INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600



Downloaded from SHAREOK. Persistent Handle URL: https://hdl.handle.net/11244/500

UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

AN INTELLIGENT SEARCH-BASED METHODOLOGY FOR SELECTION OF SAMPLE POINTS FOR FORM ERROR ESTIMATION

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

MOHAMMAD AFFAN BADAR
Norman, Oklahoma
2002

UMI Number: 3059900



UMI Microform 3059900
Copyright 2002 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

© Copyright by Mohammad Affan Badar 2002
All Rights Reserved.

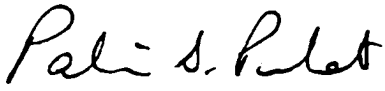
AN INTELLIGENT SEARCH-BASED METHODOLOGY FOR SELECTION OF SAMPLE POINTS FOR FORM ERROR ESTIMATION

A Dissertation APPROVED FOR THE
SCHOOL OF INDUSTRIAL ENGINEERING

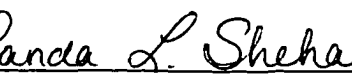
BY



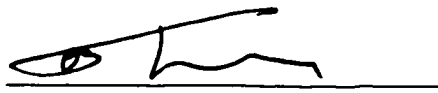
Dr. Shivakumar Raman, Chair



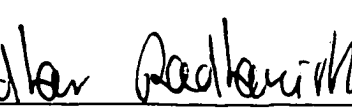
Dr. Pakize Simin Pulat, Co-Chair



Dr. Randa L. Shehab



Dr. Theodore B. Trafalis



Dr. Sridhar Radhakrishnan

ACKNOWLEDGEMENTS

First, I am very grateful to the Almighty for His Guidance and Support throughout my life. Then, I would like to express my sincere gratitude to my late parents. In fact, everything I have accomplished is because of them. I am also thankful to my sister and brothers and their families for their cooperation in general and for taking care of our parents during their last days while I was away in particular. Further, I would like to thank my wife for her understanding and patience, especially after the birth of our daughter, Sidrah. I am grateful to my in-laws and the family as well for their thoughts.

Special thanks are due to my advisors, Dr. Shivakumar Raman and Dr. P. Simin Pulat for their continuous guidance and assistance during these past years as a PhD student. I am grateful to my committee member, Dr. Randa L. Shehab for her guidance, particularly in the experimental design and analysis part of the dissertation. I would also like to express my appreciation to my committee members, Dr. Theodore B. Trafalis and Dr. Sridhar Radhakrishnan for their valuable suggestions leading to the completion of this work. I am thankful to Dr. Thomas L. Landers as well for his advice and help. Further gratitude is extended to the entire faculty, staff, and colleagues at the School of Industrial Engineering, especially to Allison Richardson, Jean Shingledecker, and Madan Mathevan for their support. Furthermore, I am thankful to the entire Graduate College staff, particularly to Dean Amelia Adams and C. Rene Jenkins for their help.

I would also like to thank all of my relatives and friends for their encouragement.

Thank you.

TABLE OF CONTENTS

LIST OF TABLES vii
LIST OF FIGURES viii
NOMENCLATURE ix
ABSTRACT x

CHAPTER
1. **INTRODUCTION** 1
2. **LITERATURE REVIEW** 5
   2.1 Coordinate Measuring Machine 6
   2.2 Sample Design 7
      2.2.1 Sample Size 7
      2.2.2 Sample Locations 11
   2.3 Surface Profiles in Manufacturing 15
   2.4 Form Error Estimation Methods 21
      2.4.1 Minimum Zone Method 22
      2.4.2 Least Squares Method 27
3. **OPTIMIZATION SEARCH METHODS** 30
   3.1 Optimization of Functions of One Variable 30
   3.2 Pattern Search Methods 33
      3.2.1 Coordinate Search 35
      3.2.2 Hooke-Jeeves Pattern Search 37
      3.2.3 Tabu Search 38
      3.2.4 Hybrid Search 43
4. **OBJECTIVE AND OVERVIEW OF METHODOLOGY** 44
   4.1 Objective 44
   4.2 Overview of Methodology 44
5. **MANUFACTURING SURFACE PATTERN** 48
   5.1 Pattern for Flatness 48
      5.1.1 End Milling 48
      5.1.2 Face Milling 59
   5.2 Pattern for straightness 69
      5.2.1 End Milling 69
      5.2.2 Face Milling 70
   5.3 Population Data 71
6. **ALGORITHM DEVELOPMENT** 72
   6.1 Straightness Using Region Elimination Search 72
   6.2 Flatness 76
      6.2.1 Tabu Search 77
      6.2.2 Hybrid Search 85
7. **EXPERIMENTAL DESIGN** 88
   7.1 Straightness 88
   7.2 Flatness 92

8. RESULTS AND DISCUSSIONS 96
8.1 Straightness 96
8.2 Flatness 97
8.3 Comparison with Previous Work 100
9. CONCLUSIONS AND RECOMMENDATIONS 104
REFERENCES 108

APPENDIX
A. POPULATION DATA 115
A.1 Straightness 115
A.1.1 End Milling 115
Table A.1.1.1: Plate #4 115
Table A.1.1.2: Plate #12 116
Table A.1.1.3: Plate #9 117
Table A.1.1.4: Plate #2 118
A.1.2 Face Milling 119
Table A.1.2.1: Plate #3 119
Table A.1.2.2: Plate #1 119
Table A.1.2.3: Plate #11 120
Table A.1.2.4: Plate #7 120
A.2 Flatness 121
A.2.1 End Milling 121
Table A.2.1.1: Plate #10 121
Table A.2.1.2: Plate #5 126
Table A.2.1.3: Plate #7 130
Table A.2.1.4: Plate #4 135
A.2.2 Face Milling 139
Table A.2.2.1: Plate #2 139
Table A.2.2.2: Plate #9 142
Table A.2.2.3: Plate #11 145
Table A.2.2.4: Plate #5 148
B. JAVA CODES 151
B.1 Least Squares Line 151
B.2 Least Squares Plane 153
B.3 Region Elimination Search 155
B.4 Tabu Search 160
B.5 Hybrid Search 173
C. SAS FILES 197
C.1 Straightness 197
C.2 Flatness 203

LIST OF TABLES

Table

1. Observed response - number of sampled points (absolute percentage error) for straightness estimation 90
2. ANOVA for the three-factor nested-factorial model for straightness 91
3. Observed response - number of sampled points (absolute percentage error) for flatness estimation 94
4. ANOVA for the four-factor nested-factorial model for flatness 95

LIST OF FIGURES

Figure

1. Sample points for a line feature, tolerance = $|e_{max(+)}| + |e_{max(-)}|$ 3
2. The pattern for CS in $\mathbf{R}^2$ with a step length Δ_k 36
3. All possible subsets of the steps for CS in $\mathbf{R}^2$ 36
4. The pattern step in $\mathbf{R}^2$, given $x_k \neq x_{k-1}$, $k > 0$ 37
5. Cutter and workpiece deflection in end milling (source: [48]) 49
6. Surface error in $y-z$ plane in end milling (source: [48]) 53
7. Aluminum workpiece for end milling 54
8. Surface measurement with CMM (source: [35]) 55
9. Flatness error profile for end-milled plate #10 57
10. Flatness error profile for end-milled plate #5 57
11. Flatness error profile for end-milled plate #7 58
12. Flatness error profile for end-milled plate #4 58
13. Mutli-tooth face milling schematic showing the instantaneous forces exerted by each tooth on the workpiece (source: [35]) 60
14. Schematic of spindle axis tilt & effect of tooth's axial position (source: [35]) 63
15. Schematic of workpiece geometry used in [35] 64
16. Surface error in face milling (source: [35]) 65
17. Cast iron workpiece for face milling 66
18. Flatness error profile for face-milled plate #2 67
19. Flatness error profile for face-milled plate #9 68
20. Flatness error profile for face-milled plate #11 68
21. Flatness error profile for face-milled plate #5 68
22. Straightness error profile for end-milled plates 69
23. Straightness error profile for face-milled plates 70
24. Flowchart of the region elimination algorithm for straightness estimation ... 73
25. Neighborhood points of a starting point in tabu search 78
26. Tabu search algorithm flowchart for flatness estimation 80
27. Hybrid search algorithm flowchart for flatness estimation 86
28. Effect of manufacturing process and step size in straightness estimation 97
29. Results of the flatness estimation using tabu search 99
30. Results of the flatness estimation using hybrid search 99
31. Manufacturing process - algorithm and strategy - algorithm plot 100

NOMENCLATURE

line: $z = z_0 + l_0 x$

plane: $z = z_0 + l_0 x + m_0 y$

α_{hx} : helix angle

$a_{p_1 p_2}$: flexibility influence coefficient at point p_1 due to a unit force at point p_2

CFY: y force center

dY : y cutter deflection

e_i, $\hat{e}_i$: linear and normal deviation of ith point from assessment feature

$e_{max(+)}$, $e_{max(-)}$: maximum deviation in +ve and –ve direction from assessment feature

F_T, F_R : elemental tangential and radial force

F_Y : y cutting force

h_t : form tolerance

K_T, K_R : empirical constants

l_0, m_0 : slopes

N : total number of data points

n : point sequence number used in the population data tables in Appendix A

R : radius

θ : angular position

x_i, y_i, z_i : cartesian coordinates of ith data point

z_0 : intercept

ABSTRACT

Efficient part feature verification through CMM requires prudent sampling of data points. This dissertation presents an adaptive sampling procedure, which uses manufacturing error patterns and optimization search methods for reducing sample size, while maintaining high accuracy. The methodology is demonstrated with straightness and flatness evaluation.

Two manufacturing processes, end and face milling are used to produce plates. Respective surface errors are quantified and previous models are validated. Sampling begins with a necessary number of initial points guided by the geometry and error profiles of the object surface. The least squares method is applied to compute a tolerance zone. Next points are sampled based on search methods with suitable intensification and diversification, looking for improvements in the zone. The final zone value is compared with that obtained for a population sample in terms of the absolute % error. For straightness estimation, region-elimination search is used. For flatness determination, tabu search and a hybrid search are employed and their performance is compared. The hybrid search developed is a combination of coordinate search, Hooke-Jeeves search and tabu search. Experiments are conducted to investigate the effect of different factors on the sample size and % error.

Comparison with other sampling methods reveals that the present approach is more efficient and reliable. The research is expected to lead to improved solutions to inspection problems faced by industries.

CHAPTER 1

INTRODUCTION

Dimensional and geometric tolerances are assigned to selected features of engineering components to satisfy certain functional or assembly requirements [81, 95]. Dimensional tolerances are defined as the permissible variation in the dimensions such as height, width, diameter, angles, etc. of a part [43]. Geometric constraints are expressed in terms of ANSI Y14.5M-1994 [3]. In these standards, allowable variation of individual and related features is based on the 'envelope principle'; that is, the entire surface of a part feature of interest must lie within two minimum separating envelopes of ideal shape. The minimum range allowed for any shape is called form tolerance/error [40]. As dimensional tolerances are easier to compute, this work is focused toward form errors.

Verification of manufactured parts is done through manual checking against standard gages or by appropriate measurements using coordinate measuring machines (CMMs). The latter have become more popular because of the flexibility, accuracy, and ease of automation. It is extremely important to estimate form tolerances correctly in order to reduce the chance of accepting bad parts or rejecting good ones. Nevertheless, there are some problems associated with probe-type CMMs. They measure only a sample of discrete points on surfaces for individual and/or related feature verification of parts whereas tolerance standards require knowledge of the entire surface [75]. Then a zone fitting technique is used to estimate the tolerance zone. Therefore, the estimation accuracy depends on both the sampling plan and the estimation method.

Two most common methods to evaluate form tolerances are the least squares (LS) and the minimum zone (MZ) methods [22]. The LS method is simpler, but its estimates may not be the minimum. The MZ method has a kind of similarity with the envelope principle mentioned in the tolerance-standard definition, however, it underestimates the deviations since it encloses only a set of sample points. The simplex search or any other suitable search technique can be used to obtain the MZ. Both methods give seriously biased estimates of the part form tolerance when the sample size is small [21]. Thus, the larger the sample size is, the better the evaluation of tolerances will be. However, large sample size will increase the measurement time and the effect of machine drift due to temperature. Therefore, a trade-off between the benefit of additional sampling and the cost of increased time is to be made (benefit-cost analysis). This is the motivation behind this dissertation, which uses the LS technique to estimate a zone.

Although optimization strategies are routinely employed for minimum zone estimation in the current metrology literature, optimization and search are rarely utilized in sample point selection. Further, each manufacturing process in conjunction with the work-holding induces different types of deflections on the manufactured parts. These deflections result in specific patterns of surface errors. If the point that has got maximum error can be caught at the initial stages, then prolonging the measurement process may not be necessary, and the measurement time can be reduced [63]. But, little to no attempts are evident in quantifying and utilizing such patterns in sampling.

A methodology for sample reduction, based on optimization search heuristics, exploiting the knowledge of manufacturing surface pattern is proposed here. The interim result of search-based selection of sample points with an example has been presented in

[7-8] without quantifying manufacturing errors. The approach is illustrated with straightness and flatness evaluation. Straightness as defined in [3] is the condition where an element of a surface, or an axis, is a straight line. A straightness tolerance as shown in Figure 1, specifies a tolerance zone within which the considered element or derived median line must lie. Similarly, flatness is the condition of a surface having all elements in one plane. A flatness tolerance is specified through a tolerance zone defined by two minimum separating parallel planes within which the surface must lie in order to meet the functional requirements of a part.

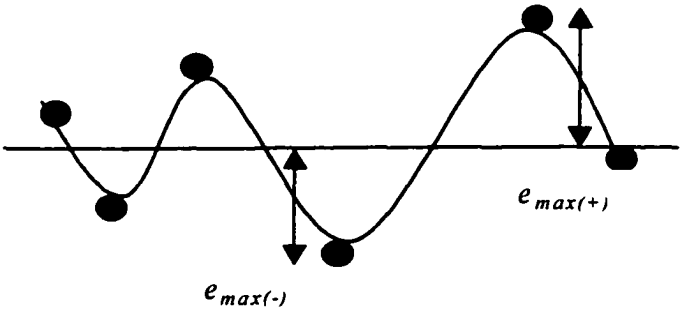


Figure 1: Sample points for a line feature, tolerance = $|e_{max(+)}| + |e_{max(-)}|$.

Continuing on the work [7-8], manufacturing surface profiles are also characterized. Kline et al. [48] have given surface error model for end milling using 7075-T6 aluminum workpiece and Gu et al. [35] for face milling using cast iron workpiece. Aluminum and cast iron plates are produced using end and face milling (fly-cutting) respectively for the present work. The surface patterns are quantified and the error models [35, 48] are verified. Initial locations to inspect and search directions are governed by the profile and geometry of the part surface. An ideal feature is fit to the initial set of points and a tolerance zone is computed. Additional points are sampled

using search methods, seeking improvement in the zone. Search is conducted in both the +ve and -ve directions from the fit feature and is stopped when a solution for the maximum deviation is realized. The two solution points are added to the initial set and the corresponding tolerance is determined. In doing so, an optimum sample size is obtained with respect to the cost-benefit analysis. The tolerance value is compared with that obtained for a very large population sample to verify the accuracy. In case of straightness, region-elimination search method [70] is employed. Tabu search [31] and a hybrid search are applied for flatness. The hybrid search developed is a combination of coordinate search [86], Hooke-Jeeves pattern search [70, 86] and tabu search. Extensive experimental analysis is conducted to verify the sturdiness and feasibility of the present adaptive sampling procedure. The results obtained signify the importance of the contribution and its potential for application in inspection for interchangeability.

The dissertation is organized as follows. Next chapter reviews the literature on CMM, sample design, manufacturing surface profiles, and form tolerance evaluation methods. Chapter 3 discusses the search methods employed. Objective and overview of the methodology are described in Chapter 4. Experimental verification of the manufacturing error profiles and collection of the population data are enumerated in Chapter 5. Chapter 6 explains the development of the search algorithms. Experimental design is discussed in Chapter 7. Chapter 8 analyzes the results and compares them with previous work. Chapter 9 contains the conclusions and suggestions for future work.

CHAPTER 2

LITERATURE REVIEW

Functional requirements or assembly conditions on a manufactured part are normally translated into geometric constraints to which the component must conform. These constraints are expressed in terms of ANSI Y14.5M-1994 [3]. In these standards, allowable variation of individual and related features is based on the 'envelope principle'; that is, entire surface of the part feature of interest must lie within two minimum separating envelopes of ideal shape. This minimum range is called form tolerance/error. It is important to use correct definition of tolerance specifications [71]. Commonly used form tolerances are straightness, flatness, circularity, cylindricity, sphericity, and conicity [21, 67, 74, 77]. Etesami [26] also includes profile of a surface as a kind of form tolerance. Formal definition of form, using computational geometry concepts with 2-D measurement data is given in [27]. To determine form tolerances, CMMs are used to take sample measurements representing a feature. The measurements are taken based on a sampling plan. If manufacturing surface errors can be quantified, it will aid in sampling. Form error evaluation algorithms are used to analyze the sampled data. The results obtained depend upon the measuring errors, tolerance specified, manufacturing process, sampling strategy, and form error estimation method [36]. In this chapter, a review is presented on CMM, sample design, manufacturing surface profiles, and form error evaluation methods.

2.1 Coordinate Measuring Machine

A coordinate measuring machine (CMM) is a computer-controlled device that uses a probe to get measurements [21, 22, 75], generally one point at a time, on a manufactured component's surface. Probe movements may be programmed or determined manually by operation of a joystick. An algorithm is used to relate the sample of coordinate data to obtain the information about a form. Since CMMs are flexible, adequately accurate, and can be easily automated, therefore they are popular in inspection of mechanical parts [50]. Now, they are also being used for statistical process or quality control in-process inspection to assess trends in manufacturing processes. This trend will increase as new CMM designs become less sensitive to shop floor conditions.

Although CMMs have gained tremendous popularity, difficulties arise with its use. Its measurements are associated with unavoidable imperfections (straightness and perpendicularity deviations in the guides, probe deformations, errors in the length measuring system, etc.) and external influences (temperature changes, vibration, etc.) which lead to random and systematic errors [89]. There are a few methods available in the literature to assess and/or reduce the measurement uncertainties [24, 39, 76]. Tolerance standards require knowledge of the entire surface, whereas a CMM measures only a small sample of the surface. Sometime CMM algorithms may also draw incorrect conclusions. As a result, it may accept bad parts and reject good ones. Therefore, the estimation accuracy depends on the sample size and the estimation method. Selection of an appropriate sample size involves a trade-off between the cost of taking additional measurements and the cost of making incorrect decisions. The choice of estimation method involves a trade-off between the ease of use and the accuracy of the estimation.

2.2 Sample Design

An important decision in any CMM application is the determination of the number and location of probe measurements to collect. If enough data points are not collected, then a large discrepancy may occur in the results [17]. Little work has been done on design issues and there exist many research opportunities.

2.2.1 Sample Size

Selection of a sample size for CMM inspection is affected in general by factors such as the size of a feature, the tolerance specification (band), and the machining process capability. A bigger size of a feature may need a larger sample size to cover the entire surface. For a specified tolerance band, a machine with a better process capability requires a smaller sample size. For the same process capability, a tighter tolerance band requires a larger sample size. Industries prefer small sample sizes; e.g., 3-5 points for a line feature, 5-8 for a plane, 4-8 for a circle [22]. This is to minimize the measurement time and to reduce the effect of machine drift; without giving any formal consideration to costs associated with the inspection errors. Therefore, a trade-off needs to be made in determining a suitable sample size, which can represent the entire population with sufficient confidence and accuracy. It is to be noted that there is a difference between the elimination of some data points from a sample set in order to decrease computational time and the reduction of a sample set to shorten the measurement time. Some techniques concerning the former issue have been suggested in [44, 65]. The latter is the subject of this study.

Weckenmann et al. [89] discussed how the use of functionality-oriented evaluation procedures requires a sampling strategy that establishes a certain minimum number and optimal distribution of data points for sufficient reliability. As the number of data points increases, the estimated values converge to the 'true' value of form error and the sampling strategy dispersion converges to zero. Babu et al. [6] simulated different measurement conditions and suggested that information regarding the manufacturing process, measurement uncertainty, and the tolerance on the part is required to choose a proper sampling plan. Zhang et al. [100] reached similar conclusion about the factors with the addition of the size of a feature, which may affect the sample size required for CMM. They used a back-propagation neural network approach in their study of 'hole' features. They measured the diameter of the hole for different number of sample points, and found the tolerance band as the difference in the average diameters for one sample size to other.

Weckenmann et al. [90] considered the effect of various sample sizes on least squares (LS) estimates of the parameters describing a circular feature. They concluded that 10 to 20 points are needed to obtain sufficient precision on parameter estimates, which is twice the sample size used in practice. In a study on circularity, Chang and Lin [14] found a relationship between the relative measurement error U and the sample size by a Monte Carlo simulation method for the ith-order harmonic function. For example, at U = 20% and i =20, they reported that the minimum sample size should be 35. Damodaran et al. [18] investigated the effect of size of data point sets and radius of the component using different algorithms for the determination of minimum circularity

tolerance zone. They reported that an adequate sample size ranges from 15 for a component of radius 2 inches to 25 for a component of radius 20 inches.

Hurt [41] noticed that the accuracy of the LS estimate improves by measuring more points and 20 or more points are needed in order to get a reasonable flatness value. Caskey et al. [13] studied the behavior of a plane fit using the sample sizes in perfect squares of 4, 9, 16, 25, 36, and 49, and could not find the sampling strategy dispersion converging to zero even with 49 points. For a line of 40 in. (sample sizes of 5, 25, 50) and a plane of 20×20 in. (sample sizes of 4, 9, 16, 25, 36, 49), Hocken et al. [36] reported that the parameters described in both the fit line and plane did not converge to stable values until a larger number of points (> 50) were sampled. All the three studies were based on simulation.

Menq et al. [56] took a more theoretical approach to the choice of sample size. Standard statistical methods were used to develop a hypothesis test on the variance of the residuals that result from a LS fit; a large variance means an unacceptable part. A sample size formula was then derived for the desired levels of Type I and II errors. The result shows that the appropriate sample size should depend on both the tolerance specification and the variability of the manufacturing process. As the tolerance gets tighter the sample size increases, while as the process capability improves the sample size decreases (also mentioned by Yau and Menq [94]). The validity of the result depends on having normally distributed deviations, which may not be true in the presence of systematic errors. Yau and Menq [95] and Yau [96] used a sensitivity measure to relate the uncertainty (or accuracy) of the evaluated best-fit parameters to the dimensional errors

and the number of measurements & their locations. In agreement with Babu et al. [6] and Zhang et al. [100], they reported that as the variance of specified tolerance (depending on manufacturing process and process variability) increases, the uncertainty of the evaluated error increases. On the other hand, as the number of measurement points increases, the uncertainty decreases. As an example, for a 2D circle with uniform distribution of data points, they reported that the uncertainty was proportional to the magnitude of dimensional error and inversely proportional to the square root of the number of points $\left(\sigma = s\sqrt{2/n}\right)$. For a 99.7% confidence level, uncertainty U = 3σ. Interestingly, they found no effect of the size of the circle radius, which is in contrast to the findings of Zhang et al. [100] and Damodaran et al. [18]. For a general 3D case, Yau [96] indicated that the sensitivity as well as uncertainty values are in general inversely proportional to the square root of the sample size. This helps in finding a suitable number of points that will control the best-fit result to satisfy the allowable uncertainty constraints.

Namboothiri and Shunmugam [63] introduced a new parameter, based on the asymptotic distribution of the form errors and with the assumption that the errors follow a normal distribution, which is a function of the sample size and the corresponding values of errors. The parameter gives the probability that the form error is less than a particular value. Thus, form error can be measured accurately with less measurement time with the help of this parameter. They also realized the importance of a sampling pattern in measurements. But they didn't propose any technique to characterize the pattern. The other drawback in their study is that the measurements were based on simulation. The present study measures actual points on the surface using a CMM.

Sweet et al. [85] used statistical approach to determine the minimum number of measurements to make at each location and their location in order to locate a part within a specified confidence interval. For planes, if the variances of the measurement errors and the machined surface imperfections are known, they recommend 12 locations: one measurement in each corner and two measurements on each side of the rectangle, to satisfy the confidence interval. If the variances are unknown, then eight locations may be used, with one measurement at each corner and one measurement in the middle of each side. For circles, with one measurement at each location, they found that the results for 20 locations were better than that of four locations. Also, it was better to use equally spaced angles for locations. It is to be noted that the problem of part location is similar to problems encountered when using CMMs, but some aspects are different enough to make them independent.

2.2.2 Sample Locations

Most CMM guidelines suggest users to distribute measurements evenly on the surface being measured [95-96]. Three sampling schemes most commonly discussed in the CMM literature are uniform or equidistant, totally randomized, and a simple form of stratified sampling called randomized block or randomized grid. Cochran [16] provided a general reference for theory related to these methods. Simple random sampling is a sampling method whereby a sample of n units is selected out of a population of N units such that every one of the n sample units has an equal chance of being chosen [16]. The two types of simple random sampling are sampling with replacement and sampling without replacement. The difference is that the latter makes it possible for the same n

units to be redrawn since the units are returned into the population after selection. In stratified sampling the population of N units is divided into sub-populations, or strata, of N_1, N_2, ..., N_L units. Then sampling is obtained by performing simple random sampling on each stratum [9]. Systematic (uniform) sampling is a sampling method where by the first unit selected determines the whole sample. The first unit is randomly generated while the others are selected automatically by some predetermined pattern. The pattern used in selecting a systematic sample is simple with regular spacing of units [9]. A population consists of N units numbered from 1 to N in some order [16]. To select a sample of n units, a random number, i, less than or equal to k, where $N = k \times n$, is drawn. Then, the ith unit and every kth subsequent unit will make a sample of n units. The sample of n units consists of units ordered as i, $i + k$, $i + 2k$, ..., $i + (n - 1)k$. Say for example of $n = 4$ units where $N = 40$ and $k = 10$, if the first drawn number is 8, the subsequent units are numbers 18, 28, and 38. Dowling et al. [22] have mentioned that a simulation study to test the behavior of CMM fitting algorithms under these three sampling schemes has been done. Results indicated that stratified sampling performs better than random sampling. This is obvious because stratified sampling ensures better coverage of the entire profile. Uniform sampling is the easiest and has this advantage as well; however it can have problems when there are periodic features in the data. Caskey et al. [13] used stratified sampling and believed, but have not proven, that this type of sampling would be more robust to feature waviness. Hocken et al. [36] performed both uniform and stratified sampling.

