

Engineering Sketch Pad (ESP)



Advanced User Training Session 01 Fitting STL Models with PLUGS

John F. Dannenhoffer, III

john@geocentrictech.com

Geocentric Technologies LLC

updated for v1.29+

- **Plugs** is a tool for finding the Design Parameter values such that a Body matches a cloud of points
- Inputs:
 - a single parameterized Body
 - a cloud of (unclassified) points
- Outputs:
 - Design Parameter values
- Executed via **Tool** → **Plugs**
- Algorithm is described in Jia, P, and Dannenhoffer, J.F., “Generation of Parametric Aircraft Models from a Cloud of Points”, AIAA-2016-1926, presented at AIAA SciTech 2016, January 2016.

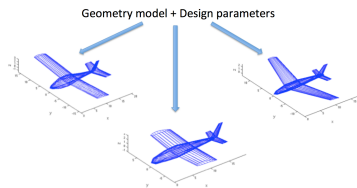
- Introduction
- Optimization technique
- Single component configuration
- Multiple component configuration
- Running time
- Implementation notes
- Homework exercises

Introduction

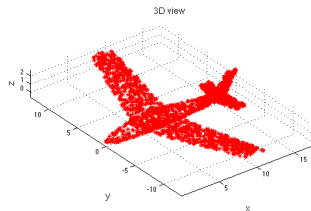
Problem statement

Glider model is defined in terms of 52 design parameters

- wing area
- aspect ratio
- . . .



Which set of parameters “best fits” a cloud of points?



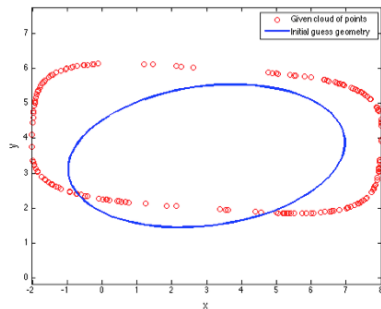
Introduction

Overall approach

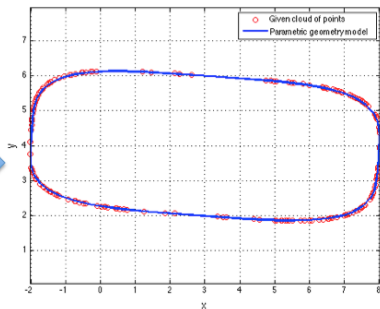
- Given:
 - Cloud of points $(x_{p(i)}, y_{p(i)}, z_{p(i)})$ for $(i = 1, \dots, m)$
 - Parametric model with design parameters d_j for $(j = 1, \dots, n)$
- Classify:
 - Pick entity associated with each point in the cloud
- Initialize:
 - Pick each parametric coordinate u_i that is closest to a discrete point in its associated entity
- Optimize:
 - Find $\vec{d}$ and $\vec{u}$ that minimizes the sum-squared distances between the points in the cloud and the configuration
- Repeat back to classification until converged

Introduction

2D super-ellipse sample problem



Red points: cloud of points
Blue line: initial guess



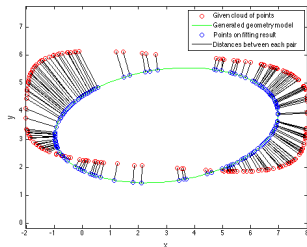
Red points: cloud of points
Blue line: final model

Optimization Technique

Objective function

$$S = \sum_{i=1}^m \underbrace{(x_{p(i)} - f_x(\vec{d}, u_i))^2}_{q_i} + \underbrace{(y_{p(i)} - f_y(\vec{d}, u_i))^2}_{q_{i+m}}$$

- $(x_{p(i)}, y_{p(i)})$ for $i = 1, \dots, m$
 - points in cloud
- $(f_x(\vec{d}, u_i), f_y(\vec{d}, u_i))$
 - functions that generate points on configuration with design variables $\vec{d}$ at parametric coordinate u_i



Optimization Technique

Levenberg-Marquardt algorithm — 1

- The Newton method to find $\vec{\beta} \equiv \{\vec{d}, \vec{u}\}$ for which derivative of S vanishes:

$$\vec{\delta} \equiv \vec{\beta}^{(s+1)} - \vec{\beta}^{(s)} = -[H(\vec{\beta}^{(s)})]^{-1} \cdot \vec{g}(\vec{\beta}^{(s)}) \quad \text{for } s = 0, 1, \dots$$

where $\vec{g}$ is gradient of S and H is Hessian of S

- Levenberg and Marquardt approximated

$$\begin{aligned} \vec{g} &= 2 \cdot J^T \cdot \vec{q} \\ H &= 2 \cdot J^T \cdot J \end{aligned}$$

where J is the (very sparse) Jacobian matrix of $\vec{q}$

- Newton's method thus becomes

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

Optimization Technique

Levenberg-Marquardt algorithm — 2

- Newton's method only works when close to minimum
- If far from minimum, introduce the gradient descent method (GDM)

$$\vec{\delta} \equiv \vec{\beta}^{(s+1)} - \vec{\beta}^{(s)} = -\vec{g}(\vec{\beta}^{(s)})$$

