudos, I., "A parametric feature-based approach to reconstructing traditional filigree jewelry," *Computer-Aided Design*, Vol. 43, No. 12, 2011, pp. 1814 – 1828.

[15]Peethambaran, J. and Muthuganapathy, R., "Reconstruction of water-tight surfaces through Delaunay sculpting," *Computer-Aided Design*, Vol. 58, No. 0, 2015, pp. 62 – 72, Solid and Physical Modeling 2014.

[16]McDonald, R. A., "Interactive Reconstruction of 3D Models in the OpenVSP Parametric Geometry Tool," *53rd AIAA Aerospace Sciences Meeting*, 2015.

[17]Marquardt, D., "An algorithm for least squares estimation of nonlinear parameters," *SIAM Journal*, Vol. 11, 1963, pp. 431–441.

[18]Dannenhoffer, J. and Haimes, R., "Design Sensitivity Calculations Directly on CAD-Based Geometry," Tech. Rep. AIAA-2015-1370, 53rd AIAA Aerospace Sciences Meeting, 2015.