Alternative designs providing good feature coverage should also be considered. McKay et al. [55] suggested a latin hypercube sampling (LHS) for the case of high-dimensional spaces. This technique assures a dense and even coverage of the entire range when the design is projected onto a single dimension. They and Stein [83] showed that the variance in a predicted surface is reduced when LHS is used instead of random or stratified sampling. Another option is Hammersley sequence sampling [52, 91, 92, 93] for planar features. The results indicated that Hammersley sequence requires fewer observations than uniform sampling to achieve a pre-specified accuracy (a nearly quadratic reduction in the number of points, i.e. loosely speaking 250,000 points reduce to 500). Hammersley sequence is also more accurate than that of random samples [52]. For a workpiece with multiple geometric features forming strata, the stratified Hammersley sampling is more robust than the stratified random sampling and stratified uniform sampling [52]. Woo et al. [93] compared Hammersley and Halton-Zaremba sequences and could not find any significant difference in their performance. The choice is largely a matter of convenience as Halton-Zaremba requires the sample size to be a power of 2, i.e., 2, 4, 8, ...

Besides the accuracy and sample size, if the length of a CMM probe path is also considered, then it can be concluded for flatness that any one sampling strategy may not be the best for all cases [46-47]. It is possible that the most accurate sampling method may not be the most efficient one. Hocken et al. [36] pointed out that on real CMMs the software provided by the manufacturer is limited. Hence, it is easier to program a measuring machine to do equal intervals in angle or space (uniform sampling) than it is to use any of the other sampling techniques. Yau [96] adopted the conjugate gradient

method to search for the optimal sampling positions given a fixed number of measurement points. To start the iterative search, he chose uniform parameter distribution to determine the initial values. He concluded that the long time taken by this method for marginal improvement in the measurement result was not justified.

Expert knowledge of the part surface should be incorporated in the choice of sampling design. For example, an operator may avoid probing areas near holes or edges that are felt to be unrepresentative of the surface. Kurfess and Banks [51] suggested the possibility of exploiting knowledge about locations of likely deformations depending on the manufacturing process. It was assumed that the measurement error has a multivariate Gaussian distribution. They showed their method on cylindricity. However, they did not employ any actual error calculations based on processing and work-holding, nor did they do any search starting with error locations. Namboothiri and Shunmugam [63] also realized the importance of a sampling pattern in measurements. If the point that has got maximum error can be caught at the initial stages following a sampling pattern, then further prolonging the measurement process is not necessary, and the measurement time can be reduced.

It is perceived that exact locations of errors are often difficult to mathematically model. However, if approximate locations are determined prior to inspection, searching for the exact locations is viable using a prudent application of search methods. Also, rather than fit a generic or empirical function to describe a surface, actual modeling of errors based on process constraints is more desirable.

2.3 Surface Profiles in Manufacturing

Surface profiles obtained in manufacturing include surface texture as well as surface error. Surface texture or finish, which includes roughness, waviness, lay, and flaws, is used to describe the general characteristics of a workpiece surface [23, 43]. Roughness consists of finer irregularities over a sampling length or cutoff smaller than that of waviness and is described in terms of arithmetic mean R_a or root-mean-square (RMS) value. Surface error is defined as the deviation of the finished machined surface from the nominal or desired surface, that is, the surface that would be produced by a completely rigid machining system [48, 84].

Texture of machined surfaces has deterministic and stochastic components in it [98]. The deterministic portion of a surface profile is formed mainly by the geometry of cutting action, namely, the tool geometry and cutting parameters. The random portion is the effect of the complex interaction of workpiece material properties, tool vibration, metal shearing during the chip formation, etc. Without considering the dynamics in the tool-workpiece system, You and Ehmann [97] extended their earlier proposed method for face milling, based on the superposition of a tertiary motion onto the conventional cutter motions, to ball nose end milling. It has been shown that the RMS value of a milled surface roughness can be reduced to a minimum by introducing a proper spindle eccentricity for a given set of machining conditions. Accordingly, a wide variety of the milled surface topographies can be obtained with tertiary cutter motions.

When structural vibrations are present during milling, Montgomery and Altintas [57] have mentioned that the relative motion between the tool and workpiece influences the uncut chip thickness variation. The cutting forces are modulated with changes in the

uncut chip thickness, which in turn excite the tool-workpiece structure, resulting in more changes to the uncut chip thickness. This process has closed loop dynamics, which may result in self-excited or chatter vibrations. They presented a comprehensive dynamic milling model, which includes the tool geometry and the vibrations of both workpiece and tool in any direction. The model analyzes primarily the peripheral milling dynamics and is valid for both rigid and flexible workpiece-tool system. The model explains the mechanism of chip thickness and predicts the topology of finished workpiece surface, cutting forces in the feeding (ploughing) and normal shearing directions, and vibrations of both the tool and work-piece simultaneously. Zhang and Kapoor [98] made an assumption that the stochastic component comes from the tool vibratory motion caused by the random excitation, which originates from the force variation due to the non-homogeneous distribution of micro hardness in the workpiece material. They developed a model to describe the phenomenon of random excitation. Zhang and Kapoor [99] presented a procedure to dynamically construct the surface topography using the results of the random excitation model and the geometric action of cutting tool. This way, the R_a value of a surface roughness profile under consideration can be obtained without being measured on a surface profilometer. Their experimental and simulation results for a turning operation confirm that when a small feed is used, the influence of the spiral trajectory of tool geometrical motion on the surface generation decreases dramatically and the random excitation system, on the opposite, is strengthened playing a significant role in surface texture generation. In face milling, Radulescu et al. [68] recommend that variable speed machining is safer to use than constant speed machining when the effects

of the tool-work dynamics and geometry on the vibration of the cutting process are hard to predict.

It is typical that the manufacturing process and work-holding (fixturing) apply different forces that manifest in the workpiece as surface errors. Significant literature is available in the modeling of deflection and work surface characterization based on material characteristics and the process employed. Ehmann et al. [101] have presented an extensive review of the investigations on the modeling of milling process. In the present work, end and face milling have been considered as the processes to produce plates. For end milling, standard cantilevered beam equations have been employed by Kline et al. [48] to quantify the tool deflection using the y-direction cutting forces, a mechanistic force model. They have also showed a method for calculating the workpiece deflection and surface errors using the Finite Element Method. The rectangular plate element was used to model the thin-walled workpiece. They have compared these calculated surface errors against the measured errors for a series of machining experiments for climb or down milling without coolant on rigid and flexible 7075-T6 aluminum work pieces. They reported a good match, particularly for the rigid workpieces. In their model, primary components of the error were the cutter and workpiece deflections. Factors, such as chip formation process and vibration, which contribute significantly in case of light cuts, producing surface errors of the order of 25 μm, were not included. They also discussed the effect of cutter run out on the surface generation mechanism. However, for their study the run out level was insignificant.

The model of Kline et al. [48] has produced good results for a class of problems in which cutter and workpiece deflections were not too large. Large deflections have a significant effect on the chip load and thus on the cutting forces and surface error. For such cases, both the peak cutting forces and surface errors when determined from the chip load calculated for a rigid system, may be over-estimated by as much as 100-150%. In order to incorporate the flexible system chip loads, Sutherland and DeVor [84] presented an iterative method to balance the cutting forces and deflections generated in end milling. The flexible system model predictions of forces and surface error were compared against both measured and rigid system model-predicted values associated with the machining conditions for a series of down milling experiments on 390 casting aluminum alloy. Instantaneous cutting forces were measured with a Kistler dynamometer and the surface errors were measured with a dial indicator. It was shown that the enhanced chip load model gives predictions of both cutting force signatures and surface error profiles that are much better than the rigid system chip load model, particularly for long cutters. This method can be used for a flexible end milling system which may result from the use of a long cutter, machining a thin-walled work piece, using a light radial depth of cut, or in general, any cut in which the deflections of the end mill or workpiece are large relative to the radial depth of cut. They also demonstrated and discussed the fact that system deflections temper the effects of run out and reduce both peak cutting force and maximum surface error.

Tsai and Liao [102] extended the work of [48, 84] and presented a dynamic finite element model to analyze the surface errors in peripheral milling of thin-walled workpieces. They modeled the helical fluted end mill and the workpiece with the pre-

twisted Timoshenko-beam element and a 3-D isoparametric 12-node element, respectively to more accurately simulate both the geometries as well as the thickness variations of the workpiece during milling. They adopted the flexible system cutting force model of [84] to estimate the dynamic cutting forces. However, their results with the dynamic and static models for the steady milling were very close. In [48], the maximum surface error occurred at the bottom of cut assuming the workpiece to be rigid. But, for the thin-walled workpiece, the flexibility may lead to an increase in the surface errors, the location of maximum error may shift upward, and there may exist larger errors at the top of cut [102].

Ismail et al. [42] developed a mechanistic model, which predicts surface roughness as well as surface profile error for surface generation in end milling, specifically peripheral milling. The model includes the effects of tool flank wear, tool vibrations, as well as run out. The results obtained from the simulation runs were verified by conducting experiments on 7075-T6 aluminum alloy workpieces. The change in the slope of the feed marks on the workpiece due to cutter run out, were noticed. The feed marks near the bottom of the cut are considerably shallower as compared to those at the top of the cut, which results in the generation of flat bands at the location of maximum cutter deflection in the feed direction. The peak-to-valley surface roughness is found to be higher at higher speed because of the larger vibration amplitude associated with the higher speed. It is observed that the wear land smears the high peaks resulting in lower peak-to-valley roughness at any height and especially at the bottom of the cut. The ploughing force acting on the wear land, influences the profile error. The error generated

with the worn tool is significantly smaller in magnitude than that obtained with the sharp tool.

For face milling, Fu et al. [28] and Kim and Ehmann [45] calculated the instantaneous chip load (thickness) on the tooth as $f_t \sin\theta_i$ where f_t is the feed per tooth and θ_i is the angular position of the tooth at that instant. The instantaneous chip cross section is then given by $f_t \sin\theta_i \times$ depth of cut, assuming that the depth of cut variation is negligible. Fu et al. [28] assumed that the normal force acting on the chip cross-section is the product of the cross-sectional area and the specific cutting pressure K_T. The radial force acting along the cutting edge is obtained by multiplying the normal force by an empirical constant K_R. Balasubramanian and Raman [10] used these equations to calculate forces while comparing alternate tool paths in face milling path planning. Gu et al. [35] presented a model to predict surface flatness error in face milling. The model includes the machining conditions, deflection of the cutter-spindle and workpiece-fixture, and static spindle axis tilt. They have used the influence coefficients to compute the tool-workpiece deflections at the points of cutting force applications.

The present research uses prior knowledge of approximate solutions of surface errors induced by manufacturing processes to aid the search of the CMM probe. The processes are modeled on the machining conditions given in Kline et al. [48] and Gu et al. [35]. Therefore, these two studies are again described in chapter 5.

2.4 Form Error Estimation Methods

A given set of CMM data can be analyzed by a variety of techniques [25] to obtain form error/tolerance. These include analysis based on point-to-point measurements and techniques based on curve fitting. Curve fitting can be viewed as an optimization problem of finding the parameters of substitute geometry that optimize a particular fitting objective for a set of points [37]. For this, first the corresponding deviation for each data from the ideal feature is obtained. Then a method for assessing feature conformance to form is used to describe the underlying part geometry, which is required for decision-making.

Expressions for the linear and orthogonal deviations for different kind of features are given in [59, 78]. Suppose that a line feature is known to lie in a horizontal plane, so that the measured points can be expressed in terms of (x, y) coordinates only. Let the ideal form be defined by $y_0 + l_0 x$, then the normal deviation corresponding to (x_i, y_i) is given by $\hat{e}_i = [y_i - (y_0 + l_0 x_i)] \times [1/(1 + l_0^2)]^{1/2}$. The value is positive if the point is above the line and negative if it falls below. If the feature is well aligned with the x-axis, the deviation can be expressed in the linear form $e_i = y_i - (y_0 + l_0 x_i)$. For flatness, the assessment plane is represented by $z_0 + l_0 x + m_0 y$ and $\hat{e}_i = [z_i - (z_0 + l_0 x_i + m_0 y_i)] \times [1/(1 + l_0^2 + m_0^2)]^{1/2}$. For a surface well aligned parallel to the x-y plane, $e_i = z_i - (z_0 + l_0 x_i + m_0 y_i)$. Linearizing approximations for other shapes also require certain feature orientations relative to the part coordinate system to work well. Corresponding

expressions for circularity, cylindricity, and sphericity can be obtained from [78] and for conicity from [67].

Many fitting criteria can be expressed as special cases of a general criterion called L_p-norm estimation [37]. The problem is to find the fit parameters that minimize $L_p = \left[\frac{1}{n}\sum_i |e_i|^p\right]^{1/p}$. The two most common fitting techniques are the minimum-zone (MZ) and the least squares (LS) techniques [22]. The LS fitting corresponds to the case $p = 2$. The limit of L_p as p goes to infinity is the largest magnitude residual, so that the L_∞ problem is minimizing the maximum magnitude residual, i.e., finding the MZ fit. It should be noted, however, that the two methods give seriously biased estimates of the part form tolerance when the sample size is small [21]. In order to reduce the bias of the two methods, some jackknife estimates [75] or Huber loss statistics [38] can be used. Damodaran et al. [18] compared the accuracy of the minimum circularity tolerance zone obtained from the five algorithms: two point method, three point method, minimum spanning circle algorithm, least squares method, and voronoi method. They concluded that the voronoi and the two point algorithms consistently give the most accurate estimates.

2.4.1 Minimum-Zone Method

The ANSI specification [3] for form tolerance describes allowable deviations from the ideal form in terms of the normal distance between the maximum inscribing and minimum circumscribing features that bound the entire feature of interest. The minimum-zone (MZ) method works on the same basis, but it encloses the set of sample

points [21-22]. The ANSI standard does not specify a method for establishing the minimum zone. Various algorithms have been developed for different features. One of the earliest papers dealing with the MZ principles was that of Murthy and Abdin [59]. The problem was formulated as a minimax problem. Concerning the search techniques to arrive at the solutions, they recommended using the simplex search for any surface, the random (Monte Carlo) search if the associated variables are few, and the spiral search when the variables are two or three. This search, however, requires extensive computer time. Fukuda and Shimokohbe [29] developed algorithms based on the minimax approximation but did not use the simplex search. The algorithm, as an example for flatness evaluation, determines the orientation of an ideal geometric plane in iterative steps such that the maximum value of the absolute residuals of the data set was the minimum. The initial plane was formed through four points that were selected randomly from the data set. During each iteration step, a point having the maximum residual error with the previously established plane was identified to replace one of the four points that formed the plane. They claimed that the time required using their algorithm for a large number of points would be less than that using even the least squares technique.

Shunmugam [77] introduced a heuristic method called the median technique, i.e., $|e_{max}|=|e_{min}|$ to obtain the minimum value of a form error $h_t = |e_{max}|+|e_{min}|$. Minimizing $f= |e_{max}|+|e_{min}|$ does not fix the position (y_0 or z_0 in a line or plane) or the size (R_0 of a circle, cylinder or sphere). Therefore, Shunmugam [78] suggested a modified function $f= |e_{max}|+|e_{min}| + 1/[|e_{max}||e_{min}|]$ to be minimized which yields a minimum value of h_t and unique values of the parameters such that $|e_{max}|=|e_{min}|$.

The simplex search method was employed. Murthy and Abdin [59] and Shunmugam [78] compared the MZ obtained from the linear (e_i) and normal deviations ($\hat{e}_i$), and found no significant difference when the errors are of minute degree. Shunmugam [81] presented a generalized algorithm to establish the reference figures outlined in the standards. The enveloping figures serve as soft-gages and can replace conventional gages used for inspection. The algorithm is also used to find the minimum separation enveloping surfaces that may be useful in closer fitting assemblies. The MZ figure is established on the basis of the theory of discrete and linear Chebyshev approximation.

Traband et al. [87] gave algorithms that guarantee the minimum zones for straightness and flatness based on the convex hull of the sample of points. Roy and Xu [72] developed an algorithm to determine the MZ for cylindricity convex hulls and voronoi diagrams. Roy and Zhang [73] presented a similar algorithm for circularity. Allada and Anand [1] used the Hough transform method to obtain the MZ of manufactured parts for verifying straightness, flatness, perpendicularity, parallelism, and angularity tolerances. This is a point-to-point measurement analysis [25]. Since the MZ is calculated based on each of the measured data points without any approximation, the estimate is accurate. Damodarasamy and Anand [19] proposed a methodology to determine a true MZ for flatness for a given set of data points using the simplex search. The perpendicular distance p from the origin to a normal plane intersecting each point in the data set is determined. The difference between the maximum p and the minimum p for a particular set of parameters (direction cosine angles) determines the MZ.

Wang [88] transformed the minimax problem into a constrained nonlinear optimization problem (NLOP) that contains an extra variable h, denoting the half width of the MZ. The reformulated problem was solved using the sequential quadratic programming (SQP) method. This method guarantees the MZ. Considering the difficulty of solving a constrained NLOP, Carr and Ferreira [11-12] suggested a kind of transformed L_p-norm solution model; so the objective function and all but one constraint are made linear. The employed linearization process does not change the optimization problem nor is there a loss of generality. They developed algorithms that solve a sequence of linear programs, which converge to the solution of the nonlinear problem. The algorithms can be used for evaluating an exact value of straightness, flatness, and cylindricity for a given sample set.

There exist algorithms using unconstrained nonlinear optimization method (NLOM), which can also find an exact MZ corresponding to the measured data set. One such algorithm developed by Elmaraghy et al. [25] is based on point-to-point measurement analysis. Their study deals with one geometric feature only, the cylinder. With respect to this feature, the algorithm can determine the size tolerance and eight different geometric deviations. The method uses the Hooke-Jeeves direct search procedure [70]. Other unconstrained NLOM algorithms for straightness [65] and flatness [44] are based on curve fitting. These algorithms utilize a downhill simplex search method. The initial condition of the NLOM is obtained by the least square (LS) method, then the MZ is optimized. In addition to straightness, Orady et al. [64] have listed vector of the parameters representing a fitting geometric feature and objective function for flatness, circularity, and cylindricity. In order to decrease computational time for

flatness, Kanada and Suzuki [44] suggested to reduce the data set containing z for different (x, y), by using only those values that are far from the mean μ, i.e., $|z - \mu| \geq 1.5\sigma$. If the data z_i are distributed normally, this way 86.6% of the original data set is thrown out. Taking the advantage of the LS results, Orady et al. [65] advised to filter out the outlier points using Grubbs statistical control and to further reduce the data set using a simple control zone specified by them (the data points inside the control zone are deleted from NLOM).

A drawback of the minimum deviation procedure is that only a few points on the feature control the position/size and slope/center. Therefore, Shunmugam [79-80] proposed a different criterion minimizing the sum of absolute deviation values $f = \sum |e_i|$. This is called the minimum average deviation method and it determines the ideal surface in such a way that the areas above and below it are equal. Therefore, the sum of the areas is a minimum. This requires simplex search procedure for the solution. In a mathematical sense, this formulation represents an L_1-norm. Namboothiri and Shunmugam [61-62] using the L_1-approximation developed a general algorithm for function-oriented form evaluation. This helps identify the wild points present on the surface of a part that can be compensated by further operation (machining) to produce a better quality part.

Another problem in finding an exact MZ is that the methods are combinatorial in nature and take a substantial amount of computing time when many points are involved, $O(n \log n)$ time for the line and $O(n^2)$ time for planes [22]. Further, exact solutions are not available for more complex features. Computational difficulties of the MZ

algorithms preclude their use in practice, but many researchers advocate the approach on theoretical grounds. They argue that the method retains the spirit of the tolerance standards, and the resulting zone width will coincide with the true deviation range in the ideal case in which sampling is infinite and dense, and measurement error is negligible. This viewpoint, however, overlooks the fact that the method tends to underestimate a feature's true deviation range for the small sample sizes used in practice (assuming no measurement error). This downward bias as well as a slow convergence to the correct value when the sample size increases, has been illustrated in [21].

2.4.2 Least Squares Method

The least squares (LS) method fits an ideal form to coordinate data by minimizing the sum of squared deviations $\sum e_i^2$ or $\sum \hat{e}_i^2$. Solutions to the LS parameters for line, circle, plane, cylinder, and sphere are given in [77] when $\sum e_i^2$ is minimized. For symmetrical and equispaced data points, the simplified solutions are presented in [79-80]. The estimated deviation range is taken to be the sum of the maximum absolute residuals on either side of the fitted surface, $h_t = |e_{max}| + |e_{min}|$. To be more accurate, multiply h_t by $[1/(1 + l_0^2)]^{1/2}$ for straightness and by $[1/(1 + l_0^2 + m_0^2)]^{1/2}$ for flatness, respectively to account for normal distance [59, 87].

If the residuals considered are normal to the fitted feature, the method is often called as normal/orthogonal least squares (NLS), and $\sum \hat{e}_i^2$ is minimized. An analytical NLS solution for straight lines is presented in [59]. This reference also contains a deterministic solution for planes assuming symmetric data points in the x-y plane to

simplify the equations. This assumption may not be reasonable for many CMMs and it would be better to solve iteratively. For other features, the NLS method requires a search procedure to arrive at the estimates. An iterative algorithm like Gauss-Newton [22] or Newton-Raphson [56] can be employed to obtain the solution. Some approximation algorithms, which lead to faster computations, can be used as well.

Murthy and Abdin [59] state that when the deviations are small, the difference in the LS and NLS methods are often not significant. Therefore, Shunmugam [78] suggests that the larger computation time for NLS is not justifiable in view of the marginal difference in values.

In comparison to the MZ method, the LS method is superior in terms of calculation time [44]. As discussed in [22], MZ tends to underestimate a feature's deviation range by treating few measurements in practice to be representative of the entire surface, assuming that measurement error is negligible. Thus, it is very sensitive to asperities (such as that caused by dirt or scratches on the surface), which render the result useless if they go undetected [95]. This is an issue for LS as well, but to a lesser extent, since it treats the data as a sample rather than as the entire population. LS generally over estimates the form error, so the values are larger than those obtained with MZ [44, 59, 64, 65 78, 79]. However, Dowling et al. [21] showed that the estimates of the deviation range for straightness and flatness from the LS approach have better statistical properties than that of the MZ estimates. Thus LS is more reliable, flexible, and suitable for practical applications [95] and fitting algorithms provided with CMMs are generally its variants. The implementation of optimization routines and approximation algorithms, however, is not standardized among manufacturers of CMMs. Therefore differences in

results across machines exist. In this work, the LS technique has been used for minimizing $\sum e_i^2$.

The present study focuses on exploiting the knowledge of manufacturing surface pattern for initial sampling and then adding data points intelligently with the help of search methods. Next chapter includes a review on optimization search methods.

CHAPTER 3

OPTIMIZATION SEARCH METHODS

As discussed in Chapter 2, straightness is the condition where an element of a surface is a straight line, which is a function of one variable. Flatness is the condition of a surface having all elements in one plane. Hence, gathering of sample points for straightness estimation is similar to optimizing functions of a single variable. Sampling for flatness evaluation resembles optimization problems suitable for pattern search methods. Therefore, optimization methods for functions of one-variable and pattern search methods are described in this chapter.

3.1 Optimization of Functions of One Variable

Reklaitis et al. [70] have discussed how to determine the global optimum of a function of one variable, i.e., $f(x)$. A simple approach is to compute all local optima and choose the best. If a function is bounded in an interval $a \leq x \leq b$, then in addition to the stationary points belonging to the interval, the boundary points may also qualify for the local optimum. Stationary points are obtained by setting $df/dx = 0$. For a minimization problem, if the value of f at the boundary points and the stationary points are found; then the smallest value corresponds to the global minimum point. However, instead of evaluating all the stationary points and their functional values, certain properties of the function can be used to determine the global optimum faster. For example, for unimodal functions, a local optimum is the global optimum. There exists a special class of

unimodal functions, called convex and concave functions, which can be identified with some simple tests. A function is convex, if f' is in increasing or at least non-decreasing order as x increases, and $f'' \geq 0$ for all x in the interval. A function is concave if $-f$ is convex. For a convex function, local minimum is the global minimum, and for a concave function, local maximum is the global maximum.

For general functions, we can employ different search methods for locating the optimal point in a given interval. Methods that locate an optimum by successively eliminating subintervals are called region-elimination methods [70]. Whether a function is continuous, discontinuous, or discrete, assume that within the domain of interest say (a, b), the function is unimodal and convex for a minimization problem. Let x_1 and x_2 be two points in the interval such that $a < x_1 < x_2 < b$. Compare the functional values at x_1 and x_2. If $f(x_1) > f(x_2)$, then the minimum does not lie in (a, x_1). If $f(x_1) < f(x_2)$, then the minimum does not lie in (x_2, b). If $f(x_1) = f(x_2)$, the minimum does not lie in (a, x_1) and (x_2, b), and must occur in (x_1, x_2) provided f is strictly unimodal. Thus, the subinterval in which the optimum does not lie is eliminated from the search. This way a search is organized in which the optimum is found by recursively eliminating sections of the initial bounded interval. When the remaining subinterval is reduced to a sufficiently small length, the search is terminated. These search methods require only functional evaluations and functions need not be differentiable.

The region-elimination methods can be broken down into two phases: bounding phase and interval refinement phase. In the initial bounding phase with an arbitrarily selected starting point, the optimum is roughly bracketed within a finite interval using the

elimination property. An example is Swann's method [70]. Once a bracket has been established around the optimum, more sophisticated interval refinement phase can be applied. Interval halving method eliminates exactly one-half the interval at each stage. This is also known as a three-point equal-interval search procedure [70] since it works with three equally spaced trial points x_1, x_{mid}, and x_2 spaced at one-fourth of the interval (a, b). At each subsequent step, at most two functional evaluations are necessary since the midpoint of subsequent intervals is always equal to one of the previous trial points x_1, x_2, or x_{mid}. Hence, after $n/2$ iteration (n functional evaluation), the initial search interval of unit length will be reduced to $(1/2)^{n/2}$. Golden section search [70] is another interval refinement method. The interval (a, b) is rescaled to unit length (0, 1) by defining a new variable w in place of x. Inside the unit interval, two trial points are located at a fraction $\tau = 0.618$ and $1-\tau = \tau^2$ from either end. With this symmetric placement, regardless of which of the corresponding function values is smaller, the length of the remaining interval is always τ. The interval remaining after n evaluation will be of the length τ^{n-1}. The search can be terminated either by specifying a limit on the number of evaluation or relative accuracy in the function value. Preferably both tests should be applied. The golden section search out-performs the interval halving in terms of the fractional reduction of the interval achieved for the same number of functional evaluation or the number of functional evaluations required for the same accuracy.

A smooth function can be approximated by a polynomial, and the approximating polynomial can be used to predict the location of the optimum. For this strategy to be effective, the function needs to be unimodal and continuous over the interval. Quadratic

estimation method and successive quadratic estimation method (Powell's method) are the polynomial approximation or point-estimation methods [70]. In addition to unimodality and continuity, if a function is differentiable, further efficiencies in the search could be achieved employing methods that require derivatives like Newton-Raphson method, bisection method, secant method, and cubic search method [70].