- Levenberg and Marquardt added a damping parameter, λ , to combine two methods:

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

- If λ is small
 - LM behaves like Newton method
- Otherwise
 - LM behaves like GDM

Optimization Technique

Levenberg-Marquardt algorithm — 3

- For each step, s
 - compute $\vec{\beta}^{(temp)} = \vec{\beta}^{(s)} + \vec{\delta}$ and then $\vec{q}^{(temp)}$ based upon $\vec{\beta}^{(temp)}$
 - if $\|\vec{q}^T \cdot \vec{q}\|^{(temp)} < \|\vec{q}^T \cdot \vec{q}\|^{(s)}$
 - update $\vec{\beta}^{(s+1)} = \vec{\beta}^{(temp)}$
 - halve λ (making it more Newton-like)
 - else
 - do not update $\vec{\beta}$
 - double λ (making it more gradient-descent-like)
 - if $\|\vec{\delta}\| < \epsilon$, stop
- The original Levenberg-Marquardt algorithm has two problems:
 - Can get stuck at local minimum because only “better” results are accepted
 - Convergence slow because some iterations do not update $\vec{\beta}$.

Optimization Technique

Simulated annealing algorithm — 4

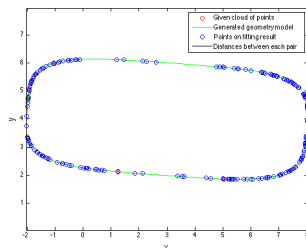
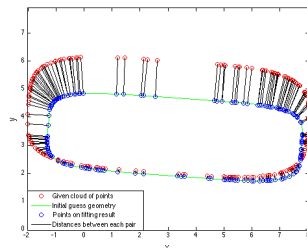
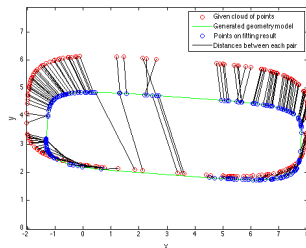
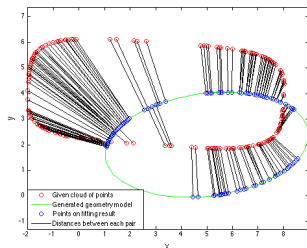
- Borrow concept from simulated annealing method
 - Candidate solutions are randomly generated
 - Accepted if the new objective is “not too much worse”
- Idea can be applied to LM

$$||\vec{q}^T \cdot \vec{q}||^{(temp)} < \underbrace{e^{||\vec{\delta}||}}_{\text{not too much worse}} ||\vec{q}^T \cdot \vec{q}||^{(s)}$$

- In early iterations, $||\vec{\delta}||$ is large, so some “uphill” steps can be taken
- As $||\vec{\delta}|| \rightarrow 0$, only “downhill” steps are taken
- New method is called Improved Levenberg-Marquardt (ILM)

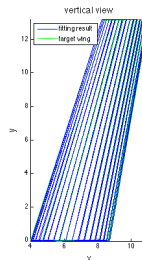
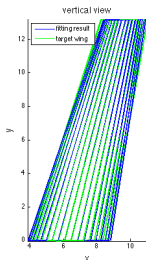
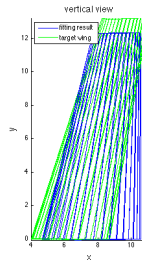
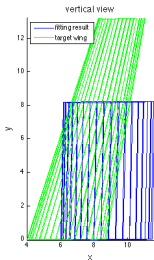
Single Component Configuration

After initialization to nearest discrete point on super-ellipse
After first optimization
Immediately after re-initialization
Final optimal result



Single Component Configuration

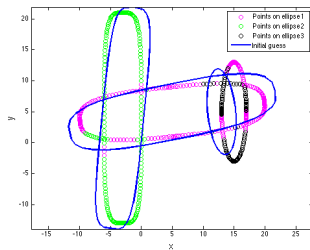
Initial guess for 3D wing After 1 cycle for 3D wing After 2 cycles for 3D wing
After 3 cycles for 3D wing



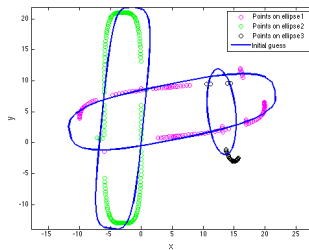
Multiple-Component Configuration

Classification for three super-ellipses in 2D

During initialization, one also needs to record to which entity each point in the cloud belongs