3.2 Pattern Search Methods

In optimization, there exist problems in which the decision variables are discrete, and are known as combinatorial problems [69]. Many combinatorial problems cannot be solved exactly using reasonable computer time and space, even when there are only a moderate number of decision variables. Therefore researchers continue to develop good heuristics, i.e., algorithms efficient with respect to computing time and storage space, and with a likelihood of delivering a solution relatively close to an optimal one [69, 82]. These heuristics can even be applied to continuously differentiable function f: $\mathbf{R}^n \rightarrow \mathbf{R}$. The main problem with a number of heuristic algorithms is their inability to cope with locally optimal points, i.e. their inability to continue the search for a global optimum after a local one is reached. Therefore, it may not be possible to state how close to optimality a particular heuristic solution is. However, many modern heuristic techniques do give high-quality solutions in practice [69].

Direct search methods for a minimization problem are methods that neither compute nor explicitly approximate derivatives of f, instead they require only values of f to proceed [15, 70, 86]. Commonly adopted direct search techniques are random search,

simplex or S^2 search, and pattern search. It should be noted that this simplex search has no relationship to the simplex method of linear programming. Pattern search (PS) methods are descent methods. Examples of such algorithms are coordinate search with specified step sizes, evolutionary operation using factorial designs, Hooke-Jeeves pattern search, tabu search, and Dennis & Torczon's multidirectional search [86]. Lewis and Torczon [53] extended the PS methods of Torczon [86] to solve bound constrained nonlinear problems, and Lewis and Torczon [54] extended it for linearly constrained problems. Audet and Dennis [5] gave a simpler analysis of the PS methods of Torczon [86] and Lewis and Torczon [53]. Audet and Dennis [4] presented a PS method for general constrained nonlinear optimization based on filter methods for step acceptance. The unifying theme that distinguishes the PS algorithms from other direct search methods is that each of them performs a search using a pattern of points and a step length control parameter that are independent of the objective function f. These methods proceed by conducting a series of exploratory moves about a current iterate before declaring a new iterate and updating the associated information. The moves can be viewed as sampling the function about the current iterate x_k in a well-defined fashion in search of a new iterate $x_{k+1} = x_k + \Delta_k$ with a better function value. The individual PS methods are distinguished in part by the manner in which these moves are conducted.

Other methods of design optimality (D-, A-, G-, and Q- optimality) have been reviewed in [60]. These methods provide criteria for comparison and evaluation of response surface methodology designs as well as computer-generated designs. Chen and Tsai [15] have mentioned that the response surface method requires considerable time

and effort on the part of users. Therefore, PS methods are considered in the present work. A few relevant PS methods are described in detail in the following paragraphs.

3.2.1 Coordinate Search

The method of coordinate search (CS) is the simplest and most obvious of all the PS methods [86]. It has other names as well like alternating directions, alternating variable search, axial relaxation, and local variation. Figure 2 shows all possible trial points defined by the pattern for CS when $n = 2$ (i.e., 2-dimension), for a step length Δ_k. To explain it, consider all possible outcomes for a minimization problem for a single iteration as shown in Figure 3. The current iterate is marked as x_k. The $x^i{}_k$'s denote trial points considered during the course of the iteration. The next iterate is marked as x_{k+1}. Solid lines indicate successful intermediate steps taken during the course of the exploratory moves while dash lines indicate points at which the function was evaluated but that did not produce further decrease in the value of the objective function f. Thus, in the first scenario shown, an intermediate step from x_k to $x^1{}_k$ (+ve x direction) resulted in a decrease in f, so another step from $x^1{}_k$ to x_{k+1} (+ve y direction) was tried and led to a further decrease. The iteration would then terminate with new point x_{k+1}. In the second scenario, after the successful intermediate step from x_k to $x^1{}_k$, a step from $x^1{}_k$ to $x^2{}_k$ (+ve y direction) did not result in any decrease, so a step in the –ve y direction was tried and yielded a reduced f value. Hence, this would be the new iterate x_{k+1}. In the worst case (the last scenario), $2n$ trial points were evaluated ($x^1{}_k$, $x^{1'}{}_k$, $x^2{}_k$, and $x^{2'}{}_k$) without producing decrease in the f value at

x_k. In such case, $x_{k+1} = x_k$ and the step size must be reduced for the next iteration. The search is terminated when the step size becomes sufficiently small. However, in the present work, a bad move (choosing the least bad among $x^1{}_k$, $x^{1'}{}_k$, $x^2{}_k$, and $x^{2'}{}_k$) is made in order to escape from a local optimum instead of reducing the step size. The search is terminated when bad moves exceed the allowed limit.

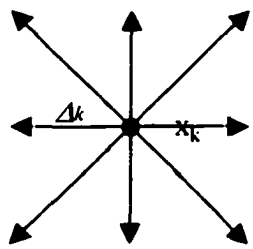


Figure 2: The pattern for CS in $\mathbf{R}^2$ with a step length Δ_k [86].

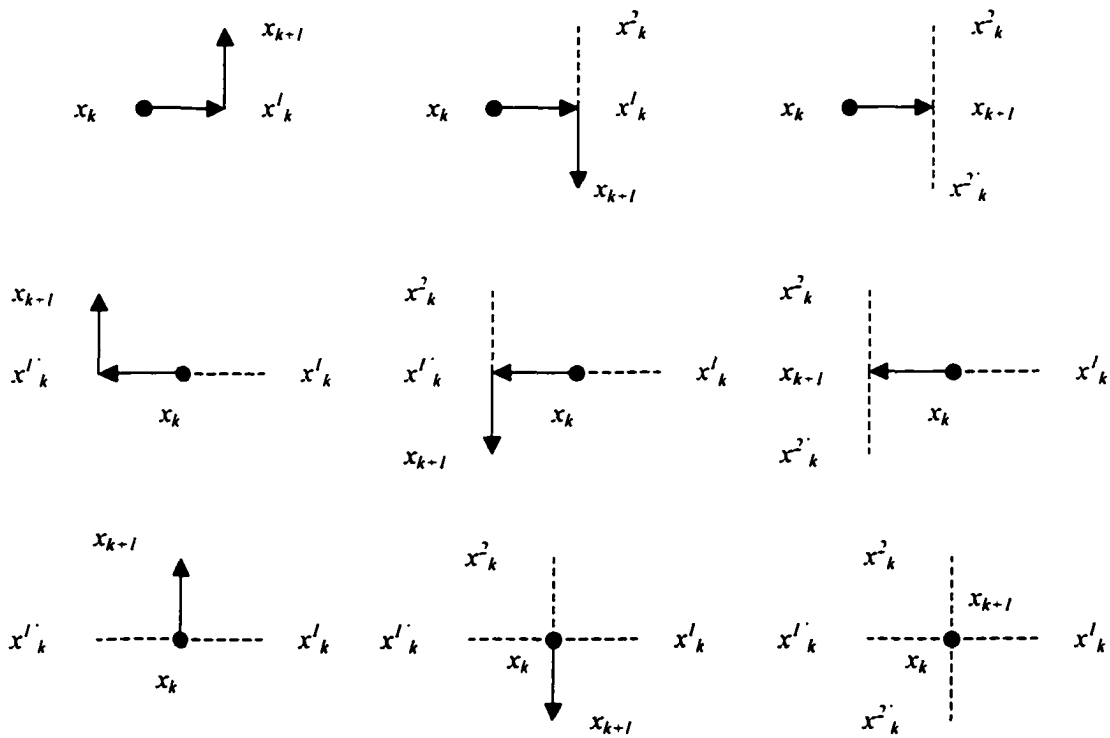


Figure 3: All possible subsets of the steps for CS in $\mathbf{R}^2$ [86].

3.2.2 Hooke-Jeeves Pattern Search

The PS method of Hooke-Jeeves (HJ) is a variant of CS that incorporates a pattern step to accelerate the progress of the algorithm by exploiting information gained from the search during previous successful iterations [86]. To explain this, consider an example of minimization problem as shown in Figure 4. Given x_{k-1} and x_k, for $k > 0$ and $x_k \neq x_{k-1}$, the algorithm takes a step $(x_k - x_{k-1})$ from current iterate x_k in the pattern direction. The function is evaluated at the trial step and this step is accepted temporarily, even if $f(x_k + (x_k - x_{k-1})) \geq f(x_k)$. The HJ procedure then proceeds to conduct CS about the temporary iterate $x_k + (x_k - x_{k-1})$ and the exploratory moves are same as shown in Figure 3. If the result of this exploratory move is a better point than that of x_k, then this point is accepted as new iterate x_{k+1}. If not, i.e., $f((x_k + (x_k - x_{k-1})) + \Delta_k) \geq f(x_k)$, then the pattern step is deemed unsuccessful and the method reduces to CS about x_k, where an exploratory search is undertaken to find a new pattern. Eventually a situation is reached when even this exploratory search fails. In this case the step size is reduced by some factor and the exploration resumes. The search is terminated when the step size becomes sufficiently small. It should be noted that when $k = 0$, the search starts with an exploratory move about x_0.

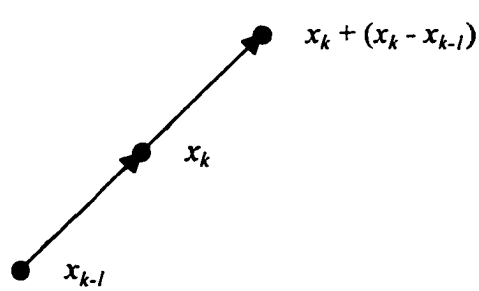


Figure 4: The pattern step in $\mathbf{R}^2$, given $x_k \neq x_{k-1}$, $k > 0$ [86].

Reklaitis et al [70] advised to modify the basic HJ method in order to exploit pattern moves. That is, when a pattern step is a success, then completely exploit the direction by conducting a line search along the pattern direction, or at least trying pattern steps of increasing magnitude. This can significantly accelerate the convergence of the method. In the present work, the modified HJ pattern search is used. That is, if $f(x_k + (x_k - x_{k-1})) \leq f(x_k)$, then it is accepted as new iterate x_{k+1} without conducting any further CS. If $f(x_k + (x_k - x_{k-1})) \geq f(x_k)$, then the pattern step is deemed unsuccessful and the method reduces to CS about x_k, where an exploratory search is undertaken to find a new pattern. As mentioned in CS, if exploratory moves fail to yield a better f value, then a bad move is made instead of reducing the step size. The search is terminated when bad moves exceed the allowed limit.

3.2.3 Tabu Search

Tabu search (TS) is a meta-heuristic intelligent problem solving technique, which guides a local heuristic search procedure to explore the solution space beyond local optimality [30, 31, 66]. The underlying idea is to forbid some search directions (moves) at a present iteration in order to avoid cycling, but to be able to escape from a local optimal point. In doing so, the method incorporates adaptive memory (remembers key elements of the path followed) and responsive exploration (makes strategic choices along the way). The memory structures work with respect to four major dimensions consisting of recency, frequency, quality, and influence. In a sense, quality is a special case of influence. Recency is associated with short-term memory, while the other three are

related to intermediate-term and long-term memory for intensification and diversification. The memory is both explicit and attributive. Explicit memory records complete solutions, typically consisting of elite solutions visited. This helps expand the local search, or in some cases, avoid repetition. The other type of memory keeps information about solution attributes that change in moving from one solution to another. This assists to reduce the local search making certain moves tabu (forbidden). Tabu status of such moves can be overridden by aspiration criteria. A special type of memory is created employing hash functions [31]. This may be considered as a semi-explicit memory, which can be used as an alternative to attributive memory.

To elaborate the above idea, consider a problem of finding an optimal tree (a sub graph without cycles consisting of edges by joining nodes 1 to 7), which has a complex nonlinear objective function [31]. A move needed for changing one tree into another is defined as to drop an edge and add another, so that the result remains a tree. Assume the move applied at iteration k tree with edges (1, 2), (1, 3), (1, 4), (2, 5), (3, 6), and (4, 7) to produce the tree of iteration k+1 consists of dropping the edge (1,3) and adding the edge (4, 6). Hence, for iteration k, we can consider $(1, 3)_{in}$ and $(4, 6)_{out}$ as the two different solution attributes. Since these are the attributes that change as a result of the move, they are tabu-active, and will be used to define the tabu status of moves at future iterations. A move is defined to be tabu if any of its attributes is tabu-active. In a recency-based memory, $(1, 3)_{in}$ can be given a tabu tenure of, for example 3, meaning edge (1, 3) is prevented from being added back to the current tree for 3 iterations. Thus, the earliest that edge (1, 3) can belong to the current tree will be iteration k+4. Similarly, $(4, 6)_{out}$ can be specified tabu-active for 1 iteration., i.e., edge (4, 6) is prevented from being removed

from the current tree for this duration. These conditions seek to avoid 'reversing' particular changes created by the move at iteration k. This example illustrates the concept of multiple tabu lists, each for a particular type of attribute [34]. It is to be noted that the tabu tenure of an attribute is chosen by experimentation or heuristic for a particular problem size, and it depends on the restrictiveness of the associated tabu condition. In this example, making $(1, 3)_{in}$ tabu-active is much less restrictive than making $(4, 6)_{out}$ tabu-active. That's why a larger tenure is given to the former. Glover [33] introduced new dynamic strategies for managing tabu lists.

As mentioned before, explicit memory can be employed to lead the search from the current solution to an elite solution, making the search to visit solutions that are difficult to reach when guided solely by changes in the objective function value. For example, consider the solution at iteration $k+1$ with edges (1, 2), (1, 4), (2, 5), (3, 6), (4, 6), and (4, 7), and assume that a previously identified elite tree with edges (1, 2), (2, 5), (3, 4), (3, 6), (4, 5), and (6, 7) has been stored in explicit memory. The primary goal of the search can now be made to find a path between the current solution and the stored elite solution. This exploration allows the search to perform moves that may be unattractive by the objective function evaluation but which may be necessary to reach better regions in the solution space. Therefore, one possibility is to forbid moves that drop edges that are part of both the current tree and the target elite tree. The first iteration of the search stage will thus forbid edges (1, 2), (2, 5) and (3, 6) from being removed. This use of explicit memory is an illustration of the path re-linking process. It is clear from the example that explicit memory and attributive memory are complementary.

In many applications, short-term (recency-based) memory, has itself resulted in high quality solutions; but in general to make TS significantly stronger, intermediate and long-term memory (frequency-based) are incorporated to intensify and diversify the search [30-34]. Accordingly a penalty/incentive function or probability function is devised to assist the search. During an intensification stage, the search focuses on examining neighbors of elite solutions, which are recorded by explicit memory. This may also initiate a return to attractive regions to search them more thoroughly. Intensification strategy is more closely carried out by intermediate-term memory. A diversification stage encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before. Diversification of the search is more closely operated by long-term memory. Strategic oscillation provides a means to achieve an effective interplay between intensification and diversification [30-32]. The process of repeatedly approaching and crossing a critical level or oscillation boundary from different directions creates an oscillatory behavior, which gives the method its name. A useful integration of intensification and diversification strategies occurs in the approach called path re-linking [30-31]. This approach generates new solutions by exploring trajectories that connect elite solutions starting from an initiating solution and reaching toward the other solutions, called guiding solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

Aspiration criteria are introduced in TS to determine when tabu status of a move can be overridden. A flow chart showing connections between candidate lists, tabu status, and aspiration criteria has been presented in [31, 34]. Glover and Laguna [30]

have discussed the four types of aspiration criteria. If all moves currently available lead to solutions that are tabu, the penalties result in choosing a 'least tabu' solution based on aspiration by default. If a tabu move gives a solution better than the best obtained so far, aspiration by objective permits it to be a candidate for selection. If a tabu attribute results in improving direction, this makes it tabu-inactive from aspiration by search direction. The fourth is aspiration by influence (degree of change). A low influence move is tolerated until the gain appears to be negligible. A high influence move is performed to break away from the local optimality. Like tabu tenure, aspiration criteria can also be given tenures [34].

Applications of tabu search in different optimization problems have been described in Glover and Laguna [31] and other references of both the authors. The areas include production planning & scheduling, facility layout & line balancing in manufacturing, telecommunication networks, transportation, financial analysis, etc. Skorin-Kapov [82] adapted tabu search to the quadratic assignment problem (QAP). QAP is to find an optimal assignment of n objects to n locations to minimize the cumulative product of flows between every two objects and distances between every two locations. The function to be optimized is quadratic and non-convex (i.e., there exist a number of local optima) and a feasible set is a set of permutations. It can be noted that in a special restricted form, a QAP may be reduced to a traveling salesman problem (TSP). The tabu search method was implemented on QAPs in a flexible form, which allowed the user to interact and change the parameters (tabu list size, iteration limit, a search diversification parameter, and number of new starting solutions i.e. restart). The results suggested that good tabu sizes increase with dimension of the problem. The algorithm

appeared to be a very efficient method for QAPs. In comparison with simulated annealing, TS gave superior solution quality. Badiru [9] applied a TS based algorithm to find the maximum and minimum thickness of a flat plate measured by CMM. Kolahan and Liang [49] used a TS approach to minimize the total processing cost for hole-making operations. The total cost involved tooling cost, machining cost, non-productive tool traveling cost, and tool switching cost.

3.2.4 Hybrid Search

It is also possible to use a heuristic algorithm, which is hybrid of more than one search method. For example, Al-Sultan and Al-Fawzan [2] developed a combination of TS and HJ algorithm to solve unconstrained optimization problems. The objective function of such problems may not necessarily be convex and therefore may possess many local minima in the region of interest.

Manufacturing processes generate predictable patterns of errors on parts as mentioned in Chapter 2. If the error profiles can be suitably identified, it will help determine approximate maximum and minimum locations on the surface. Then, optimization search methods can be applied in the neighborhood of such locations to further improve the solutions. Thus, the surface error quantification will reduce the search time (i.e., sampling time) and increase the accuracy. Based on this idea, a methodology for adaptive sampling in coordinate metrology is developed and discussed in the next chapter.

CHAPTER 4

OBJECTIVE AND OVERVIEW OF METHODOLOGY

4.1 Objective

This research deals with the application of manufacturing surface profiles and optimization search methods in sample point selection for form tolerance estimation using CMM. The objective is to develop a unique and sturdy adaptive sampling procedure to reduce sample size while maintaining accuracy of the zone estimation during efficient part feature verification. Straightness and flatness evaluations are illustrated as examples in form tolerance verification.

4.2 Overview of Methodology

Regardless of the form tolerance evaluation method, it is obvious that the location and number of data points affect the time of measurement and accuracy of the result. The present work proposes search-based sampling and uses the linear least squares (LS) method for tolerance estimation. The search is guided by the geometry of the object surface and the manufacturing process used to produce it. That is, the surface profile is suitably quantified to simplify the selection of number and locations for initial data points. Parts have been manufactured by end milling and face milling. The surface pattern generated by each process is modeled as per literature. The predicted patterns are experimentally verified in Chapter 5.

Search-based sampling starts with initial points dictated by the surface pattern. For the initial sample, a fit feature and the corresponding deviation e_i of each point are obtained. For straightness, if the assessment line is $z = z_0 + l_0 x$, then the deviation can be expressed in the form of $e_i = z_i - (z_0 + l_0 x_i)$, and the LS solutions [59, 77, 87] are given as $l_0 = \left(N\sum x_i z_i - \sum x_i \sum z_i\right) / \left[N\sum x_i^2 - \left(\sum x_i\right)^2\right]$, $z_0 = \left(\sum z_i - l_0 \sum x_i\right)/N$, and tolerance $h_t = \left(\left|e_{\max(+)}\right| + \left|e_{\max(-)}\right|\right)/\sqrt{1+l_0^2}$. For flatness, if the assessment plane is represented by $z = z_0 + l_0 x + m_0 y$ and $e_i = z_i - (z_0 + l_0 x_i + m_0 y_i)$, then the LS solutions [77, 87] are given as

$$term = \left(N\sum x_i z_i - \sum x_i \sum z_i\right)\left(N\sum x_i y_i - \sum x_i \sum y_i\right) / \left\{N\sum x_i^2 - \left(\sum x_i\right)^2\right.,$$

$$m_0 = \frac{\left[term + \sum y_i \sum z_i - N\sum y_i z_i\right]}{\left[\left\{\left(\sum x_i \sum y_i - N\sum x_i y_i\right)^2 / \left[N\sum x_i^2 - \left(\sum x_i\right)^2\right]\right\} + \left(\sum y_i\right)^2 - N\sum y_i^2\right]},$$

$l_0 = [m_0 (\Sigma x_i \Sigma y_i - N\Sigma x_i y_i) + N\Sigma x_i z_i - \Sigma x_i \Sigma z_i] / [N \Sigma x_i^2 - (\Sigma x_i)^2]$,

$z_0 = (\Sigma z_i - l_0 \Sigma x_i - m_0 \Sigma y_i) / N$, and tolerance $h_t = \left(\left|e_{\max(+)}\right| + \left|e_{\max(-)}\right|\right)/\sqrt{1+l_0^2+m_0^2}$.

Here, x_i, y_i, z_i are the cartesian coordinates of ith data point, z_0 the intercept, l_0, m_0 are the slopes, N the total number of data points, and $e_{max(+)}$ and $e_{max(-)}$ the maximum deviation in the +ve and –ve direction from the assessment feature respectively.

Then a search method is used to pick up next point until an optimum e_{max} is reached. The search is performed in both the +ve and –ve directions from the fit surface. Thus a trade-off is attempted between the time (number of points measured) and accuracy. With the two solution points so obtained and the initial points, the final form tolerance is determined. As the measurements are taken intelligently, number of the

sample points is much smaller than would be expected for achieving the desired accuracy level. In any iteration, more than one point may be evaluated, but only one point results in an actual move depending on the search technique and that point is considered as a sampled point. The initial set is accounted for in the total number of sampled points.

For straightness, region-elimination (RE) search [70] is employed to choose additional data points. Application of the RE algorithm is discussed with a flow chart in Chapter 6. It is to be noted that the objective is to reduce the sample size while demonstrating reasonable accuracy in the tolerance evaluation. For demonstration purposes, only three iteration with the intervals of Δ, $\Delta/2$, and $\Delta/4$ have been used ($\Delta = 4 \times$ step size). Effect of step size is studied in Chapter 7 on the design of experiments. The algorithm looking for an optimum e_{max} starts from a point in the initial sample that has got the maximum deviation value in the direction of interest. If the surface pattern and the initial sample predict the possibility of more than one optimum, then the search has to be performed in the neighborhood of all such points. Then from all the optima, the point yielding the best e_{max} is selected as an optimum solution. For example, in case of an end-milled surface, the pattern (Figure 22) shows one optimum in the +ve direction from the initial fit line about the center of the edge, but two optima in the –ve direction around the ends. In case of a face-milled surface (Figure 23), there are two optima in either direction.

For flatness, two pattern search methods, tabu search [31] and a hybrid search are applied to sample data points outside the initial set. The hybrid search (HS) developed is a combination of coordinate search [86], Hooke-Jeeves pattern search [70, 86] and tabu

search (TS). The objective is to estimate the form tolerance with a reduced number of points. Therefore, instead of full-scale TS or HS with large number of allowed bad moves and re-starts to find a very high quality solution, approximate TS and HS which can yield a good solution after only a few iterations are used. More details of the algorithms are given in Chapter 6. A tabu size of one is used. The algorithm starts from a point that has got the maximum deviation among the initial data points in the –ve or +ve direction depending which direction a solution is being sought. The search is stopped if the bad moves (BM) exceed the maximum bad moves allowed or if the iterations exceed the maximum iterations allowed. A constant number of allowed iterations of 35 as used in [9], is selected for both the algorithms. The algorithms are tested for different number of allowed bad moves as well as with no re-start (RS0) and one re-start (RS1). This has been elaborated in Chapter 7 on the experimental design. If a search needs to be re-started (e.g., in case of RS1), then the re-start would originate from a point that has got the second maximum deviation in the initial set, if it is not covered in the previous attempt. Results of both algorithms (TS and HS) are compared in terms of the accuracy of flatness tolerance and the sample size (which also includes the initial set) in Chapter 8.

CHAPTER 5

MANUFACTURING SURFACE PATTERN

The search methods require prior knowledge of approximate locations of form errors induced by manufacturing processes to guide the movement the CMM probe. Two processes: end (peripheral) milling and face milling are considered for which the surface error models are given in [48] and [35] respectively. Constraints on the manufacturing processes, tools and work-holding are placed to ensure that the model assumptions are not violated. Machining experiments conducted to verify the models and the surface profiles obtained are discussed in this chapter.

5.1 Pattern for Flatness

5.1.1 End Milling

End milling process is extensively used in the aerospace industry for a variety of roughing and finishing operations on aircraft structural components such as bulkheads and wing sections. An analysis of surface error in end milling requires an understanding of the surface generation mechanism. Kline et al. [48] have mentioned that a milled surface generated by climb or down milling shows a series of tooth marks or 'troughs' approximately parallel to the axial depth of cut produced by the cutter teeth as they slide across the surface. As the cutter rotates, a tooth first begins to generate the finished surface at the bottom of cut. This point of contact between the cutter and the finished surface slides up the workpiece as the cutter rotates due to the helix angle. Also, as the cutter rotates, the magnitude of the cutting forces and the distribution of the forces on the

cutter vary. In peripheral down milling, the cutting forces generally tend to separate the cutter and workpiece as shown in Figure 5. The surface dimensional errors increase with the cutter feed rate or radial depth of cut [102]. To fully analyze the surface error problem, models for the cutting force system, deflection of the cutter, and deflection of the workpiece are required [48].

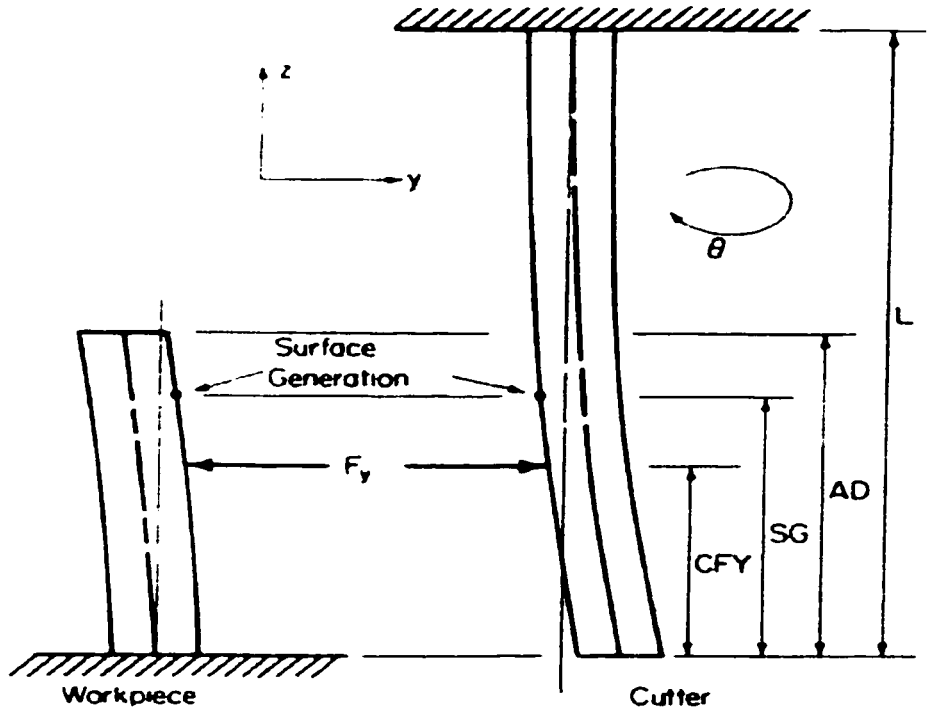


Figure 5: Cutter and workpiece deflection in end milling (source: [48]).

Prediction of the Cutting Force System: A computer-based mechanistic model of the force system in end milling has been developed in [20]. The model has been called a mechanistic model because the chip load or thickness and cutting forces are computed based on the cutter geometry and cutting conditions. The model equations are given as

$$F_T = K_T \, . \, dZ \, . \, t_C \qquad (1)$$

$$F_R = K_R \, . \, F_T \qquad (2)$$

where, F_T is the elemental tangential force, F_R the elemental radial force, dZ the force element width, t_C the chip load, and K_T & K_R the empirical constants. The empirical constants may be estimated from average x and y force data measured over a few cutter revolutions and used with the model to predict instantaneous force system characteristics including the force distributions along the axis of the cutter, the total forces on the cutter, and the force centers. The force distributions along the axis of the cutter are obtained by resolving the elemental forces acting along the flutes into the x (feed) and y (perpendicular to feed) directions. The force center is the center of mass or center of force of a force distribution, i.e., it is the point of application of x or y force (point force) along the axis of the cutter necessary to produce the same bending moment about the tool holder as the bending moment produced by the distributed force loading. It is seen that the y force and force center are key elements in the prediction of surface error, because the y force tends to separate the cutter and workpiece [48]. This cutting force model assumes a rigid machining system. The model is used to compute cutter and workpiece deflections, but the feedback effect of these deflections on the chip load and cut geometry is not included in the force calculations. However, it is found in [48] that this assumption is not critical for the accurate prediction of surface error profiles.

Prediction of End Mill Deflection: A relatively simple but effective model has been used in [48] for predicting the deflection of the cutter. It is assumed that the end mill is a cantilever beam, rigidly supported by the tool holder and that the end mill deflects due to the x and y cutting forces applied to the cutter at the x and y force centers. The model [48] for the y deflection of a slender cantilever beam loaded with a point force is given in Eq. (3).

$$d_Y(Z) = \frac{F_Y}{6EI_Y}[(CFY - Z)^3 - (L - Z)^3 - 3(L - Z)^2(L - CFY)] \qquad (3)$$

where $d_Y(Z)$ is the y cutter deflection at point Z, F_Y is the y cutting force, CFY is the y force center, I_Y is the moment of inertia of the end mill ($Iy = D^4/48$), E is the modulus of elasticity of the end mill, D is the diameter of the cutter, and L is the effective length of the cutter. The deflection of an end mill illustrating these variables is shown in Figure 5.

Prediction of Workpiece Deflection: A thin-walled workpiece, such as that encountered in aerospace applications, can be thought of as a rectangular plate clamped on three edges with the fourth edge free to deflect, and as the workpiece is machined, the thickness of the plate is reduced. The deflection of the plate is computed in [48] with the help of the finite element method applying the y cutting force located at the y force center to the plate.

The above three models are combined to yield surface error predictions. Surface error is defined as the deviation of the finished machined surface in the y direction from the nominal or desired surface, that is, the surface that would be produced by a completely rigid machining system. If the angular position of the cutter where a tooth contacts the finished surface at the free end of the cutter is called zero degree, the surface generation point (SG) or point of contact along the z-axis between the cutter tooth and finished surface is defined [48] as

$$SG = R.\theta / \tan \alpha_{hx} \qquad (4)$$

where R is the radius of the cutter, α_{hx} is the helix angle, and θ is the angular position of the cutter in radians. With the surface generation point defined, the surface generation procedure may now be described, for a particular set of machining conditions, at a particular point along the workpiece.

1. At an angular position of the cutter of zero degree, the cutting forces and force centers are computed from equations (1-2).

2. The surface generation point is determined from Eq. (4).

3. The y cutting force is applied to the cutter at the y force center, and the cutter deflection is estimated at the point of surface generation from Eq. (3).

4. If a flexible workpiece is being machined, the y cutting force is applied to the workpiece at the y force center, and the workpiece deflection is computed at the point of surface generation from the finite element program.

5. The cutter and workpiece deflections in y direction are added to yield the surface error prediction at the surface generation point.

The cutter is rotated, and this sequence of operations is repeated to trace out the surface error profile produced by the y force along the axial depth of cut. Figure 6 shows the cutter deflection, workpiece deflection, and predicted surface error profiles for one of the tests conducted by Kline et al. [48] for end (peripheral) milling. They compared the calculated surface errors against the measured errors for a series of machining experiments without coolant on rigid and flexible 7075-T6 aluminum workpieces in down milling. They found a good match, particularly for the rigid workpieces. Experiments are carried out in this work as well to verify their error model. Figure 7 displays the current workpiece and the machining conditions as given in [48] are detailed later. For those conditions, Kline et al. [48] reported that an angular rotation of the cutter of 176.4 deg is required for a tooth to move from the bottom of cut, $z = 0$, to the top of cut, $z = 50.8$ mm (2 in.). The workpiece deflection (Figure 6) from $z = 0.0$ to 25 mm is small because it is being computed near the supported edge of the workpiece. From 32 to

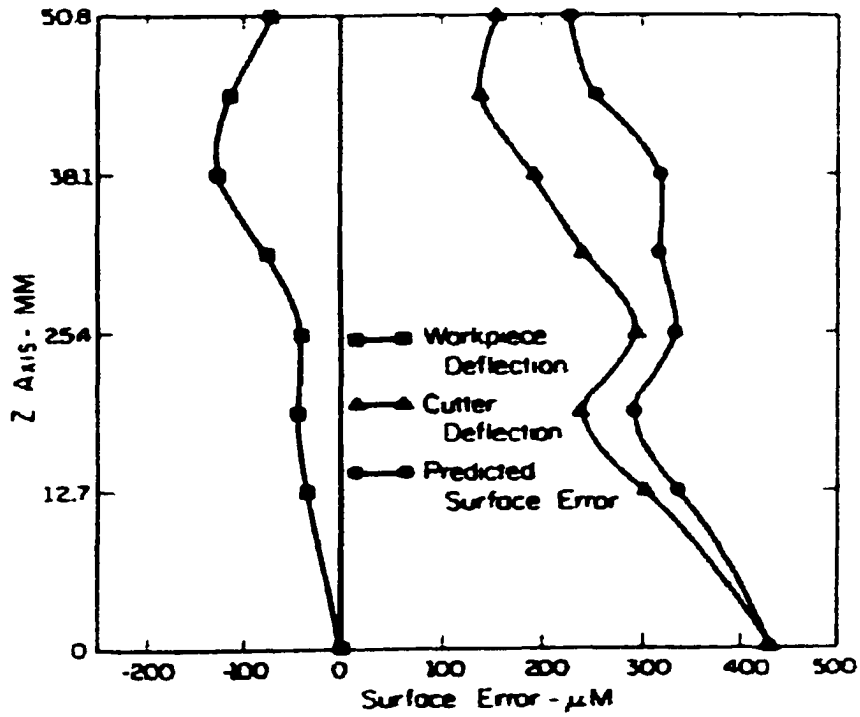


Figure 6: Surface error in $y-z$ plane in end milling (source: [48]).

45 mm, the workpiece deflection increases greatly. This region along the profile corresponds to an angular position of the cutter of 110 to 155 deg, and in this region, the force center is high on the workpiece. With both the surface generation point and force center high, larger deflection occurs. Figure 6 shows a kink or curvature in the cutter deflection profile from 13 to 25 mm. At 13 mm, the force center is high on the cutter and the y force is at a minimum. At 19 mm, the force center shifts lower on the cutter and the force increases giving rise to greater deflection. Moving further up the profile, the cutter deflection decreases because the force center moves higher on the cutter and the deflections are computed nearer the fixed end of the cutter. Theoretically, the error profile of Figure 6 is the profile produced by a single tooth. Assuming for a four-fluted cutter with no runout that the cutting force and force center profiles repeat every 90 deg, successive teeth on the cutter trace out the same profile, with the spacing between

profiles or toothmarks equal to the feed per tooth. For a very small runout level in [48], the effect of runout on the cutting forces and surface accuracy is negligible.

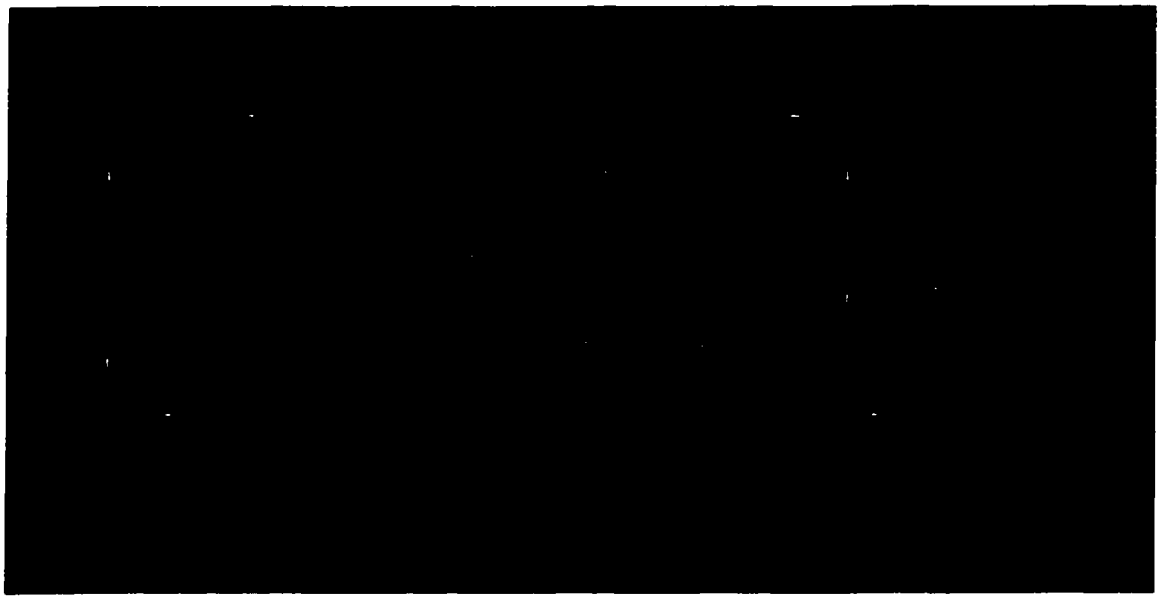


Figure 7: Aluminum workpiece for end milling.

Kline et al [48] have validated the model of Figure 6 measuring the surface errors of the plates used in their experiments with a dial indicator across the length of the workpiece (start to end of cut) and across the width of the workpiece (bottom to top of cut). It can be said that at the bottom of cut, the surface errors are relatively constant across the cut because along this edge, the workpiece is rigid, and the surface errors are produced solely by cutter deflection. Across the top of cut, the surface errors increase from the start to middle of cut, and the errors at the end of cut are slightly larger than the corresponding errors at the start of cut. Measuring across the middle of the plate, the same effect is noted. The surface errors increase from the top to bottom of cut and each profile has a kink/curvature in it near the middle of cut (also reported in [84]). Kline et al [48] measured the profiles approximately midway between the start and end of cut so that the full effect of the workpiece deflection can be studied. Their model for predicting

surface error is limited to cases where the primary components of the error are the cutter and workpiece deflections and they are not too large.

For the present work, 12 plates have been produced using end (peripheral) milling, each with a new cutter, so as to minimize the influence of wear and fracture of used tools. Similar conditions have been used as given in [48]. The plates are numbered from 1 to 12 in the sequence they have been milled. The details are as follows:

Workpiece: 7075-T6 aluminum, 11 × 3 × 0.25 inches, sandwiched between two C-shaped fixtures from the three sides (fixture was made of 1inch thick steel), unsupported workpiece = 8.5 × 2 × 0.25 in. as shown in Figure 7.

Cutter: 4-fluted high-speed steel end mill, diameter = 0.75 in., 30° helix angle, flute length = 3.75 inches.

Machining conditions: no coolant, down milling, radial depth of cut = 0.05 in., axial depth of cut = 2 in., cutter speed = 525 rpm, # of teeth = 4, feed = 0.015 in./tooth = 0.015×525×4 = 31.5 in./min.

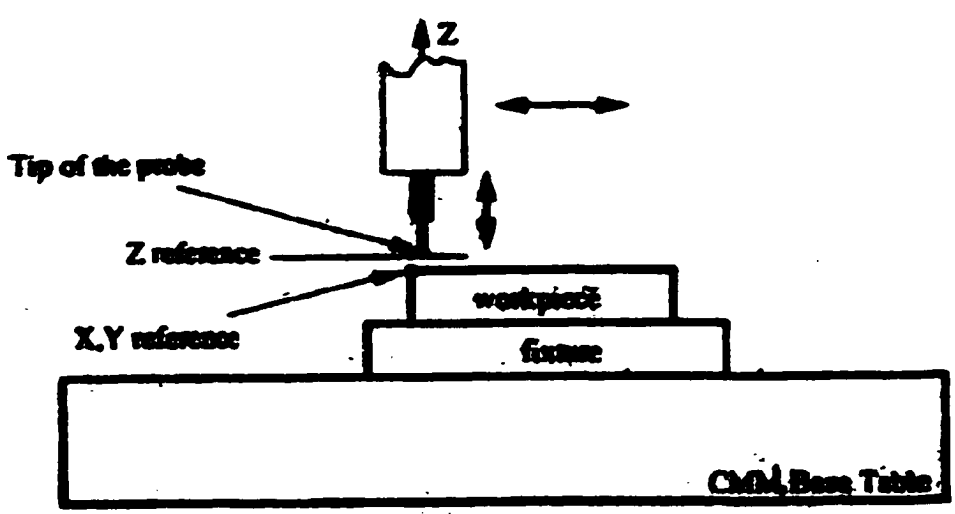


Figure 8: Surface measurement with CMM (source: [35]).

For validation of the above error model, measurements are carried out using a Brown & Sharpe® Reflex™ 343 series CMM, as shown in Figure 8. Note that the workpiece in Figure 7 is shown in vertical (x-z) plane for the milling. But for the measurements, the plate is laid on horizontal CMM table. Hence (x-z) is changed to (x-y). Before the measurements, the CMM coordinate system is initialized. From the total machined area, a surface of 3 in. long in the center (i.e., x = 5 to 8) and 2 in. wide (top to bottom of cut, i.e. y = 5 to 7) is measured to study the surface profile. The CMM probe gives (x, y, z) values for each data point. Now, x = 5 and x = 8 will be referred to as start and end of cut. Four plates, # 10, 5, 7, and 4 are randomly selected and their surface patterns have been shown in Figures 9 - 12. In these figures, the deviation (e) values obtained across the width (y = 5 to 7) are plotted for x values corresponding to the start, center, and end of cut. The profiles obtained agree with that of Kline et al. [48]. It is seen in Figure 6 that the surface errors (i.e., the deviations of the finished surface from the nominal surface) are positive. Since this study is focused on flatness zone, an ideal plane is fit to the points measured from the surface and the deviations from the fit plane are shown in Figures 9 – 12. Therefore, some points have negative values and some have positive.

Looking into Figures 9 - 12, across the length, at the bottom of cut (y = 7), the surface errors are relatively constant (deviations are all positive). Along this edge, the workpiece is rigid, and the surface errors are produced solely by cutter deflection, so constant surface errors are expected. Across the top of cut (y = 5), the deviations are negative at the start, positive at the center, and negative at the end of cut. Kline et al. [48] reported that the errors at the end should be slightly larger than the corresponding errors

at the start of cut, which are true in the present case as well, except in plate #7. Measuring around the center of cut, large deviations in the -ve direction at the start and end of cut and small deviations (-ve or +ve) at the center are noted.

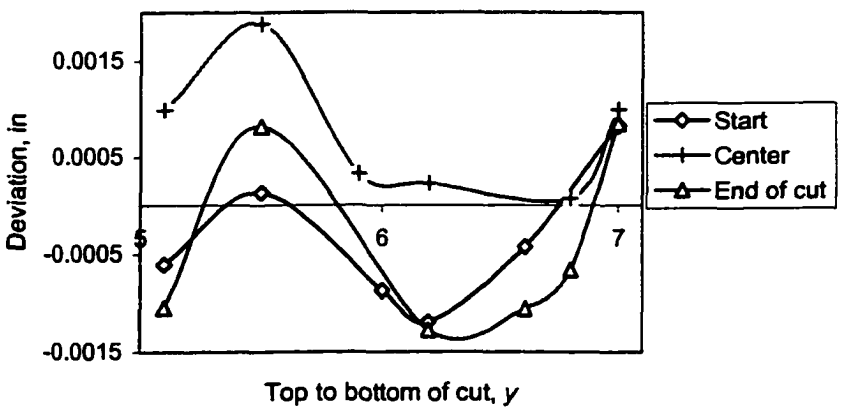


Figure 9: Flatness error profile for end-milled plate #10.

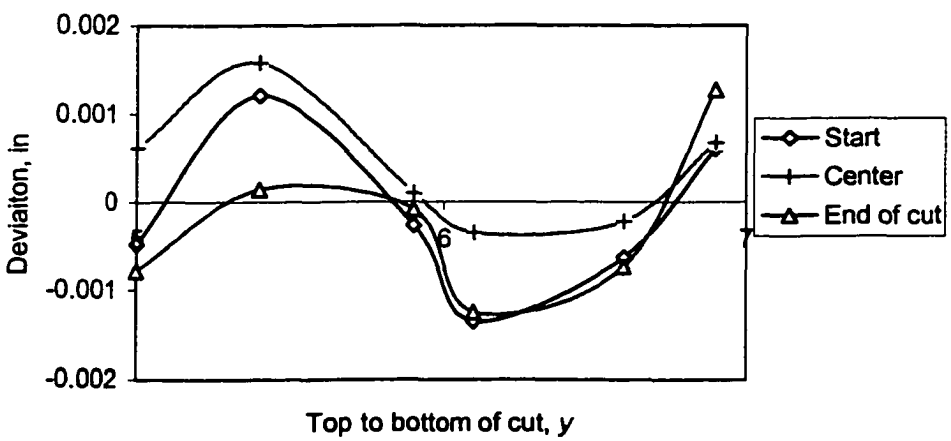


Figure 10: Flatness error profile for end-milled plate #5.

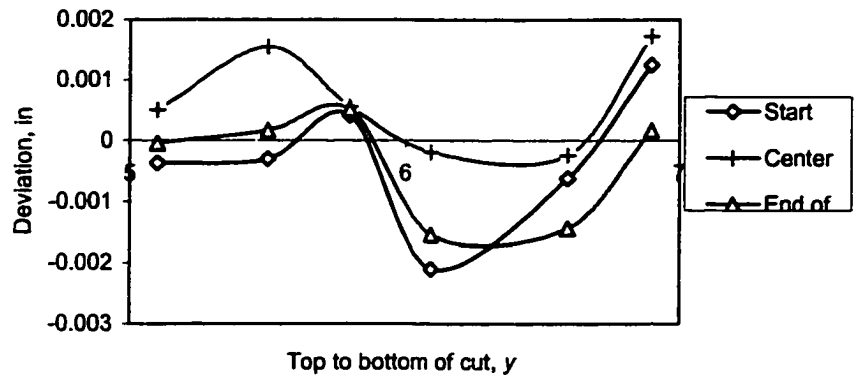


Figure 11: Flatness error profile for end-milled plate #7.

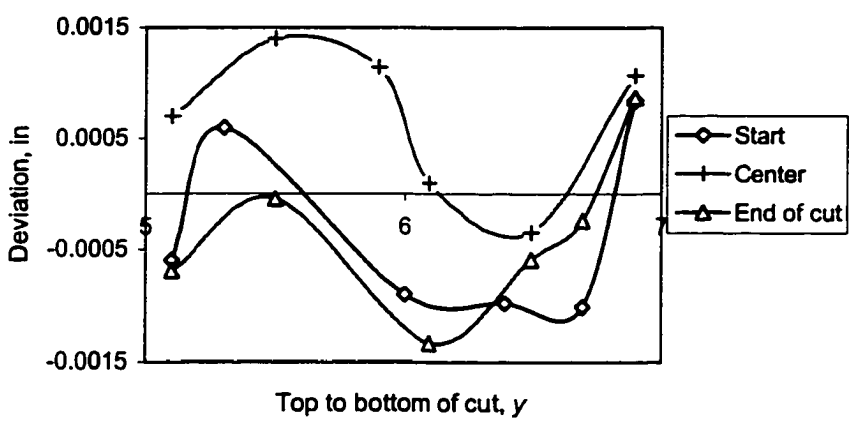


Figure 12: Flatness error profile for end-milled plate #4.

Across the width, the surface errors increase from top to bottom of cut and each profile has a kink/curvature in it near the center of cut as reported in [48, 84]. At the top of cut, the points have negative deviation and then start having positive deviation as we move down. At the middle (i.e., about 1 in. from the top), the points have -ve deviation (a kink). Then again at the bottom of cut, the points have +ve deviation. This means five segments 0 - 0.2, 0.4 - 0.8, 0.95 - 1.15, 1.3 - 1.7, 1.8 - 2 inch from the top (y = 5) are important for initializing the search procedure. In the first segment, select a point at the

end of cut (x = 7.8 - 8); in the second, select a point at the middle (x = 6.4 - 6.6); in the third, at the end; in the fourth, at the start (x = 5 - 5.2); and in the last segment, at the middle. This way, five points can be selected as the initial sample of points to be measured.

5.1.2 Face Milling

In the automotive industry, components such as engine blocks, cylinder heads, and crank castings are face milled. For face milling, Gu et al. [35] have presented a model to predict surface flatness error. The model includes the machining conditions (speed, feed, depth of cut, tool geometry), deflections of the cutter-spindle and workpiece-fixture, static spindle axis tilt, and axially inclined tool path. Models for each of these factors given in [35] are presented here.

Elastic Deformation of the System due to Cutting Forces: The elastic deflection of the machining system consisting of the cutter-spindle and workpiece-fixture assemblies is computed by finite element analysis. This involves constructing a finite element model of the system and creating a flexibility influence coefficient database. The flexibility influence coefficient, $a_{p_1p_2}$, is defined as the static deflection in the axial direction (normal to the workpiece surface) at point p_1 due to a unit force applied at point p_2 in one of the three cartesian directions. The influence coefficients for the workpiece-fixture assembly and/or the cutter-spindle assembly need to be computed only once, provided their configurations do not change. Once the influence coefficient database has been generated the effect of different processing conditions on the form error can be simulated quickly.

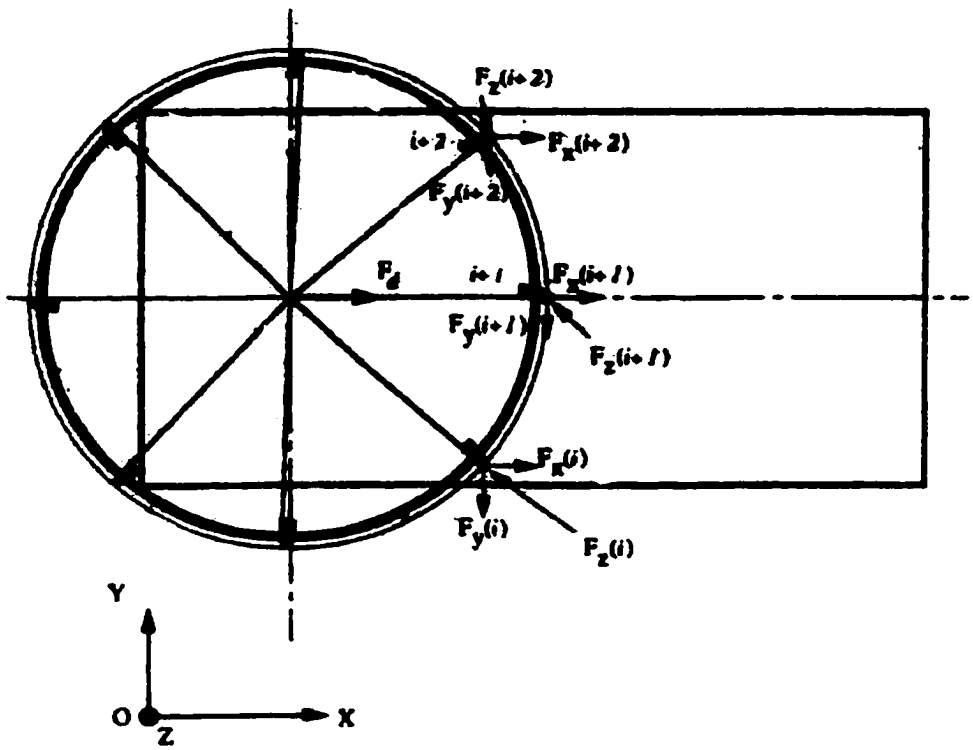


Figure 13: Mutli-tooth face milling schematic showing the instantaneous forces exerted by each tooth on the workpiece (source: [35]).

In face milling more than one insert is usually engaged in cutting at any time instance (see Figure 13). Hence, the relative deflection of the cutter and workpiece at a particular insert location is due to the combined effect of the cutting forces, produced by all the inserts engaged in cutting. In general, the instantaneous insert locations do not coincide with the finite element nodes. In order to compute the elastic deflections at points other than the nodal positions, Gu et al. [35] have developed an equivalent influence coefficient method. The equivalent influence coefficient (nodal or otherwise), $a'_{p_1 p_2}$, is determined from the nodal influence coefficients and the shape functions of the elements enclosing points p_1 and p_2. Mathematically, this can be written as

$$a'_{p_1p_2} = \begin{cases} \sum_{k=1}^{m} h_k(x_{p_1}, y_{p_1}, z_{p_1}) \sum_{l=1}^{n} a_{kl} h_l(x_{p_2}, y_{p_2}, z_{p_2}); p_1 \neq p_2 \\ \sum_{k=1}^{m} a_{kk} h_k(x_{p_1}, y_{p_1}, z_{p_1}); p_1 = p_2 \end{cases} \quad (5)$$

where $h_k(x_{p_1}, y_{p_1}, z_{p_1})$ and $h_l(x_{p_2}, y_{p_2}, z_{p_2})$ are the shape functions of the elements enclosing points p_1 and p_2, respectively; m and n are the number of nodes in the two elements; a_{kl} is the flexibility influence coefficient at node k of the element enclosing p_1 due to the unit force at node l of the element containing p_2.