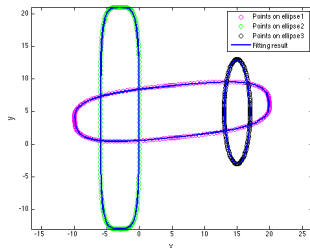
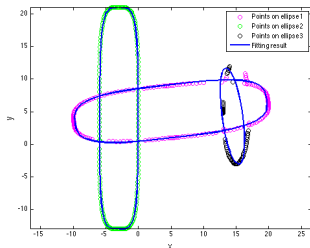
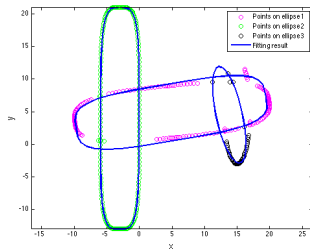
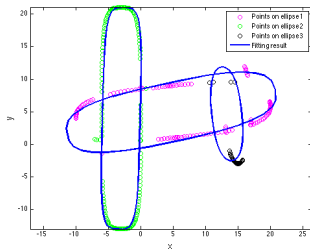


To avoid misclassifications, points near 2 or more entities are not classified (in early passes)



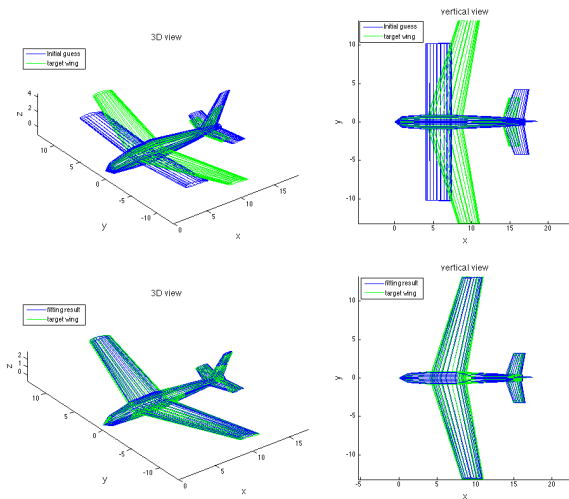
Multiple-Component Configuration

After 1 cycle for 3 super-ellipses After 3 cycles for 3 super-ellipses After 7 cycles for 3 super-ellipses After 10 cycles for 3 super-ellipses



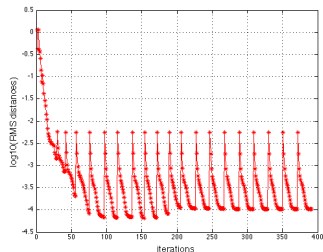
Multiple-Component Configuration

3D glider: initial and final configurations

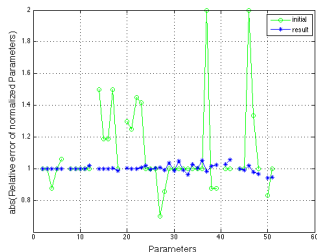


Summary of results for 3D glider with classification

20 cycles of up to 30 iterations each



RMS of distances

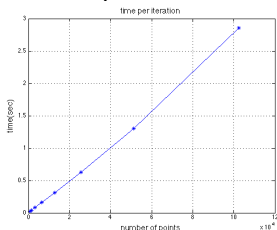


Normalized design parameters $\vec{d}$

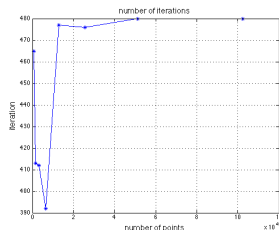
Running Time vs. Num of Points in Cloud

3D glider results with 16 cycles of up to 30 iterations each

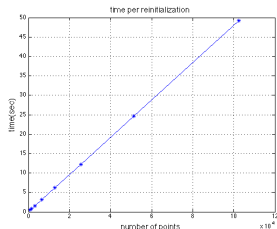
Time per iteration



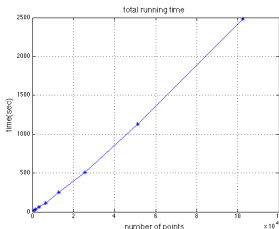
Number of iterations



Time per initialization



Total running time



- Phase 1
 - Take up to 50 Levenberg-Marquardt optimization steps to vary the Design Parameters to match the bounding box of the point cloud
 - Unclassify all points in the cloud
- Phase 2
 - In a sequence of passes
 - classify each point in the cloud by determining the most likely Face to associated it with; if there is no obvious Face, leave the point unclassified
 - if all points are classified and the point classifications are the same as in the previous pass, exit
 - perform up to 50 steps of the Levenberg-Marquardt optimizer

- While executing a Phase, pressing the yellow button will stop **Plugs** at the end of the current pass
- If **Plugs**→**Quit** is selected, the DESPMTRs are returned to their values before **Plugs** was started
- If **Plugs**→**Save** is selected, the DESPMTRs are updated and a file named `plugs.despmtrs` is created

- Run Plugs on `exercises/session01/fuselage` using `exercises/session01/fuselage.cloud`
- Before leaving Plugs, look at the results, especially near the nose and tail. Since you suspect that you need more cross-sections, **Quit** out of Plugs
- Change `nsect` to 8 and re-run Plugs
- Since the results are better, **Save** the results that Plugs gave you.