The instantaneous cutting force components in the three cartesian directions acting on insert i of the cutter are given by

$$\begin{Bmatrix} F_{ix}(t) \\ F_{iy}(t) \\ F_{iz}(t) \end{Bmatrix} = \begin{bmatrix} \cos\theta_i(t), -\sin\theta_i(t), 0 \\ \sin\theta_i(t), \cos\theta_i(t), 0 \\ 0, 0, 1 \end{bmatrix} \begin{Bmatrix} F_c(\theta_i(t))\cos\alpha_v - F_t(\theta_i(t))\cos\gamma_L \sin\alpha_v \\ F_c(\theta_i(t))\sin\alpha_v + F_t(\theta_i(t))\cos\gamma_L \cos\alpha_v \\ F_t(\theta_i(t))\sin\gamma_v \end{Bmatrix} \quad (6)$$

where, $F_c(\theta_i(t))$ and $F_t(\theta_i(t))$ are the instantaneous equivalent orthogonal forces in the cutting and thrust directions acting on the ith insert, respectively; $\theta_i(t)$ is the instantaneous angular position of the ith insert; α_v defines the direction of the relative velocity vector, and γ_L is the effective lead angle of the cutting edge.

The surface flatness error resulting from a change in the axial depth of cut due to elastic deflection of the workpiece-fixture assembly, $SE_{wp}(x_i(t), y_i(t))$, can now be computed as

$$SE_{wp}(x_i(t), y_i(t)) = -\sum_{j=1}^{NT} \sum_{k=x,y,z} a'_{p_i p_j} \delta(j) F_{jk} \quad (7)$$

where, NT is the total number of inserts. The index j denotes the insert number while index k refers to the direction of the cutting force component in the Cartesian coordinate system such that F_{jk} represents the force on the workpiece in the kth direction due to

insert j; $\delta(j)$ is the Kronecker Delta function which defines the insert and workpiece engagement by taking on a value of 1 when the insert is engaged in cutting and 0 otherwise. The negative sign in Eq. (7) implies that when the forces on the workpiece are positive (i.e., the axial force is in the direction of the workpiece surface normal), the resulting surface height of the workpiece is lower than the nominal height due to increased depth of cut.

The surface error arising from elastic deflection of the cutter-spindle assembly, $SE_{sp}(x_i(t), y_i(t))$, is given by

$$SE_{sp}(x_i(t), y_i(t)) = \sum_{j=1}^{NT} \sum_{k=T,R,A} a''_{p_i p_j} \delta(j) F_{jk} \qquad (8)$$

where, F_{jk} represents the cutting force on the jth insert in the kth direction with respect to a local cylindrical coordinate system fixed to the spindle axis; $a''_{p_1 p_2}$ refers to the axial deflection of cutter-spindle assembly at the position of the ith insert tip due to force in the kth direction acting on the jth insert. Equation (8) does not have a negative sign because, when the forces on the cutter are positive, the resulting workpiece surface height is lower than the nominal height due to an increase in the depth of cut. Gu et al. [35] have reported that the effect of the cutter-spindle assembly is negligible, in particular for fly cutting.

Static Spindle Axis Tilt: The spindle axis tilt is described by two parameters, γ_{ts} and β_{ts}, which represent the magnitude and direction of tilt; β_{ts} is measured in the x-y plane in the ccw direction (see Figure 14). The presence of spindle axis tilt results in varying axial depth of cut along the insert path. The surface flatness error due to static spindle axis tilt, $SE_{st}(x_i(t), y_i(t))$, is given by

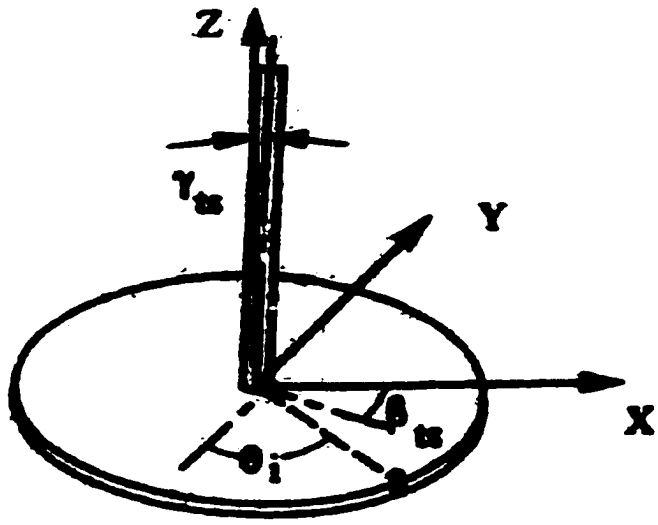


Figure 14: Schematic of spindle axis tilt and effect of tooth's axial position (source: [35]).

$$SE_{st}(x_i(t), y_i(t)) = -R_c(i)\sin\gamma_{ts}\sin[\theta_i(t) - \beta_{ts} - \alpha_f(t)] \qquad (9)$$

where, $R_c(i)$ is the radius of the ith insert measured from the spindle axis, and $\alpha_f(t)$ defines the direction of cutter feed with respect to the x-axis of the global coordinate system. The negative sign in Eq. (9) indicates that the surface height of the workpiece is lower than the nominal height when the spindle tilt angle γ_{ts} is positive.

The surface error equation due to the axially inclined tool feed path is given in [35]. The cumulative surface error at the location of ith insert tip at time t can be obtained by adding the individual surface errors described above. Since the errors due to cutter-spindle deflection and axially inclined cutter path are small, the total surface error may be predicted taking into account the effects of workpiece-fixture deflection and static spindle axis tilt [35].

Gu et al. [35] have verified the surface flatness error model experimentally for fly-cutting as well as multiple-insert cutting. The tool geometry used was as follows: 4 in. diameter, 15° lead angle and -7° radial & axial rake angles. Square carbide inserts of

grade KC 710 with nose radius of 3/64 in. were used. A cast iron workpiece block, 6 × 3 × 1 in. (152 × 76.2 × 25.4 mm), with holes and slots as shown in Figure 15, was used. The middle four holes are fixturing holes for clamping the workpiece. The cutter was fed along the centerline of the workpiece in the +ve x direction. It entered at $y \cong 76$ mm and exited at $y = 0$ in section 1. The surface flatness errors were measured with a CMM in the four sections shown by the dotted lines. The predicted total surface flatness errors due to the effects of the workpiece-fixture system and static spindle axis tilt, and the measured errors for a fly-cutting test are shown in Figure 16. The model prediction agrees very well with the measured surface errors. It has been explained in [35] that the effect of spindle axis tilt is noticed to dominate the total error profile.

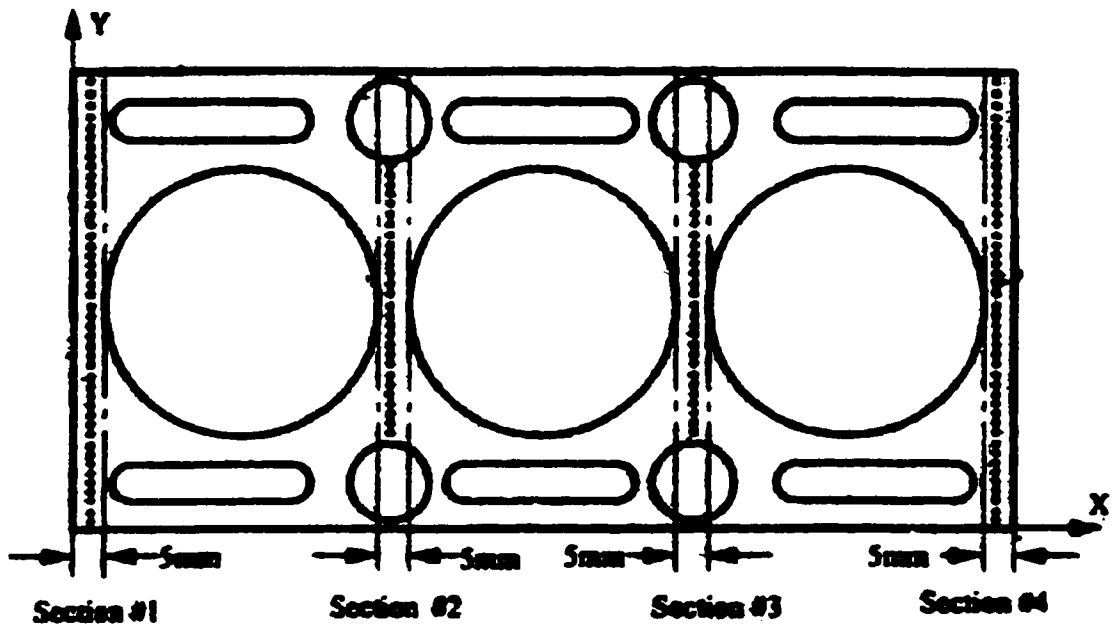


Figure 15: Schematic of workpiece geometry used in [35].

In the present work, tests have been carried out to measure the surface errors and compare with Gu et al.'s model [35]. Twelve plates, each with a new insert, have been manufactured, as done with the end-milled plates. Figure 17 shows the current

workpiece. The middle four holes are fixturing holes as explained with respect to Figure 15. The machining conditions are as follows.

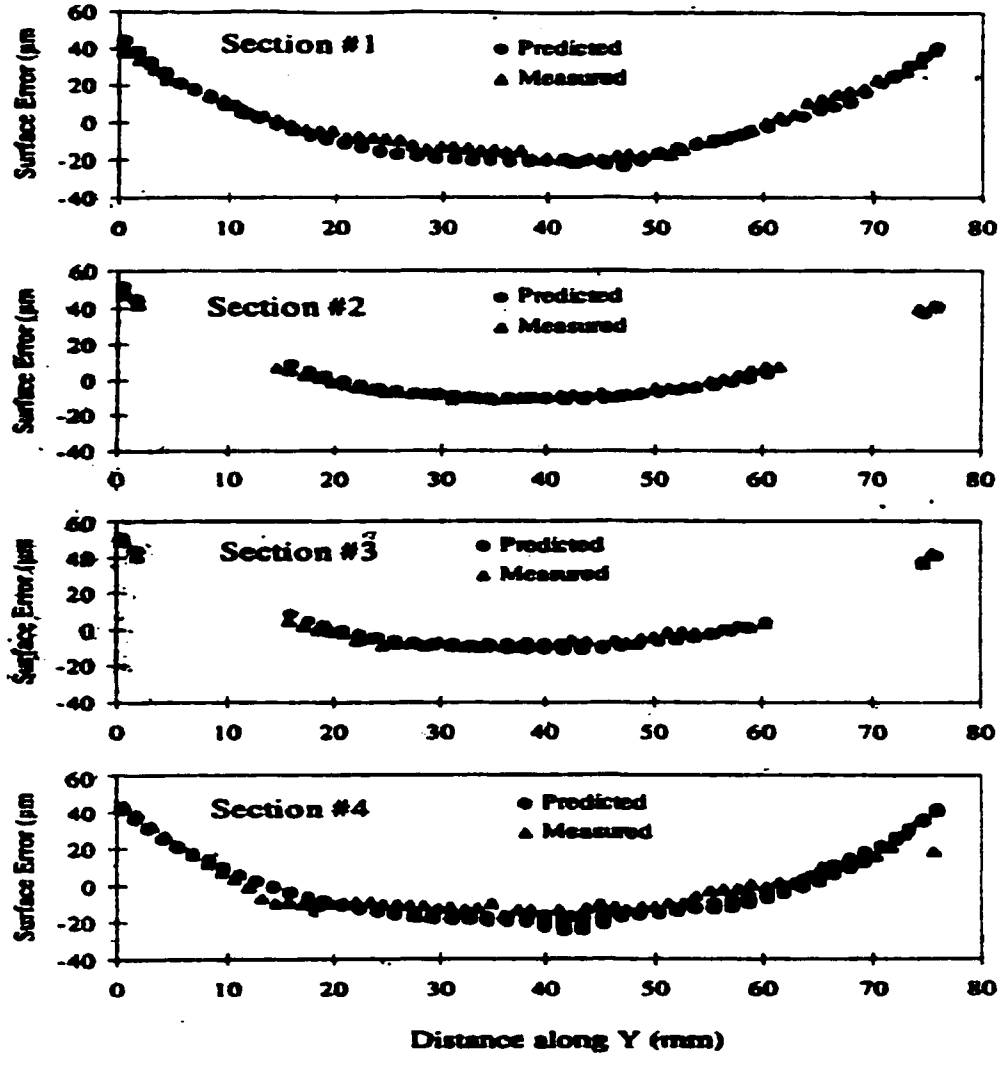


Figure 16: Surface error in face milling (source: [35]).

Workpiece: cast iron, 6 × 2 × 1 in. Note that the width of 2 in. is smaller than 3 in. used in [35].

Tool: Cutter diameter = 3 in., 15° radial rake angle, 17° axial rake angle. Square carbide inserts with PVD coating, 0° lead angle, 11° rake angle, and nose radius of 3/64 in. (Gu

et al. [35] used a cutter of 4 in. diameter with 15° lead angle and -7° radial & axial rake angles and square carbide inserts of grade KC 710 with nose radius = 3/64 in.).

Machining conditions: no coolant, # of insert = 1 (fly-cutting), cutter speed = 1000 rpm clockwise, feed = 0.007 in./tooth = 7 in./min, axial depth of cut = 0.05 inch.

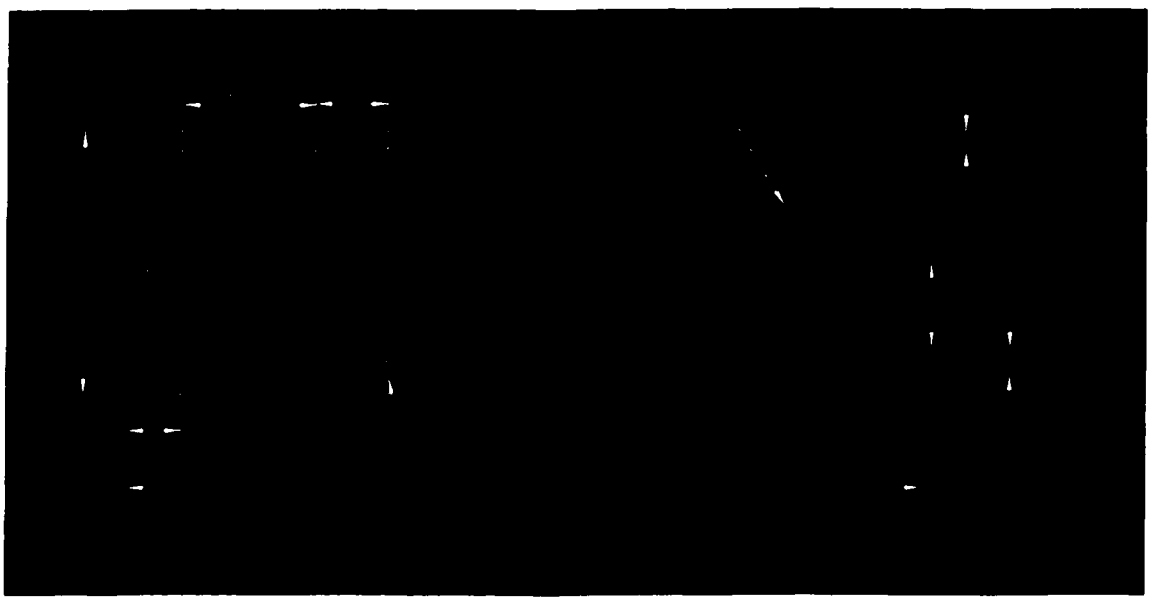


Figure 17: Cast iron workpiece for face milling.

The surface is divided into sections 1 to 4 across the length in the feed direction (Figure 17). Sections 2 and 3 are near the workpiece fixture locations. In section 1, the cutter enters at $y = 3$ and exits at $y = 1$ where as in section 4, the cutter enters at $y = 1$ and leaves at $y = 3$. Four plates # 2, 9, 11, and 5 are randomly selected for the surface measurements using the CMM as described in Section 5.1.1 with Figure 8. The surface profiles measured as shown in Figures 18 - 21 did not agree fully with the profiles given by Gu et al. [35] in Figure 16. This may be due to the following reasons. The present cutter is not identical to the one used in [35], although the inserts are similar. Further, there is no tilt in the spindle used in this work, whereas their spindle was tilted and the effect of spindle axis tilt was found to dominate the total surface error profile.

It can be seen from Figures 18 - 21 that deviations (e) of the points along sections 1 and 4 are negative while that of the points along sections 2 and 3 are positive from the fit plane. This is because the cutter removes more material in sections 1 & 4 than sections 2 & 3. Across the width of sections 1 & 4, absolute deviation is larger in the neighborhood of the cutter entry than the exit, except in plate #2. As expected along sections 2 & 3, there is not much difference across the width, except in plate #9. Based on the profiles (Figures 18 – 21), five initial points can be selected as follows: one each near the cutter entry in section 1 ($y \approx 3$), near the lower hole in section 2 ($y \approx 1.5$), near the upper hole in section 3 ($y \approx 2.5$), near the cutter entry in section 4 ($y \approx 1$), and the last anywhere near the middle ($y \approx 2$) or the cutter exit. Each of these points represents a positive or negative deviation. Extending to the inspection of such parts, the alone developed rule-set will be used instead of the one proposed by gu et al. [35] consistent with the present observations. Note that this is a first attempt at determining initial locations. More research and mathematical models must be developed for sturdier procedures of initial sample point determination.

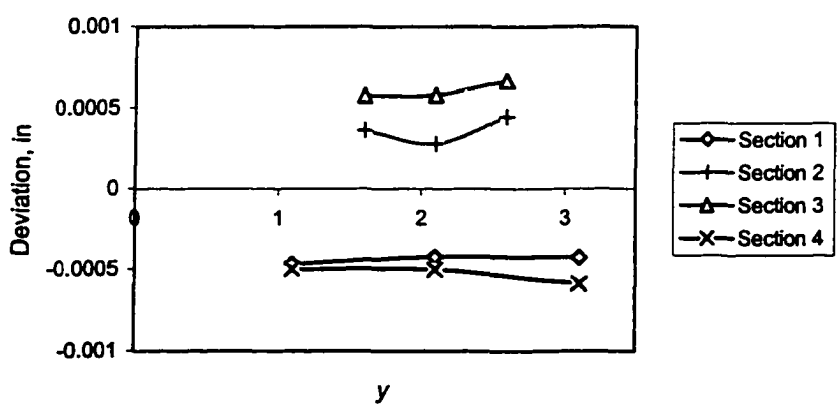


Figure 18: Flatness error profile for face-milled plate #2.

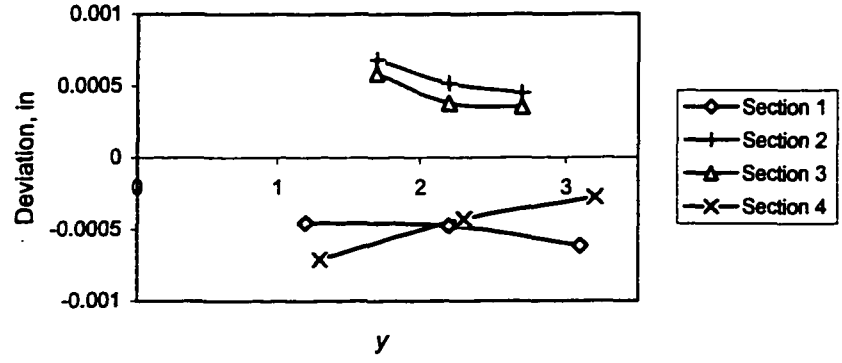


Figure 19: Flatness error profile for face-milled plate #9.

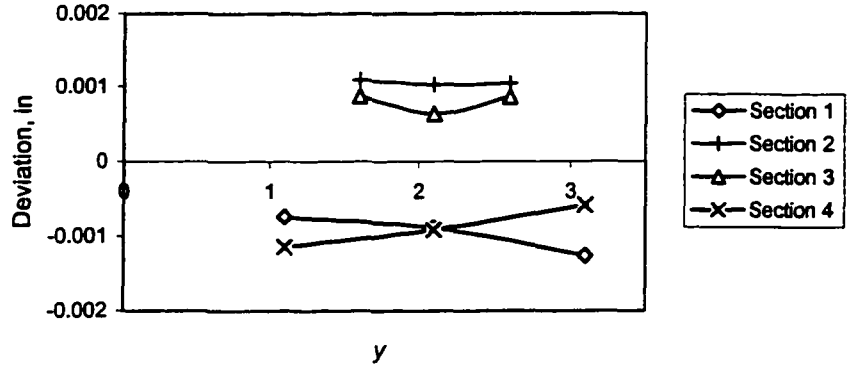


Figure 20: Flatness error profile for face-milled plate #11.

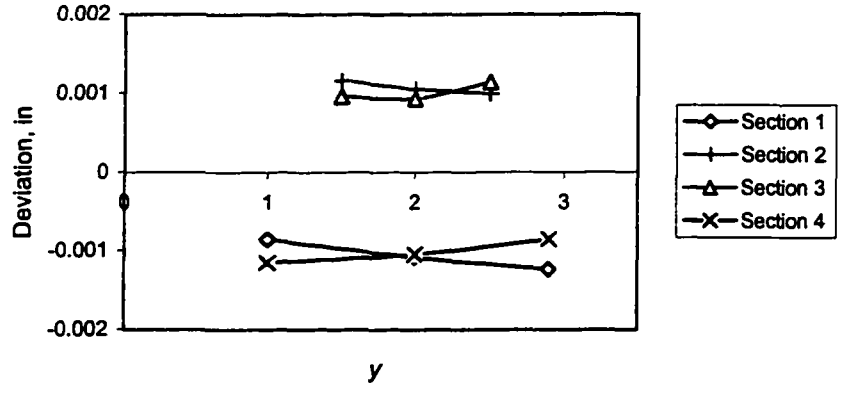


Figure 21: Flatness error profile for face-milled plate #5.

5.2 Pattern for Straightness

5.2.1 End Milling

The unsupported workpiece (see Figure 7) for the end milling is 8.5 × 2 × 0.25 in. The error measurements for flatness have been discussed in Section 5.1.1. For straightness, the plates are measured at the center from the top of cut (i.e., at $y = 6$) across the length of 6 in. (i.e., $x = 4$ to 10) leaving 1.25 in. from both sides of the unsupported workpiece. For a fixed y, (x, z) coordinates of the data points are obtained using the CMM probe. The CMM measurement process has been explained earlier with the help of Figure 8. The value of x increases from the start to end of cut. The surface error pattern is displayed in Figure 22. The deviations (e) are negative at the start, positive at the middle, and then negative at the end of cut. Therefore, three points (one each near the start, middle and end) may act as the initial data set and then the region-elimination search can be performed in their neighborhood.

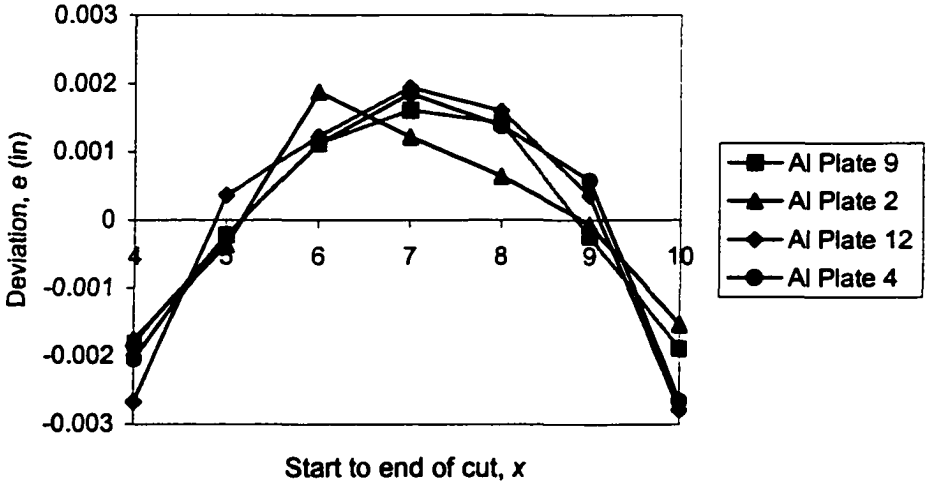


Figure 22: Straightness error profile for end-milled plates.

5.2.2 Face Milling

The error measurements for flatness have been described in Section 5.1.2. For straightness, the cast iron plates (see Figure 17) are measured along the centerline at constant $y = 2$ and Figure 23 shows the error pattern. The value of x increases from the start to end of cut. For the initial set, four points can be chosen, one each from section 1 ($x = 2$ - 2.3), 2 ($x = 3.4$ - 4.4), 3 ($x = 5.6$ - 6.6), and 4 ($x = 7.7$ - 8). From sections 2 & 3, points in the center of the section will be better as initial points.

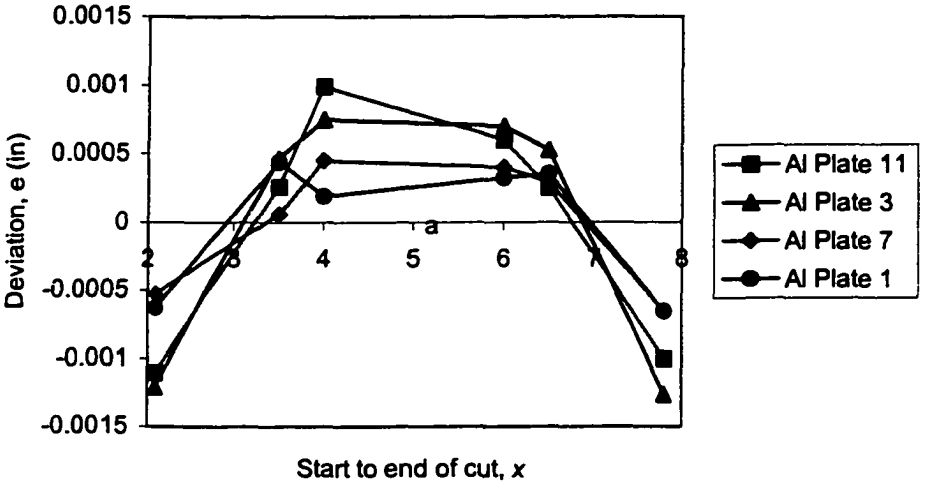


Figure 23: Straightness error profile for face-milled plates.

Now, the number and approximate locations for selecting the initial set of points, on the basis of the measured surface error profiles, for flatness and straightness evaluation of the plates produced from the end and face milling are known. Instead of the actual errors, one can also use the predicted errors from the models given in [48] and [35] for end and face milling respectively for the initial sample. Next points will be sampled on the basis of the search methods described in the next chapter.

5.3 Population Data

After getting the surface profiles of the plates, measurement data are obtained for a population (i.e., very large) sample. The data for straightness for plate # 4, 12, 9, and 2 from end milling at a fixed value of $y = 6$ and x varying from 4 (start) to 10 (end of cut) are presented in Appendix A, Tables A.1.1.1 – A.1.1.4. The data at a fixed value of $y = 2$ for plate # 3, 1, 11, and 7 from face milling are shown in Tables A.1.2.1 - A.1.2.4. It may be noted that each face-milled plate has four sections from start to end of cut: 1 (x = 2 - 2.3), 2 (x = 3.4 - 4.4), 3 (x = 5.6 - 6.6), and 4 (x = 7.7 - 8). For flatness, the data for plate # 10, 5, 7, and 4 from end milling are given in Tables A.2.1.1 – A.2.1.4. Top to bottom of cut corresponds to y = 5 to 7 and start to end of cut is x = 5 to 8. The data for plate # 2, 9, 11, and 5 from face milling are displayed in Tables A.2.2.1 – A.2.2.4. Each plate has four sections from start to end of cut: 1 (x = 2 - 2.3), 2 (x = 3.4 - 4.4), 3 (x = 5.6 - 6.6), and 4 (x = 7.7 - 8). The cutter enters at y = 3 in section 1 and exits at y = 1 whereas in section 4, it enters at y = 1 and leaves at y = 3. In sections 2 & 3, the fixturing holes are located at y = 2.5 and y = 1.5. Corresponding form tolerance obtained using the least squares (LS) method for the population is also shown in these tables. The tolerance obtained from the proposed sampling method will be compared with the tolerance computed for the population in terms of the absolute percentage error to find the estimation accuracy. The Java codes for the LS line and plane are given in Appendix B.

CHAPTER 6

ALGORITHM DEVELOPMENT

As mentioned in Chapter 4, sampling starts with initial points guided by the surface patterns obtained in the previous chapter. We suggest three points in end-milled plates and four points in face-milled plates to determine straightness, and five points for flatness estimation in plates machined by either process. Determination of their locations is explained in Chapter 5. For the initial sample, a fit feature and the corresponding deviation e_i of each point are obtained using the linear least squares (LS) technique. Then a search method is used to pick up next points intelligently until an optimum e_{max} is reached. The search is performed in both the +ve and -ve directions from the fit surface. At any iteration, the algorithm avoids the neighborhood points that are outside the surface. With the two points obtained for optimum e_{max} and the initial points, the final form tolerance is determined.

6.1 Straightness Using Region Elimination Search

For straightness evaluation, region-elimination (RE) search [70] is employed to choose additional sample points. The RE method assumes that the function is unimodal within the interval of interest. For a multimodal function, the interval needs to be refined to obtain multiple optima. Then a global optimum can be determined. A flow chart for the RE algorithm is shown in Figure 24. It is to be noted that the objective is to reduce the sample size while demonstrating reasonable accuracy in the tolerance estimation, and therefore only three iterations with the intervals of Δ, $\Delta/2$, and $\Delta/4$ are used (where $\Delta = 4$

× step size). In the next chapter on experimental design, step sizes of 0.05, 0.1, and 0.2 inch are investigated to find a proper step size.

Search starts from a point that has got the maximum deviation among the initial data set. If the surface pattern and the initial sample predict the possibility of more than one optimum in any direction, then the search has to be performed in the neighborhood of all such points. Then from all the optima, the point yielding the largest e_{max} is selected as an optimum solution. For example, in case of end milling, the pattern (Figure 22) shows one optimum about the center of the edge in the +ve direction but two optima near both ends in the -ve direction from the initial fit line. In case of face milling (Figure 23), there are two optima in both the directions.

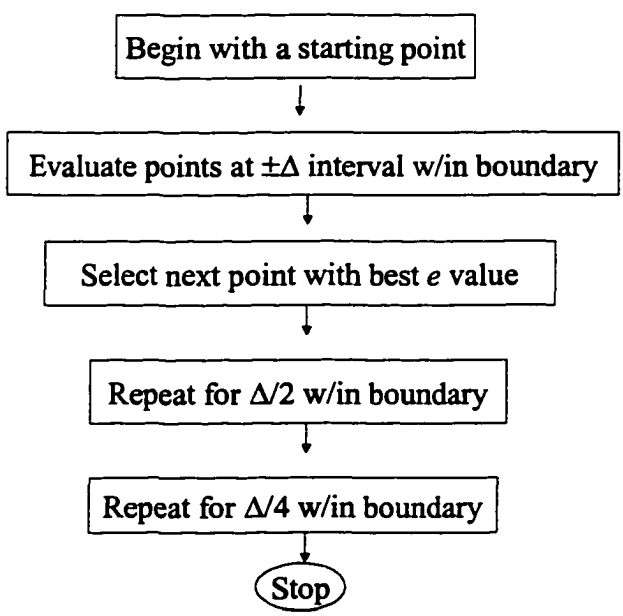


Figure 24: Flowchart of the region elimination algorithm for straightness estimation.

It is to be noted that the current CMM software is not integrated with the RE search method. Therefore, the population data (x, z) for a fixed y, are collected using the CMM probe as outlined in Chapter 5 (see Figure 8 and Section 5.3) to represent the edge

for which straightness is being evaluated. On this data set, the RE algorithm is applied. Java codes for the algorithm to find points corresponding to $e_{max(-)}$ and $e_{max(+)}$ have been written and are given in Appendix B. The algorithm to find a solution point pertaining to $e_{max(-)}$ has the following steps:

Step 1: Initialization Step

Specify x coordinate limits (x_{lbound}, x_{rbound}) to define the boundary points of the edge and step size (i.e., Δs) or interval Δ, where $\Delta = 4 \times \Delta s$.

Specify the fit line parameters for the initial set of points obtained on the basis of the surface error profile.

Read population data points (x, z) of the discretized edge from a file.

Specify the position (index) of a starting point in the discretized space to get (x_{now}, z_{now}). Also, specify the corresponding e_{now}, which is known or can be found out from the equation of the line specified.

Step 2: Search from x_{now}

$x_{left} = x_{now} - 4\Delta s$ $\qquad$ $x_{right} = x_{now} + 4\Delta s$

Check whether the points are within the boundary, i.e., $x_{left} \geq x_{lbound}$ and $x_{right} \leq x_{rbound}$. The points outside the boundary will not be considered.

Calculate e_{left} and e_{right} and compare with e_{now}.

Set the least of the three as e_{now} and the corresponding x as x_{now}.

Step 3:

$x_{left} = x_{now} - 2\Delta s$ $\qquad$ $x_{right} = x_{now} + 2\Delta s$

Check whether the points are within the boundary.

Calculate e_{left} and e_{right} and compare with e_{now}.

Set the least of the three as e_{now} and the corresponding x as x_{now}.

Step 4:

$x_{left} = x_{now} - \Delta s$ $\quad$ $x_{right} = x_{now} + \Delta s$

Check whether the points are within the boundary.

Calculate e_{left} and e_{right} and compare with e_{now}.

The least of the three is the best estimate of $e_{max(-)}$ and the corresponding x is the solution point in the –ve direction. The algorithm stops here.

The algorithm to find a point corresponding to $e_{max(+)}$ is similar with the exception that when we compare the values of e, we go for the largest value.

Example:

Consider aluminum plate #4 from end milling. Population data measurement with the CMM has been described in Chapter 5 (Section 5.3). Straightness tolerance h_t obtained for the population sample of 121 points is 0.005475 inch (see Table A.1.1.1).

Sample points following the present methodology are selected as follows. Initial 3 points on the basis of the surface profile discussed in Chapter 5 (Figure 22) are selected as (x = 4.1, z = -14.85474), (6.95, -14.854), and (9.85, -14.86047). The LS solutions for the initial set are l_0 = -0.001 and z_0 = -14.84944. The corresponding deviations (e) of the three points from the fit line respectively are -0.0012, 0.00239, and -0.00118, and the tolerance computed is h_t = 0.00359. Additional data points will be sampled intelligently with the help of the RE search to improve the zone.

It is clear from the e values of the 3 points, that a point with an optimum $e_{max(-)}$ may exist in the neighborhood of x = 4.1 or 9.85 (this has also been discussed in the RE

algorithm development earlier). Hence, the search algorithm is applied with $\Delta s = 0.05$ around both points. As an example, the first iteration is shown here.

Iteration 1: From $x = 4.1$ ($e = -0.0012$), $\Delta s = 0.05$ (i. e., $\Delta = 0.2$)

Point in the left ($x = 3.9$) is outside the boundary. In the right, at $x = 4.3$, $e = -0.00042$. Comparing the two e values, point $x = 4.1$ is better, so no move is made.

In the next iteration, the algorithm looks around $x = 4.1$ with a reduced interval $\Delta/2 = 0.1$. Note that for this iteration, $x = 4$ is within the boundary. This way the search is continued and stopped after the 3rd iteration. The solution obtained is ($x = 4.05$, $z = -14.85511$) corresponding to $e_{max(-)} = -0.00162$. Similarly, the search is conducted about $x = 9.85$. But the earlier solution at $x = 4.05$ is better.

In the same way, a solution point ($x = 6.85$, $z = -14.85302$) pertaining to an optimum $e_{max(+)}$ is obtained by applying the search around $x = 6.95$. The two points pertaining to $e_{max(-)}$ and $e_{max(+)}$ are added to the initial set of 3 points. Then the final tolerance estimated has an error of 6.1 % compared to the population value. The number of points sampled is 8 (3 initial and 5 in the search), although other points are evaluated during the search. In comparison to 121 points of the population, a reduced set of 8 points accounted for an accuracy of 93.9 % illustrating the potential of the applied search.

6.2 Flatness

For flatness evaluation, two pattern search methods, tabu search (TS) and hybrid search (HS) are applied to sample data points outside the initial set of five points determined from the surface error patterns discussed in Chapter 5. The objective is to

estimate the form tolerance with a reduced number of points. Therefore, instead of full-scale TS or HS with large number of allowed bad moves and re-start to find a very high quality solution, approximate TS and HS which can yield a good solution after only a few iterations are used. A fixed step size (i.e., the distance between each search point) of 0.1 inch is selected. Tabu size of one and maximum iteration of 35 have been used as mentioned in Chapter 4. Search is performed twice, first in the -ve direction from the initial fit plane and then in the +ve direction. It starts from a point that has got the maximum deviation among the initial data set depending on which direction a solution is being sought. It stops if the bad moves exceed the maximum bad moves allowed or if the iterations exceed the maximum iterations allowed. The algorithms are tested for different number of allowed bad moves (BM1, BM3, and BM5) as well as with no re-start (RS0) and one re-start (RS1). The reason for selecting these levels of strategy is detailed in the next chapter on experimental design. If a search needs to be re-started (e.g., in case of RS1) in any direction, then the re-start would originate from a point that has got the second maximum deviation from the initial fit plane (if it is outside the region searched before), and so on. Performance of TS and HS algorithms is compared in Chapter 7.

6.2.1 Tabu Search

To apply the tabu search method, a starting point from the initial set is selected as explained above. Figure 25 is a representation of a hypothetical search area divided into rectangular grids. The size of each grid is Δs by Δs, where Δs is the step size. Deviations (e) of neighborhood points with respect to the starting point (x_0, y_0) are obtained. Four neighborhood points as shown in Figure 25 are $(x_0 + \Delta s, y_0)$, $(x_0, y_0 + \Delta s)$, $(x_0 - \Delta s, y_0)$,

and (x_0, y_0 - Δs). A move is made to the point with the 'best' e value. If all the neighbors result in worse e value, then the least bad is selected. In case of searching for $e_{max(-)}$, the least is the 'best' and for $e_{max(+)}$, the largest is the 'best'.

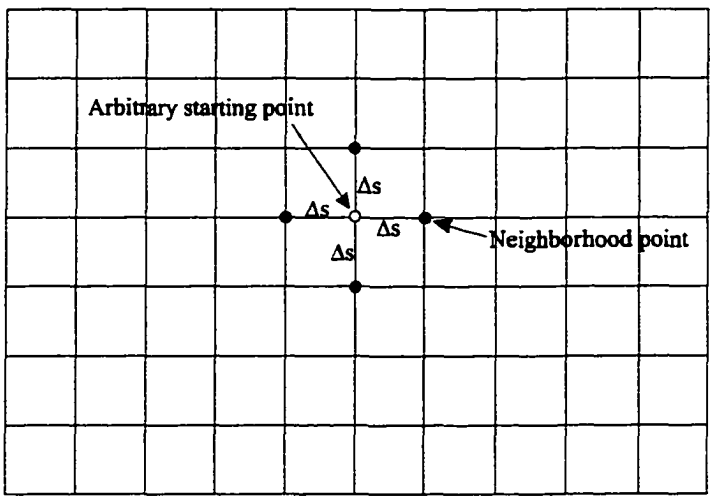


Figure 25: Neighborhood points of a starting point in tabu search.

For the next iteration, point (x_0, y_0) becomes tabu (forbidden). A move is defined to be tabu if any of its attributes is tabu-active for certain tenure or size. Tabu size determines the length of the tabu list. Tabu list is a short-term memory to store the coordinates of the most recent moves. This will prevent the reversal of the search to any point in the tabu list. Aspiration criteria are applied to determine when the tabu status of a move needs to be overridden [30]. In the present algorithm, a tabu size of one is used, so that in any iteration not all moves would be tabu. Hence, no aspiration criteria are required. The search continues based on the total number of iterations specified, the tabu size, and the number of bad moves allowed.

If a move is made to a neighborhood point with a better e value than that of the current point, that move is referred to as a "good" or "improving" move. If all neighbors have e values worse than that of the current point, then the move is made to the

neighborhood point with the least worse e value; and this move is referred to as a "bad" or "non-improving" move. The bad move parameter is an aid to help the algorithm escape the entrapment of local optimality. The larger the number of bad moves allowed, more iterations would be performed for a particular search. The search is terminated prematurely if the bad moves exceed the allowed number of bad moves, regardless of whether the number of moves has exceeded the maximum number of iterations allowed.

If the two successive iterations result in good moves in the same direction, then the present search is intensified in this direction to examine more elite solutions. Occasionally, it is possible that the normal moves of Δs length may not be sufficient to break away from the local optimality. Hence, a high influence (longer) move is performed to diversify the search in a region not visited before and that becomes the starting point for a re-start of the search. This has been done in the case of RS1, which allows a re-start. A flow chart for the approximate TS algorithm applied in this work is displayed in Figure 26. It is clear from the flow chart that the search stops if bad moves (BM) exceed maximum bad moves allowed (bm) or iterations exceed maximum number of iterations (M).

It is to be noted that the current CMM software is not integrated with the tabu search or hybrid search (discussed later) algorithm. Therefore, the population data (x, y, z) is collected using the CMM probe as outlined in Chapter 5 (see Figure 8 and Section 5.3) to represent the plate surface for which flatness is being evaluated. On this data set, the respective search algorithm is applied. Java codes developed for the algorithm to find points corresponding to $e_{max(-)}$ and $e_{max(+)}$ are given in Appendix B.

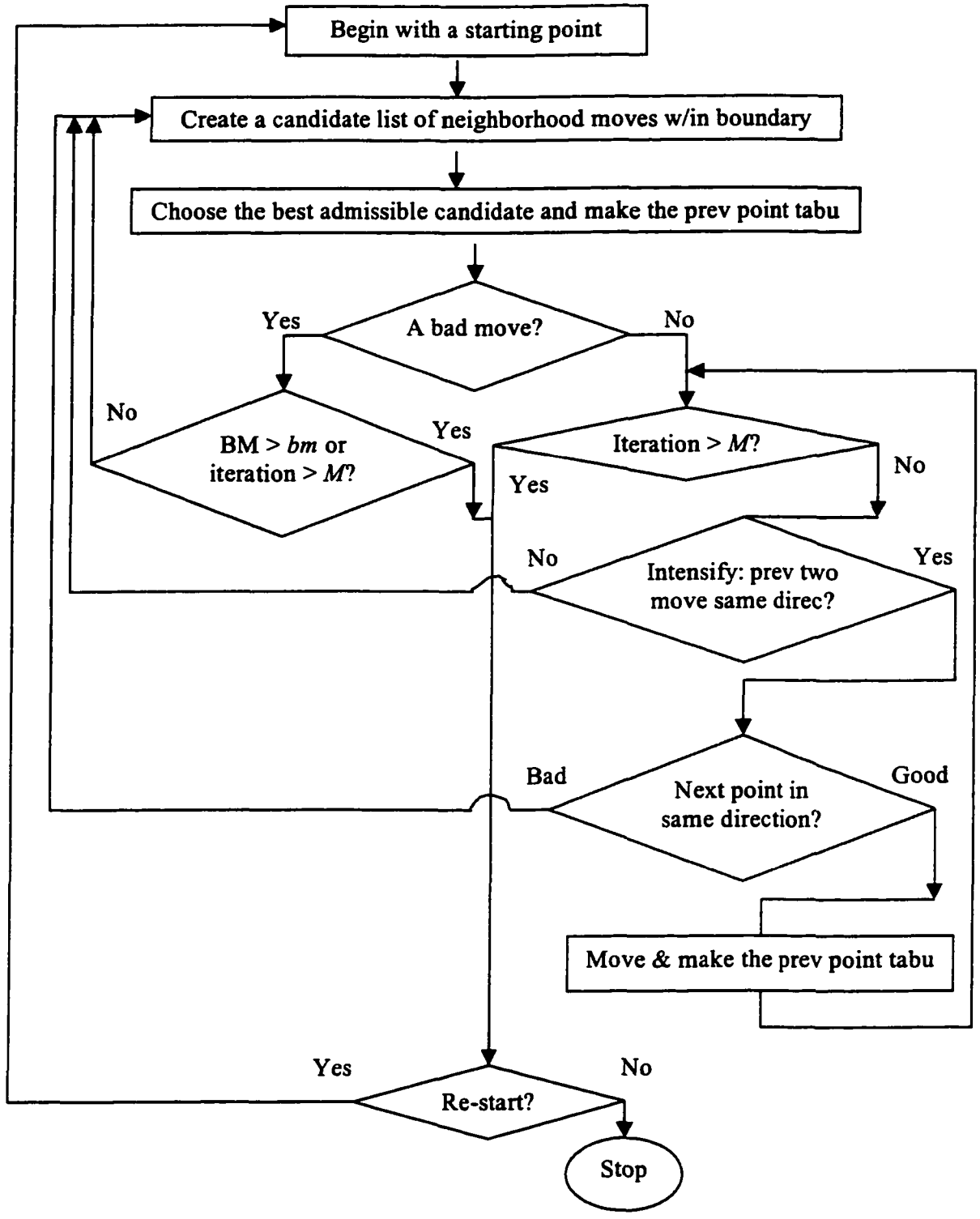


Figure 26: Tabu search algorithm flowchart for flatness estimation.

The TS algorithm has following steps:

Step 1: Initialization Step

Specify x & y values for the boundaries of the plate, step size (Δs), maximum bad moves (bm), maximum number of iterations (M), and the fit plane parameters for the initial set of points.

Read population data points (x, y, z) of the discretized plate from a file.

Specify the position (row & column) of a starting point in the discretized space pertaining to the search for $e_{max(-)}$ or $e_{max(+)}$ from the initial set to get (x_{now}, y_{now}, z_{now}). Also specify the corresponding e_{now}, which is known or can be obtained from the equation of the plane specified.

Set initial values:

e_{best} = 100 (a very large value) to search for a point with $e_{max(-)}$,

e_{best} = -100 (a very small value) to search for a point with $e_{max(+)}$,

iteration = 0, badmove = 0, k = 0,

tabu point (x = 100.0, y = 100.0, i.e., a hypothetical point) for tabu size = 1.

Step 2: Updating best solution and tabu list

If e_{now} is better than e_{best}, then $e_{best} = e_{now}$ and (x_{best}, y_{best}) = (x_{now}, y_{now}).

Otherwise, e_{best} remains unchanged.

Add (x_{now}, y_{now}) to the tabu list.

Step 3: Neighborhood Search from (x_{now}, y_{now})

iteration = iteration + 1

Define neighborhood points as:

$$x_1 = x_{now} + \Delta s, y_1 = y_{now} \qquad x_2 = x_{now}, y_2 = y_{now} + \Delta s$$

$x_3 = x_{now} - \Delta s, y_3 = y_{now}$ $x_4 = x_{now}, y_2 = y_{now} - \Delta s$

Ignore any point, if it is outside the surface boundaries.

Calculate e_i value for (x_i, y_i), $i = 1, 2, 3, 4$ that is not in the tabu list.

Let e_b denote the best e_i value with $b = i$.

If tie, then break arbitrarily with smaller i.

If $b = 1$ then define a direction $d_k = (1, 0)$

Else if $b = 2$ then $d_k = (0, 1)$

Else if $b = 3$ then $d_k = (-1, 0)$

Else (i.e., $b = 4$), $d_k = (0, -1)$.

Step 4: Intensification Check Step

If $k \leq 1$ or ($k > 1$ and $d_k \neq d_{k-1}$) then

if e_b is better than e_{now}, go to step 6

otherwise, go to step 7.

Otherwise, if e_b is not better than e_{now}, go to step 7

otherwise, go to step 5.

Step 5: Intensification Step

$(x_{now}, y_{now}) = (x_b, y_b)$, $e_{now} = e_b$

$(x_{next}, y_{next}) = (x_{now}, y_{now}) + \Delta s \times d_k$

If (x_{next}, y_{next}) is not within the boundary, go to step 2.

Otherwise, calculate e_{next}.

If e_{next} is better than e_{now}, then

iteration = iteration + 1

if iteration > M; stop and check for e_{best}.

otherwise, $k = k + 1$

$d_k = d_{k-1}$

$(x_b, y_b) = (x_{next}, y_{next})$ and $e_b = e_{next}$

Go to the start of step 5.

Otherwise go to step 2.

Step 6: Improving Move Step

$(x_{now}, y_{now}) = (x_b, y_b)$, $e_{now} = e_b$

If iteration > M; stop and check for e_{best}.

Otherwise, $k = k + 1$

Go to step 2.

Step 7: Non-improving Move Step

$(x_{now}, y_{now}) = (x_b, y_b)$, $e_{now} = e_b$

badmove = badmove + 1

If badmove > bm or iteration > M; stop and check for e_{best}.

Otherwise, $k = k + 1$

Go to step 2.

Step 8: Output Step

Display the solution point $(x_{best}, y_{best}, z_{best})$ with e_{best} and the iteration needed to reach the solution. If stopped due to bm, then iteration should be less than M. Also, display the area searched (i.e., this row to this row and this column to this column).

Step 9: Re-start Step

If re-start is desired, then start in a region, which was not covered in the previous

search to diversify the search. Specify the row & column of the starting point pertaining to the next best point in the initial set to get (x_{now}, y_{now}, z_{now}) and specify the corresponding e_{now}.

Set initial values:

iteration = 0, badmove = *0*, k = 0.

Go to step 2.

Example:

Consider aluminum plate #5 from end milling. Population data measurement has been explained in Chapter 5. Flatness tolerance h_t obtained for the population of 651 points is 0.005959 inch (see Table A.2.1.2).

Sample points following the present methodology are selected as follows. Initial 5 points as discussed in Chapter 5 guided by the surface profile are selected as (x = 8, y = 5, z = -14.86218), (6.6, 5.5, -14.85546), (7.9, 6, -14.85815), (5.2, 6.4, -14.85391), and (6.4, 7, -14.85195). For this set, the LS solutions are l_0 = -0.0015, m_0 = 0.003138 and z_0 = -14.86487. The corresponding deviations (e) of the 5 points from the fit plane respectively are -0.001, 0.002051, -0.000258, -0.001323, and 0.000554. The computed tolerance represents an error of 43.4 % from the zone estimated for the population. Additional data points are chosen intelligently with the help of the approximate TS with step size of 0.1, no re-start, and one bad move allowed (RS0BM1).

The starting points to search for $e_{max(-)}$ and $e_{max(+)}$ based on the e values obtained above for the initial set would be (x = 5.2, y = 6.4) and (6.6, 5.5), respectively. For $e_{max(-)}$, the search yields a solution as (5, 6.6, -14.85461) and for $e_{max(+)}$ as (6.8, 5.2, -14.85487).

The two points are added to the initial set of 5 points. Then the tolerance estimated has an error of only 10.4 % compared to the population zone. The number of points sampled is 15 (5 initial and 10 in the search), although other points are evaluated during the search. In comparison to 651 points of the population, a reduced set of 15 points accounted for an accuracy of 89.6 % proving the usefulness of the approach.

6.2.2 Hybrid Search

Hybrid search (HS) developed is a combination of coordinate search (CS), modified Hooke-Jeeves (HJ) pattern search, and TS (as explained above with tabu size of one). The methods of CS and HJ search have been discussed in Chapter 3. A flowchart for the approximate HS algorithm developed in this work is shown in Figure 27. In the flow chart, BM stands for bad moves, *bm* for maximum bad moves allowed, and *M* for maximum iterations allowed. The algorithm begins with CS exploratory moves, then utilizes modified HJ search along the pattern direction as well as TS.

For starting point (x_0, y_0), the first iteration will be as follows:

Evaluate a neighborhood point $(x_0 + \Delta s, y_0)$ for its deviation (e).

If it is better than (x_0, y_0), evaluate $(x_0 + \Delta s, y_0 + \Delta s)$.

If it is better than $(x_0 + \Delta s, y_0)$, this is the new iterate, i.e., (x_1, y_1).

Otherwise evaluate $(x_0 + \Delta s, y_0 - \Delta s)$.

If it is better than $(x_0 + \Delta s, y_0)$, this is the new iterate.

Otherwise, $(x_0 + \Delta s, y_0)$ is the new iterate.

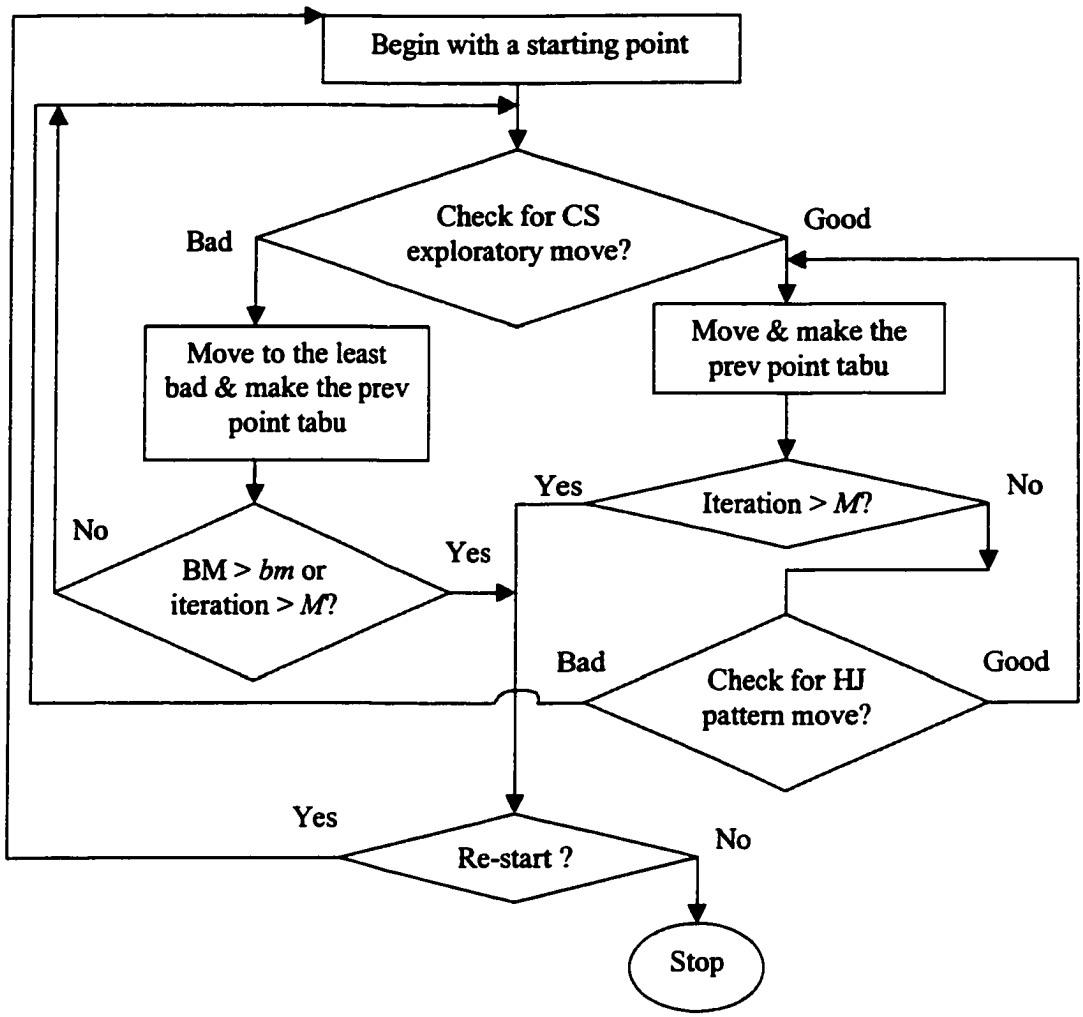


Figure 27: Hybrid search algorithm flowchart for flatness estimation.

Otherwise, i.e., $(x_0 + \Delta s, y_0)$ is not better than (x_0, y_0), evaluate $(x_0 - \Delta s, y_0)$.

If it is better than (x_0, y_0), evaluate $(x_0 - \Delta s, y_0 + \Delta s)$.

If it is better than $(x_0 - \Delta s, y_0)$, this is the new iterate.

Otherwise evaluate $(x_0 - \Delta s, y_0 - \Delta s)$.

If it is better than $(x_0 - \Delta s, y_0)$, this is the new iterate.

Otherwise, $(x_0 - \Delta s, y_0)$ is the new iterate.

Otherwise evaluate $(x_0, y_0 + \Delta s)$.

If it is better than (x_0, y_0), this is the new iterate.

Otherwise evaluate $(x_0, y_0 - \Delta s)$.

If it is better than (x_0, y_0), this is the new iterate.

Otherwise, select the least bad from $(x_0 + \Delta s, y_0)$, $(x_0 - \Delta s, y_0)$, $(x_0, y_0 + \Delta s)$, and $(x_0, y_0 - \Delta s)$ as the new iterate.

For the next iteration (x_0, y_0) is tabu. If the new iterate (x_1, y_1) is a bad move, then again CS moves are explored about it. Otherwise, the HJ pattern move is evaluated. That is, the algorithm takes a step $[(x_1, y_1) - (x_0, y_0)]$ from (x_1, y_1) in the pattern direction. If e of $\{(x_1, y_1) + [(x_1, y_1) - (x_0, y_0)]\}$ is better than (x_1, y_1), then it is accepted as new iterate (x_2, y_2). If the pattern move is unsuccessful, then the method reduces to CS about (x_1, y_1). This way the search continues and stops when BM exceeds *bm* or iteration exceeds *M*.

Example:

Consider the example given for the approximate TS for no re-start and one bad move allowed (RS0BM1). For the same initial set of points, the starting points for the HS algorithm remain the same. The approximate HS obtains the same solution points but with fewer iterations. The number of points sampled is 11 (5 initial and 6 in the search). In comparison to 651 points of the population, a reduced set of 11 points accounted for an accuracy of 89.6 % proving the usefulness of the method.

CHAPTER 7

EXPERIMETNAL DESIGN

In order to explore the effects of certain factors when using search methods in selecting sample points, experiments are carried out. Before this, the factors to be varied in the experiment, the ranges over which these should be varied, the specific levels at which runs should be made, and the dependent variable are determined. The response variables are identified to be number of sampled data points using the search methodology and corresponding absolute percentage error of the form tolerance achieved in comparison to the entire population (i.e., a very large sample). The first independent variable (factor A) used throughout all analyses is manufacturing process (i.e., surface pattern) to determine if the number and location of data points depend upon the process. Two levels of manufacturing process are chosen, end milling and face milling.

The present methodology proposes to start the sampling with initial set of data points dictated by the surface pattern obtained in Chapter 5 and then to sample additional points using a search technique. Selection of the initial set by other means may not be able to capture the 'true' surface pattern and hence may result in higher tolerance estimation error or may require more points to be sampled to achieve the desired level of accuracy [63].

7.1 Straightness

For the evaluation of the straightness of an edge, the second independent variable (factor B) is step size (i.e., Δs) used in the region elimination (RE) search method.

Badiru [9] selected the step size to be 0.2 inch for the tabu search algorithm. Three levels of step size as 0.05, 0.1, and 0.2 inch are selected making Δ of the RE algorithm = 4 × step size = 0.2, 0.4, and 0.8. Four random plates are selected from each of the end-milled plates (# 4, 12, 9, 2) and face-milled plates (# 3, 1, 11, 7). Details of the machining conditions and measurement surface have been outlined in Chapter 5. Other factors are not changed for the experiment.

As explained in Chapter 5 on the basis of the surface patterns (see Figures 22-23), three initial points are selected (one from each end and one from the center) for end milling and four initial points are selected (one each from section 1, 2, 3, and 4) for face milling. Plates are nested within the levels of manufacturing process, whereas step size and process are crossed creating a nested-factorial design [58]. Each plate is repeated four times with a different set of the initial points; however, the same set is used for all the step sizes. All the observed data ($N = 2 \cdot 4 \cdot 3 \cdot 4 = 96$) have been presented in Table 1.

If y_{ijkl} denotes the observed response, the linear statistical model for the design is

$$y_{ijkl} = \mu + A_i + B_j + P_{k(i)} + (AB)_{ij} + (BP)_{jk(i)} + \varepsilon_{(ijk)l} \qquad (10)$$

where, μ is the overall mean effect, A_i is the effect of the ith manufacturing process, B_j is the effect of the jth step size, $P_{k(i)}$ is the effect of the kth plate within the ith level of process, $(AB)_{ij}$ is the process × step size interaction, $(BP)_{jk(i)}$ is the step size × plates within process interaction and $\varepsilon_{(ijk)l}$ is the random experimental error term. As the levels (treatments) of process and step size are specifically selected, both are fixed. Plate is a random factor. So, it is a mixed model.

Table 1: Observed response - number of sampled points (absolute percentage error) for straightness estimation

		Step size (inch)		
Manufacturing Process	Plate	0.05	0.1	0.2
End milling (Al plates, Population=121 points)	4	8(6.1)	7(12.8)	7(14.7)
		7(0.0)	6(0.14)	6(10.2)
		7(0.0)	6(7.2)	4(20.9)
		7(6.9)	6(8.2)	4(19.7)
	12	7(6.75)	7(6.75)	5(14.23)
		7(11.4)	5(6.08)	4(6.97)
		6(3.25)	5(2.69)	5(5.82)
		4(0.61)	4(0.61)	4(0.61)
	9	5(4.7)	4(4.7)	3(13.3)
		6(2.0)	7(2.0)	6(0.0)
		7(2.04)	7(4.2)	5(2.6)
		3(9.6)	4(3.8)	4(8.26)
	2	6(9.58)	4(21.86)	4(19.05)
		5(9.84)	5(9.84)	4(3.54)
		6(15.17)	5(14.53)	5(24.9)
		8(3.76)	8(2.43)	5(30.62)
Face milling (CI plates, Population = 56 points)	3	11(7.35)	8(11.0)	6(2.7)
		9(8.6)	7(8.6)	6(6.06)
		11(6.74)	9(9.86)	6(2.33)
		10(1.24)	12(0.31)	9(0.48)
	1	9(6.3)	8(2.0)	6(2.0)
		11(3.9)	8(13.9)	7(13.9)
		9(18.4)	10(13.56)	7(25.2)
		9(0.85)	7(13.75)	6(16)
	11	11(9.4)	10(4)	7(6.2)
		10(12.0)	9(12.0)	6(3.9)
		13(4.2)	7(4.4)	6(5.7)
		9(2.9)	9(0.5)	8(2.3)
	7	10(4.65)	6(0.03)	6(0.03)
		8(4.17)	7(4.17)	7(4.17)
		6(7.93)	7(7.93)	6(7.93)
		10(2.7)	7(0.65)	5(11.14)

The null hypotheses are constructed as follows:

1. Manufacturing process has no effect; H_0: $A_i = 0$, $i = 1, 2$
2. Step size has no effect; H_0: $B_j = 0, j = 1, 2, 3$.
3. Plate within process has no effect; H_0: $P_{k(i)} = 0$, $k = 1,2,3,4$; $i = 1, 2$.

4. There is no interaction between process and step size; H_0: $(AB)_{ij} = 0$ for all i, j.
5. There is no interaction between step size and plate within process; H_0: $(BP)_{jk(i)} = 0$ for all i, j, k.

Symbolically the total sum of squares SS_T may be written as

$$SS_T = SS_A + SS_B + SS_{P(A)} + SS_{AB} + SS_{BP(A)} + SS_E \quad (11)$$

Computational formulas for the sums of squares can be obtained from [58]. Assume that the model (Eq. 10) is adequate and that the error terms ε_{ijkl} are normally and independently distributed with mean zero and constant variance σ^2. Then each of the ratios of mean squares $MS_A/MS_{P(A)}$, $MS_B/MS_{BP(A)}$, $MS_{P(A)}/MS_E$, $MS_{AB}/MS_{BP(A)}$, and $MS_{BP(A)}/MS_E$ is distributed as F with ν_1 numerator degrees of freedom (df) and $\nu 2$ denominator df, and the critical region would be the upper tail of the F distribution. The analysis of variance is done using SAS and is summarized in Table 2. Complete SAS output is given in Appendix C. A null hypothesis H_0 is rejected when $F_0 > (F_{\alpha,\nu 1,\nu 2})$ for $\alpha = 0.05$.

Table 2: ANOVA for the three-factor nested-factorial model for straightness

Source	ν_1, ν_2	# of points		% error		$F_{0.05}$
		MS	F_0	MS	F_0	
Mfg process (A)	1, 6	168.01	32.63*	61.6	0.41	5.99
Step size (B)	2, 12	45.29	84.7*	111.29	3.51	3.89
P(A), plates w/in A	6, 72	5.15	3.1*	149.07	5.03*	2.234
A × B interaction	2, 12	6.17	11.53*	84.52	2.66	3.89
B × P(A) interaction	12, 72	0.535	0.32	31.74	1.07	1.902
Error	72	1.663		29.646		
Total	95					

To check the model adequacy, the residuals are examined. It is clear from the SAS output that the residuals are structureless (i.e., followed no clear pattern). Therefore,

they are independent. A test of the normality (0, σ^2) assumption is made plotting the residuals as suggested in [9, 18, 58]. It can be observed from the SAS output, the plot of the cumulative probability vs the residuals is a straight line centered about zero. Therefore the normality test is satisfied. An analysis of the results is presented in the next chapter.

7.2 Flatness

In case of flatness estimation, factor A is again manufacturing process: end and face milling. A fixed step size (i.e., Δs) of 0.1 in. is chosen. Factor B is strategy (i.e., number of bad moves and re-start allowed in the search algorithm). In general, increasing the allowed bad moves will increase the number of sampled points. Badiru [9] used three levels of bad moves as 5, 6, and 10 in his work on tabu search (TS). In the pilot work [8], the authors demonstrated the TS algorithm with 3 bad moves and hybrid search (HS) algorithm with 1 bad move. They [8] also illustrated the HS algorithm with no re-start as well as with one re-start. The goal of the present work is to reduce the sample size while maintaining the estimation accuracy. Therefore, four levels of strategy are selected as RS0BM1 (i.e. re-start = 0 and bad move = 1), RS0BM3, RS0BM5, and RS1BM1 for the investigation. Factor C is search method and two levels are: tabu search and hybrid search. Four random plates (# 10, 5, 7, 4) are selected from the end-milled plates and plates (# 2, 9, 11, 5) from the face-milled specimens. Details of the machining condition and measurement surface have been outlined in Chapter 5. Plates are nested within the levels of manufacturing process. Therefore, this design with four factors is a nested-factorial design [58].

Each plate is repeated four times with a different set of the initial points; however, the same set is used for the four levels of strategy and two levels of search technique. The number and locations of the initial points for both milling processes on the basis of the surface patterns have been discussed in Chapter 5. The initial set consists of five points for both processes. In case of an end-milled plate, from top to bottom of cut (i.e., $y = 5$ to 7) segments 0 - 0.2, 0.4 - 0.8, 0.95 - 1.15, 1.3 - 1.7, 1.8 - 2 inch from the top are important to choose the initial set. In the first segment, a point at the end of cut ($x = 7.8$ - 8) can be selected; in the second, a point at the center ($x = 6.4$ - 6.6); in the third, at the end; in the fourth, at the start ($x = 5$ - 5.2); and in the last segment, at the center. In case of a face-milled plate, five points can be selected as follows: one each near the cutter entry in section 1 ($y \approx 3$), near the lower hole in section 2 ($y \approx 1.5$), near the upper hole in section 3 ($y \approx 2.5$), near the cutter entry in section 4 ($y \approx 1$), and the last any where near the center ($y \approx 2$) or the cutter exit. In general, $N = 2 \cdot 4 \cdot 4 \cdot 2 \cdot 4 = 256$ observations and the observed data have been presented in Table 3.

If y_{ijklm} denotes the observed response, the linear statistical model for the design is

$$y_{ijklm} = \mu + A_i + B_j + P_{k(i)} + C_l + (AB)_{ij} + (AC)_{il} + (BP)_{jk(i)} + (BC)_{jl} + (CP)_{lk(i)} + (ABC)_{ijl} + (BCP)_{jlk(i)} + \varepsilon_{(ijkl)m} \quad (12)$$

where, μ is the overall mean effect, A_i is the effect of the ith manufacturing process, B_j is the effect of the jth strategy, $P_{k(i)}$ is the effect of the kth plate within the ith level of process, C_l is the effect of the lth search method, $(AB)_{ij}$ is the process × strategy interaction, $(AC)_{il}$ is the process × search algorithm interaction, $(BP)_{jk(i)}$ is the strategy × plates within process interaction, $(BC)_{jl}$ is the strategy × search method interaction, $(CP)_{lk(i)}$ is the algorithm × plates within process interaction, $(ABC)_{ijl}$ is the process ×

Table 3: Observed response - number of sampled points (absolute percentage error) for flatness estimation

Mfg. Process	Plate	Tabu search method				Hybrid search method			
		Strategy				Strategy			
		RS0 BM1	RS0 BM3	RS0 BM5	RS1 BM1	RS0 BM1	RS0 BM3	RS0 BM5	RS1 BM1
End milling (pop = 651 points)	10	10(34.8)	16(33.7)	26(33.7)	15(34.8)	9(33.7)	14(33.7)	23(33.7)	14(33.7)
		8(21.8)	15(26.7)	22(26.7)	14(21.8)	8(21.8)	13(21.8)	19(21.8)	16(17.3)
		11(46.1)	18(41.9)	24(41.9)	17(21.8)	10(46.1)	19(46.1)	31(46.1)	16(16.3)
		8(35.2)	15(32.8)	25(32.8)	17(16.6)	8(35.2)	15(35.2)	29(35.2)	12(24.7)
	5	10(24.3)	20(26.9)	28(26.9)	15(24.2)	13(30.0)	21(30.0)	28(30.0)	18(30.0)
		15(10.4)	24(10.4)	35(10.4)	21(10.4)	11(10.4)	16(10.4)	26(10.4)	19(10.4)
		12(46.9)	20(48.1)	30(4.8)	17(30.0)	12(44.1)	22(44.1)	32(44.1)	16(31.2)
		10(44.1)	18(44.1)	26(44.1)	18(37.7)	10(44.1)	17(44.1)	25(44.1)	15(37.7)
	7	12(40.8)	20(41.2)	32(41.2)	15(40.8)	10(40.0)	17(40.8)	26(41.2)	14(40.0)
		17(35.3)	21(35.3)	30(36.4)	31(30.7)	16(35.3)	21(35.3)	30(36.4)	28(30.7)
		19(33.3)	29(33.3)	41(33.3)	22(33.3)	16(42.9)	23(42.9)	31(42.9)	19(42.9)
		12(49.6)	19(49.6)	29(38.6)	16(47.4)	12(49.6)	19(49.6)	27(49.8)	17(47.4)
	4	13(57.1)	19(57.1)	27(57.1)	18(48.3)	15(50.9)	21(50.9)	27(50.9)	20(48.3)
		14(51.6)	22(51.6)	32(41.1)	22(38.2)	14(51.6)	21(51.6)	29(46.9)	21(38.2)
		13(41.8)	22(24.1)	27(24.1)	18(29.5)	12(58.7)	18(58.7)	26(58.7)	16(29.5)
		10(21.8)	17(21.8)	23(21.8)	21(8.0)	10(21.8)	16(21.8)	21(21.8)	18(21.8)
Face milling (pop = 410 points)	2	11(29.2)	20(30.9)	27(24.2)	19(11.5)	9(31.6)	17(31.3)	26(31.9)	16(11.5)
		13(17.6)	21(7.1)	27(7.1)	19(17.6)	11(17.6)	20(6.1)	24(6.1)	17(17.6)
		13(13.8)	19(10.5)	25(10.5)	17(13.8)	13(14.2)	18(10.8)	27(10.8)	17(14.2)
		12(28.2)	21(28.2)	29(23.8)	21(28.2)	13(27.1)	23(22.6)	29(21.5)	22(12.2)
	9	12(7.4)	19(7.3)	24(7.3)	21(7.4)	10(7.4)	15(7.4)	22(7.4)	18(7.4)
		11(9.4)	17(9.4)	24(9.4)	16(9.4)	11(9.4)	17(9.4)	23(9.4)	16(9.4)
		9(17.4)	17(17.4)	25(17.4)	15(8.3)	10(15.7)	18(17.4)	26(17.4)	14(5.9)
		13(7.9)	19(8.1)	29(16.3)	23(7.9)	9(8.9)	16(8.9)	20(8.9)	18(4.6)
	11	13(2.8)	24(2.8)	34(2.8)	25(11.8)	11(3.4)	20(2.6)	25(2.6)	21(8.5)
		12(9.9)	21(9.3)	28(9.3)	22(9.9)	10(10.2)	17(9.0)	24(9.0)	18(10.2)
		16(11.6)	29(11.6)	41(11.6)	27(11.6)	14(10.5)	21(10.5)	27(10.5)	26(7.0)
		16(0.26)	25(0.26)	33(0.26)	26(0.26)	14(0.65)	20(0.65)	26(0.65)	21(2.8)
	5	13(6.0)	21(6.0)	29(6.0)	21(6.0)	13(6.0)	20(6.0)	27(6.0)	19(6.0)
		15(4.8)	22(4.8)	29(4.8)	26(4.8)	13(4.8)	18(4.8)	24(4.8)	19(4.8)
		13(9.3)	25(9.3)	37(9.3)	20(8.6)	14(9.3)	23(9.3)	32(9.3)	24(8.6)
		8(15.3)	16(15.3)	24(15.3)	15(14.4)	8(15.3)	15(15.3)	21(15.3)	16(19.7)

strategy × search method interaction, $(BCP)_{jlk(i)}$ is the strategy × algorithm × plates within process interaction, and $\varepsilon_{(ijkl)m}$ is the random experimental error term. Manufacturing process, strategy, and search algorithm are fixed. Plate is a random factor. Hence, it is a

mixed model.

Computational formulas for the mean squares can be obtained from [58]. Assume that the model (Eq. 12) is adequate and that the error terms $\varepsilon_{(ijkl)m}$ are normally and independently distributed with mean zero and constant variance σ^2. Then each of the ratios of mean squares $MS_A/MS_{P(A)}$, $MS_B/MS_{BP(A)}$, $MS_C/MS_{CP(A)}$, $MS_{P(A)}/MS_E$, $MS_{AB}/MS_{BP(A)}$, $MS_{AC}/MS_{CP(A)}$, $MS_{BP(A)}/MS_E$, $MS_{BC}/MS_{BCP(A)}$, $MS_{CP(A)}/MS_E$, $MS_{ABC}/MS_{BCP(A)}$, and $MS_{BCP(A)}/MS_E$ is distributed as F with ν_1 numerator df and $\nu 2$ denominator df, and the critical region would be the upper tail of the F distribution. The analysis of variance is shown in Table 4. Complete SAS output is given in Appendix C. A null hypothesis H_0 is rejected when $F_0 > (F_{\alpha,\nu 1,\nu 2})$ for $\alpha = 0.05$.

Table 4: The ANOVA table for the four-factor nested-factorial model

Source	ν_1, ν_2	# of points		% error		$F_{0.05}$
		MS	F_0	MS	F_0	
Mfg process (A)	1, 6	10.16	0.08	37713.9	39.82*	5.99
Strategy (B)	3, 18	2571.8	467.27*	267.685	6.96*	3.16
Algorithm (C)	1, 6	164.16	8.49*	94.466	4.37	5.99
P(A), plates w/in A	6, 192	123.95	12.21*	947.156	9.89*	2.10
A × B interaction	3, 18	17.0	3.09	119.785	3.11	3.16
A × C interaction	1, 6	32.347	1.67	181.054	8.37*	5.99
B × P(A) interaction	18, 192	5.504	0.54	38.454	0.40	1.61
B × C interaction	3, 18	11.483	2.99	23.995	3.47*	3.16
C × P(A) interaction	6, 192	19.337	1.90	21.62	0.23	2.10
A × B × C	3, 18	2.92	0.76	19.109	2.76	3.16
B × C × P(A)	18, 192	3.841	0.38	6.91	0.07	1.61
Error	192	10.155		95.763		
Total	255					

To check the model adequacy, the residuals are examined. It is clear from the SAS output that the residuals are structureless. Therefore, they are independent. Plot of the cumulative probability vs the residuals is a straight line centered about zero. Therefore the normality test is satisfied. The results are discussed in the next chapter.

CHAPTER 8

RESULTS AND DISCUSSIONS

Based on the analysis of variance presented in Tables 2 and 4 in Chapter 7 and SAS outputs in Appendix C, the results are analyzed in this chapter.

8.1 Straightness

Considering the number of sampled points as the dependent variable in Table 2, manufacturing process is significant, step size is significant, plates within manufacturing process differ significantly, and interaction between manufacturing process and step size is significant. Tukey's studentized range test (see Appendix C) reveals that end-milled plates require significantly less number of points (mean = 5.5) than that of face-milled plates (8.15). Step sizes of 0.2, 0.1, and 0.05 respectively require 5.59, 6.9, and 7.97 average points significantly different from each other. Concerning the significant interaction effect, it is clear from Figure 28 that changing step size from 0.05 to 0.1 or from 0.1 to 0.2, sample size decreases rapidly for face milling in comparison to end milling. Considering the absolute % error in the tolerance estimation compared with the population as the dependent variable, plates are significant; however, manufacturing process and step size are not significant.

Plates within manufacturing process are significant in both response variables (number of points and % error). This may be due to the variation in the plate surface before the milling and/or variation resulting from different set of the initial points. If the initial points selected truly represent the surface pattern, then the number of sampled

points and the resulting % error are less.

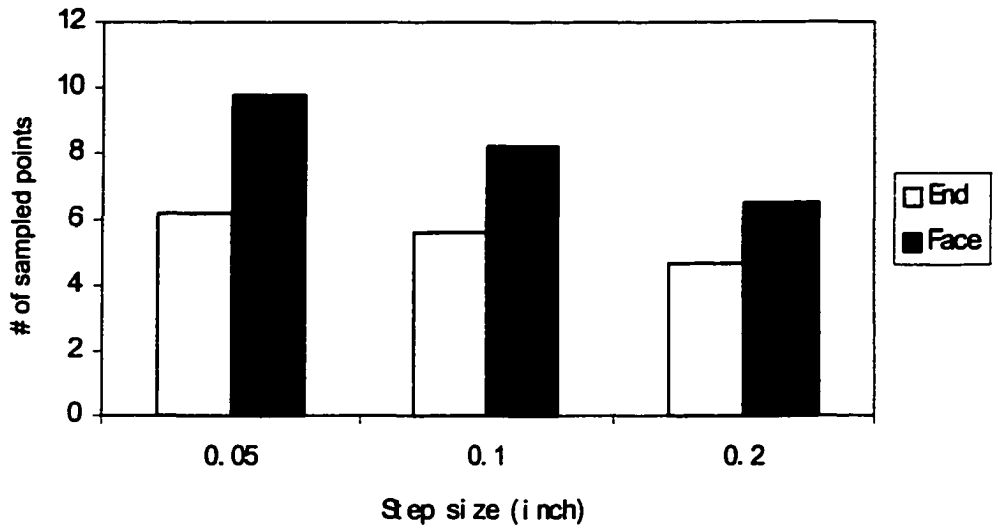


Figure 28: Effect of manufacturing process and step size in straightness estimation.

8.2 Flatness

Considering the number of sampled points as the dependent variable in the ANOVA Table 4, strategy is significant, search algorithm is significant, and plates within manufacturing process also differ significantly. However, manufacturing process is not significant, i.e., mean number of sampled points for end milling (19.3) is significantly same as for face milling (19.7). Pertaining to strategy, Tukey's studentized range test (see Appendix C) reveals that average number of points for RS0BM1 (11.9) is significantly less than RS1BM1 (18.9) or RS0BM3 (19.7), which are not significantly different, but significantly less than RS0BM5 (27.4). Also, the average sample size for HS (18.7) is significantly smaller than TS (20.3). Considering the absolute % error in the tolerance estimation compared with the population as the dependent variable, manufacturing process is significant, strategy is significant, plates within manufacturing

process are significant, manufacturing process × algorithm interaction is significant, and strategy × algorithm interaction is also significant. Tukey's test reveals that the average % error for end milling (35.2) is significantly higher than face milling (10.95). Pertaining to strategy, % errors for RS0BM1 (24.91), RS0BM3 (24.15) and RS0BM5 (23.06) are same as well as RS0BM3, RS0BM5 and RS1BM1 (20.24) are same. But the error for RS0BM1 is significantly higher than that for RS1BM1. Also, the average error for HS (23.7) is not significantly different from TS (22.5). These results have also been displayed in Figures 29 - 31. Concerning the significant interaction effects, Figure 31 shows that the error increases from TS to HS in end milling whereas decreases in face milling. Also, it increases from TS to HS in all four levels of strategy with the highest increment in RS0BM5 and almost no increment in RS1BM1.

Considering both the sample size and accuracy of the flatness estimation, it can be concluded that the tolerance zone obtained from significantly same sample size for face milling has a very high accuracy (i.e., 89% of the population sample) as compared to 65% for end milling. This is because the average tolerance zone of the four plates obtained from the population sample for face milling (0.0023 inch) is smaller in comparison to end milling (0.00704), i.e., the face milling process capability is better in the present case because of the machining conditions and workpiece used. Strategy RS0BM1 yields a very low sample size without losing significant accuracy in comparison with RS0BM3 and RS0BM5. A larger sample size with RS1BM1 increases the accuracy (more evident for end milling). Although HS algorithm results in a significantly smaller sample size for significantly same accuracy, it is practically not much different from TS. Hence, it will be a matter of convenience in choosing between the algorithms. As

mentioned earlier, variability of plates within manufacturing process can be because of the variation in the raw material, variation associated with the process capability, and variation resulting from different set of the initial points. If the initial points represent the surface pattern truly, then the sampled points and the resulting error are less.

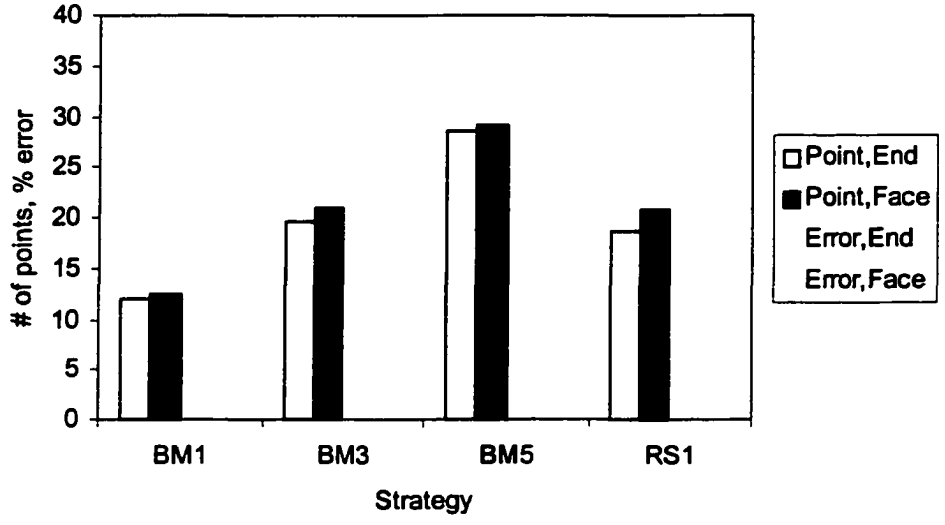


Figure 29: Results of the flatness estimation using tabu search.

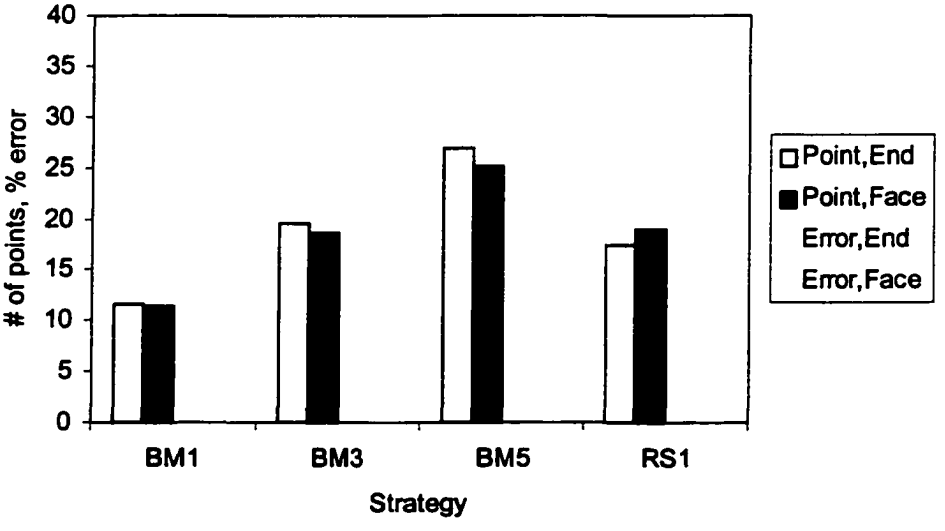


Figure 30: Results of the flatness estimation using hybrid search.

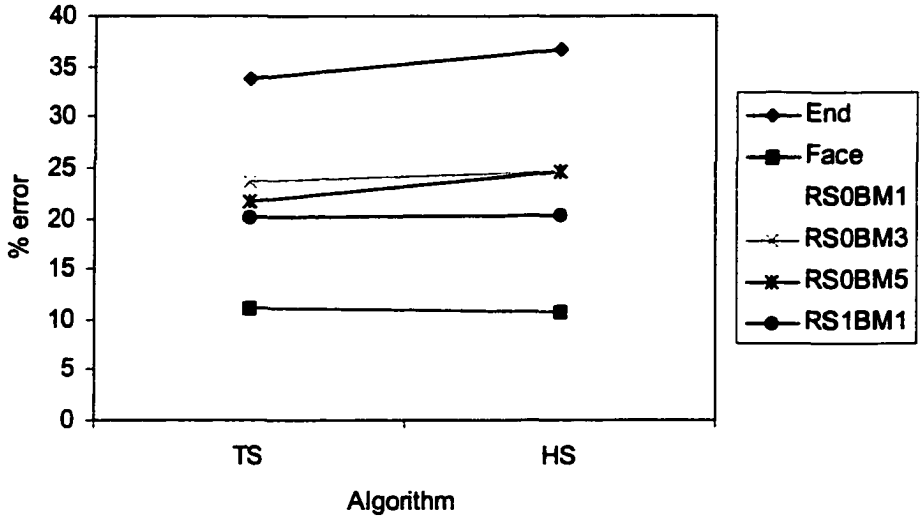


Figure 31: Manufacturing process - algorithm and strategy - algorithm plot.

8.3 Comparison with Previous Work

The present methodology exploits the knowledge of manufacturing error patterns, as suggested in [51, 63] for initial sample point selection. Then it uses search methods intelligently for continuing sampling. It is interesting to find that following this approach, the number of the sample points is much smaller than that required by other generally applied procedures for achieving the desired accuracy level.

In case of straightness evaluation, it is observed that smaller step size increases the sample size with no significant effect on the accuracy of the estimation. The lowest sample size corresponding to step size of 0.2 inch is found to be 5 and 7 respectively for end and face milling. This yields a zone having an average accuracy of 92.6 % with respect to the zone obtained from the population sample. Length of the line feature considered is 6 inches. For a line of 40 in., Hocken et al. [36] reported that the parameters described in the fit line did not converge to stable values until a larger number

of points (> 50) were sampled. An increase in the size of a feature may increase the sample size [18, 100] or may not [95-96]. The present work is focused around the characterization of surface error pattern. Depending on the surface profile, increasing the length to 40 in. may increase the initial set size and in turn the total number of sampled points or may reduce the estimation accuracy. Despite this, the sample size obtained from the present method is much smaller than that would be expected based on the work of Hocken et al. [36] for the desired accuracy, which identifies its potential for future use.

For the population data, average straightness tolerance of four plates for end milling is 0.005 inch and for face milling is 0.0019. This means the latter has a better process capability. This is because of the machining conditions and workpiece used in this work. Otherwise, in general, face milling is a 'roughing' operation and end milling is a 'finishing' operation. As the process capability improves the sample size should decrease [56, 94]. But, in the present case, it is observed that end milling requires less number of points than face milling. This is because of an extra initial point in face milling based on the surface error profile and the fact that the search is performed in the neighborhood of four points instead of three points as in end milling. For flatness, again the face milling process capability is better as the average tolerance zone of four plates computed from the population data for this process is smaller (0.0023) in comparison to end milling (0.00704). Hence, for same sample size, face milling is found to yield a much higher accuracy (89 %) than end milling (65 %).

Flatness estimation with strategy RS0BM1 (i.e., no re-start and one bad move) requires a sample size of 12 and yields an absolute error of 37.2 and 11.9 % compared to the zone obtained from the population sample for end and face milling, respectively.

Strategy RS1BM1 (i.e., one re-start) increases the sample size to 20 but reduces the error to 29.6 and 10.7 %, respectively. This result is in agreement with that of Hurt [41], who recommended measuring 20 or more points in order to get a reasonable flatness value by the LS method. Caskey et al. [13] studied the behavior of a plane fit and could not find the sampling strategy dispersion converging to zero even with 49 points. For a plane of 20 × 20 in., Hocken et al. [36] reported that the parameters described in the fit plane did not converge to stable values until a larger number of points (> 49) were sampled. In comparison to the work of [13, 36], the present method for point selection looks more efficient. However, it is be noted that the surface measured in this study is 3 × 2 in. Therefore, detailed comparisons need to be made considering the surface of same size. Also, it is to be noted that the earlier studies [13, 36, 41] were based on simulation whereas the present work incorporates actual surface data using CMM.

In another simulated manufacturing surface (that included milling), Woo et al. [93] applied the Hammersley and Halton-Zaremba sequences in 2-D sampling and found no discernible difference in their performance. If the present end-milled plates are measured with Hammersley sequence of 10 points and Halton-Zaremba sequence of 16 points, the average error of four plates will be 39.7 % and 46.2 %, respectively (note in the present study, 12 points yields 37.2 % error and 20 points gives 29.6 % error). In contrast to the finding in [93], Halton-Zaremba sampling performed very poorly in comparison to Hammersley sampling. This discrepancy is because they modeled a random surface while this research has used actual fabricated plates. Further, any sampling not using the knowledge of the actual surface pattern may on occasion catch the approximate error locations with a reduced sample size (like Hammersley) and at other

times may not (like Halton-Zaremba in this case) even with a larger sample size. The present procedure at the very least provides proof for this experimentally. Thus, the manufacturing error patterns are extremely important in sample point selection.

In this work, tolerance h_t is obtained by multiplying ($|e_{max(+)}| + |e_{max(-)}|$) by $[1/(1 + l_0^2)]^{1/2}$ for straightness and by $[1/(1 + l_0^2 + m_0^2)]^{1/2}$ for flatness, respectively to account for normal distance as suggested in [59, 87]. But the values are similar to what would be obtained from ($|e_{max(+)}| + |e_{max(-)}|$). This is in greement with that given in [59, 78] that the zone obtained from the linear (e_i) and normal deviations ($\hat{e}_i$) are not significantly different when the errors are of minute degree.

Based on the results discussed and comparisons with other related work made in this chapter, concluding remarks are given in the next chapter. It is to be noted that in this study, the surface error profiles have been measured to verify the surface error models given in [48] and [35] for end and face milling respectively. One must ideally predict the errors with these suggested mathematical models and use that to guide the initial point selection, now that their feasibility has been experimentally verified. Then use the search methods to sample additional points.

CHAPTER 9

CONCLUSIONS AND RECOMMENDATIONS

An intelligent methodology for sample reduction, using optimization search heuristics and manufacturing surface error profiles, is presented here for form tolerance verification in coordinate metrology. It is shown that a high accuracy in sampling can be achieved through a prior characterization of manufacturing errors on the part being measured. Error profile models given in [48] and [35] for end and face milling respectively are experimentally verified. The end milling profile obtained is in good agreement with the model. The face milling errors do not agree completely with the profile presented in [35] because of the difference in the tool geometry and spindle axis tilt. However, both models can be utilized to predict the errors. The surface errors dictate initial locations to sample. Next points to inspect are determined using search methods looking for improvements in the tolerance zone estimated by the linear least squares technique. The methodology is demonstrated with examples of straightness and flatness tolerance. For straightness estimation, region-elimination search is used. For flatness determination, tabu search and hybrid search are employed.

In case of straightness evaluation, it is observed that small step size increases the sample size with no significant effect on the accuracy of the estimation. The lowest sample size corresponding to step size of 0.2 inch is found to be 5 and 7 respectively for end and face milling. This yields a zone having an average accuracy of 92.6 % with respect to the zone obtained from the population sample. Flatness estimation with strategy RS0BM1 (i.e., no re-start and one bad move) requires a sample size of 12 and

results in an accuracy of 62.8 and 88.1 % for end and face milling respectively without losing significant accuracy in comparison with RS0BM3 and RS0BM5. Strategy RS1BM1 (i.e., one re-start) increases the sample size to 20 but increases the accuracy as well to 70.4 and 89.3 %, respectively. Concerning the two search methods employed in flatness, although hybrid search results in a significantly smaller sample size for the same accuracy, it is practically not much different from tabu search. Hence, it is a matter of convenience in choosing between the two algorithms. Comparison of the present sampling procedure with other methods reveals that this is more efficient and reliable. Also, unlike the findings in [93], Halton-Zaremba sequence performed very poorly against Hammersely sequence for the end-milled surface (flatness determination).

Based on the findings of this dissertation, it is suggested to exploit the knowledge of the manufacturing process used to produce an object surface in part feature verification. This can be done utilizing the surface error profile to sample initial data points. In the literature, there exist manufacturing error models for different processes, which can be used. If for any manufacturing process, there is no previous model available, then the surface error pattern can be quantified experimentally for one part and this pattern can be used for other parts produced from the same process. Once a set of initial data points is selected and corresponding form tolerance is computed, additional data points can be sampled using a suitable search method in order to improve the tolerance zone. In case of straightness estimation, region-elimination search method can be employed and for flatness evaluation, tabu search or hybrid search method can be used.

In the future, the approach can be extended for the estimation of other form tolerances (circularity and cylindricity). Since coordinate metrology applies to die casting, cold

forming, machining, and grinding domains of tolerances, these processes may be suitably characterized for their surface error patterns.

In this work, the least squares (LS) technique is employed to compute the tolerance zone. The minimum zone (MZ) method can be used in the future. LS over-estimates and MZ under-estimates the corresponding zone, if the sample size is small. Therefore, other techniques (e.g., Huber loss statistics [38]) should be incorporated to improve the zone estimation.

As explained in Chapter 4 and illustrated with an example on each search method in Chapter 6, from a starting point, the search goes to all the allowed neighborhood points and evaluates them. Then it selects the 'best' or the 'least bad' neighborhood point resulting a final move ('good' or 'bad') of the particular iteration. This point is referred to as sampled point. This way, the search continues. In the region-elimination algorithm for straightness verification, if the neighborhood points are not good, then instead of making a 'bad' move, the search stays at the starting point and the interval is reduced for the next iteration. Thus, the total number of sampled points includes the initial points and the points sampled during the search. The other evaluated points are not considered because they do not take part in the tolerance zone improvement and hence, are not stored for the next iteration. The search, however, does take some time to go to such points and evaluate them. Therefore, total inspection time should include the time to measure the initial data points, the sampled points, and the evaluated points. In the future, the search algorithms can be integrated with the measurement software of CMMs to determine the total measurement time.

Further, length of the CMM probe path must also be considered. CMM measurement uncertainty data can also be included in the analysis to further enhance the accuracy of the zone estimation.

REFERENCES

1. Allada, V. and Anand, S., 1994, "Computer-Aided Inspection Using Hough Transform," IIE *3rd Industrial Engineering Research Conf. Proceedings*, Atlanta, May 18-19, pp. 40-45.
2. Al-Sultan, K.S. and Al-Fawzan, M.A., 1997, "A Tabu Search Hooke and Jeeves Algorithm for Unconstrained Optimization," *European J. of Operational Research*, Vol. 103, pp. 198-208.
3. ASME Y14.5M-1994, *Dimensioning and Tolerancing*, The American Society of Mechanical Engineers, New York, NY.
4. Audet, C. and Dennis Jr., J.E., 2000, "A Pattern Search Filter Method for Nonlinear Programming without Derivatives," TR00-09, Dept. of Computational & Applied Mathematics, Rice Univ., Houston, TX.
5. Audet, C. and Dennis Jr., J.E., 2000, "Analysis of Generalized Pattern Searches," TR00-07, Dept. of Computational & Applied Mathematics, Rice Univ., Houston, TX.
6. Babu, U., Raja, J., and Hocken, R.J., 1993, "Sampling Methods and Substitute Geometry Algorithms for Measuring Cylinders in Coordinate Measuring Machines," *Proceedings of the Eighth Annual Meeting*, American Society for Precision Engineering, pp. 70-73.
7. Badar, M.A., Raman, S., and Pulat, P.S., 2000, "Search-Based Selection of Sample Points for Form Error Estimation," ASME *Proceedings of the ASME Manufacturing Engineering Division - 2000*, R.J. Furness, ed, MED–Vol. 11, pp. 73-80.
8. Badar, M.A., Raman, S., and Pulat, P.S., 2002, "Intelligent Search-Based Selection of Sample Points for Straightness and Flatness Estimation," ASME *J. of Manufacturing Science and Engineering*, accepted.
9. Badiru, K., 1999, "A Tabu-Search-Based Algorithm for Surface Inspection," MS Thesis (Advisor: P. Simin Pulat), Univ. of Oklahoma, Norman, OK.
10. Balasubramanian, N. and Raman, S., 1998, "Modeling Gradual Process Variables in Path Planning," Technical Paper – Society of Manufacturing Engineers, MR, Vol. MR98-17, SME, Dearborn, MI.
11. Carr, K. and Ferreira, P., 1995a, "Verification of Form Tolerances Part I: Basic Issues, Flatness, and Straightness," *Precision Engineering*, Vol. 17, No. 2, pp. 131-143.
12. Carr, K. and Ferreira, P., 1995b, "Verification of Form Tolerances Part II: Cylindricity and Straightness of a Median Line," *Precision Engineering*, Vol. 17, No. 2, pp. 144-156.
13. Caskey, G., Hari, Y., Hocken, R., Machireddy, R., Raja, J., Wilson, R., Zhang, G., Chen, K., and Yang, J., 1992, "Sampling Techniques for Coordinate Measuring Machines," *Proceedings of the 1992 NSF Design and Manufacturing Systems Conf.*,

Atlanta, GA, Jan 8-10, pp. 983-988.

14. Chang, H. and Lin, T.W., 1993, "Evaluation of Circularity Tolerance Using Monte Carlo Simulation for Coordinate Measuring Machine," *Int. J. of Production Research*, Vol. 31, No. 9, pp. 2079-2086.

15. Chen, M.-C. and Tsai, D.-M., 1996, "Simulation Optimization through Direct Search for Multi-Objective Manufacturing Systems," *Production Planning & Control*, Vol. 7, No. 6, pp. 554-565.

16. Cochran, W.G., 1977, *Sampling Techniques*, 3rd ed, John Wiley & Sons, Inc., New York, NY.

17. Coy, J., 1990, "Sampling Error for Co-ordinate Measurement," *Proceedings of the 28th Int. Matador Conf.*, Manchester, UK, pp. 481-489.

18. Damodaran, P., Fernando, S., and Anand, S., 1996, "A Comparative Evaluation of Algorithms Used for the Determination of Roundness Tolerance," IIE *5th Industrial Engineering Research Conf. Proceedings*, Minneapolis, May 18-20, pp. 339-344.

19. Damodarasamy, S. and Anand, S., 1999, "Evaluation of Minimum Zone for Flatness by Normal Plane Method and Simplex Search," *IIE Transactions*, Vol. 31, No. 7, pp. 617-626.

20. Devor, R.E., Kline, W.A., and Zdeblick, W.J., 1980, "A Mechanistic model for the Force System in End Milling with Application to Machining Airframe Structures," *Manufacturing Engineering Transactions*, Vol. 8, (8th North American Mfg. Research Conf. Proceedings, May 18-21), pp. 297-303.

21. Dowling, M.M., Griffin, P.M., Tsui, K.-L., and Zhou, C., 1995, "A Comparison of the Orthogonal Least Squares and Minimum Enclosing Zone Methods for Form Error Estimation," ASME *Manufacturing Review*, Vol. 8, No. 2, pp. 120-138.

22. Dowling, M.M., Griffin, P.M., Tsui, K.-L., and Zhou, C., 1997, "Statistical Issues in Geometric Feature Inspection Using Coordinate Measuring Machines," ASA & ASQC *Technometrics*, Vol. 39, No. 1, pp. 3-17.

23. Drozda, T.J. and Wick, C., 1983, *Tool & Manufacturing Engineers Handbook*, Vol.1 Machining, 4th edn, SME, Dearborn, MI.

24. Edgeworth, R. and Wilhelm, R.G., 1996, "Uncertainty Management for CMM Probe Sampling of Complex Surfaces," ASME *Manufacturing Science and Engineering*, MED-Vol. 4, pp. 511-518.

25. Elmaraghy, W.H., Elmaraghy, H.A., and Wu, Z., 1990, "Determination of Actual Geometric Deviations Using Coordinate Measuring Machine Data," ASME Manufacturing Review, Vol. 3, No.1, pp. 32-38.

26. Etesami, F., 1988, "Tolerance Verification through Manufactured Part Modeling," *J. of Manufacturing Systems*, Vol. 7, No. 3, pp. 223-232.

27. Etesami, F., 1990, "Analysis of Two-Dimensional Measurement Data for Automated Inspection," *J. of Manufacturing Systems*, Vol. 9, No. 1, pp. 21-34.

28. Fu, H.J., DeVor, R.E., and Kapoor, S.G., 1984, "A Mechanistic Model for the Prediction of the Force System in Face Milling Operations," ASME *J. of Engineering for Industry*, Vol. 106, No. 1, pp. 81-88.

29. Fukuda, M. and Shimokohbe, A., 1984, "Algorithms for Form Error Evaluation – Methods of the Minimum Zone and the Least Squares," *Proceedings of the Int. Symp. On Metrology for Quality Control in Production*, Tokyo, Japan, pp. 197-202.

30. Glover, F. and Laguna, M., 1993, "Tabu Search," *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves (ed.), John Wiley & Sons, Inc., New York, NY, pp. 70-150.

31. Glover, F. and Laguna, M., 1997, *Tabu Search*, Kluwer Academic Publishers, Boston, MA.

32. Glover, F., 1989, "Tabu Search – Part I," *ORSA J. on Computing*, Vol. 1, No. 3, pp. 190-206.

33. Glover, F., 1990a, "Tabu Search – Part II," *ORSA J. on Computing*, Vol. 2, No. 1, pp. 4-32.

34. Glover, F., 1990b, "Tabu Search: A Tutorial," *Interfaces*, Vol. 20, No. 4, pp. 74-94.

35. Gu, F., Melkote, S.N., Kapoor, S.G., and DeVor, R.E., 1997, "A Model for the Prediction of Surface Flatness in Face Milling," ASME *J. of Manufacturing Science and Engineering*, Vol. 119, Nov., pp. 476-484.

36. Hocken, R.J., Raja, J., and Babu, U., 1993, "Sampling Issues in Coordinate Metrology," ASME *Manufacturing Review*, Vol. 6, No. 4, pp. 282-294.

37. Hopp, T.H., 1993, "Computational Metrology," ASME *Manufacturing Review*, Vol. 6, No. 4, pp. 295-304.

38. Huber, P.J., 1996, *Robust Statistical Procedures*, 2nd edn, SIAM, Philadelphia, PA.

39. Hulting, F.L., 1992, "Methods for the Analysis of Coordinate Measurement Data," *Computing Science and Statistics*, Vol. 24, pp. 160-169.

40. Hulting, F.L., 1995, "Comment: An Industry View of Coordinate Measurement Data Analysis," *Statistica Sinica*, Vol. 5, pp. 191-204.

41. Hurt, J.J., 1980, "A Comparison of Several Plane Fit Algorithms," *Annals of the CIRP*, Vol. 29, No. 1, pp. 381-384.

42. Ismail, F., Elbestawi, M.A., Du, R., and Urbasik, K., 1993, "Generation of Milled Surfaces Including Tool Dynamics and Wear," ASME *J. of Engineering for Industry*, Vol. 115, No. 2, pp. 245-252.

43. Kalpakjian, S., 1995, *Manufacturing Engineering and Technology*, 3rd edn, Addison-Wesley, New York, NY.

44. Kanada, T. and Suzuki, S., 1993, "Evaluation of Minimum Zone Flatness by Means of Nonlinear Optimization Techniques and Its Verification," *Precision Engineering*, Vol. 15, No. 2, pp. 93-99.

45. Kim, H.S. and Ehmann, K.F., 1993, "A Cutting Force Model for Face Milling Operations," *Int. J. of Machine Tools & Manufacture*, Vol. 33, No. 5, pp. 651-673.

46. Kim, W.-S. and Raman, S., 2000, "On the Selection of Flatness Measurement Points in Coordinate Measuring Machine Inspection," *Int. J. of Machine Tools & Manufacture*, Vol. 40, pp. 427-443.

47. Kim, W.-S., 1998, "An Investigation of Sampling Strategies in Flatness Inspection," MS Thesis (Advisor: Shivakumar Raman), Univ. of Oklahoma, Norman, OK.

48. Kline, W. A., DeVor, R.E., and Shareef, I.A., 1982, "The Prediction of Surface Accuracy in End Milling," ASME *J. of Engineering for Industry*, Vol. 104, No. 3, pp. 272-278.

49. Kolahan, F. and Liang, M., 2000, "Optimization of Hole-Making Operations: a Tabu-search Approach," *Int. J. of Machine Tools & Manufacture*, Vol. 40, No. 12, pp. 1735-1753.

50. Kunzmann, H. and Waldele, F., 1988, "Performance of CMMs," *Annals of the CIRP*, Vol. 37, No. 2, pp. 633-640.

51. Kurfess, T.R. and Banks, D.L., 1995, "Statistical Verification of Conformance to Geometric Tolerance," *Computer-Aided Design*, Vol. 27, No. 5, pp. 353-361.

52. Lee, G., Mou, J., and Shen, Y., 1997, "Sampling Strategy Design for Dimensional Measurement of Geometric Features Using Coordinate Measuring Machine," *Int. J. of Machine Tools & Manufacture*, Vol. 37, No. 7, pp. 917-934.

53. Lewis, R.M. and Torczon, V., 1999, "Pattern Search Algorithms for Bound Constrained Minimization," SIAM *J. on Optimization*, Vol. 9, No. 4, pp. 1082-1099.

54. Lewis, R.M. and Torczon, V., 2000, "Pattern Search Methods for Linearly Constrained Minimization," SIAM *J. on Optimization*, Vol. 10, No. 3, pp. 917-941.

55. McKay, M.D., Beckman, R.J., and Conover, W.J., 1979, "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," ASA & ASQC *Technometrics*, Vol. 21, No. 2, pp. 239-245.

56. Menq, C.-H., Yau, H.-T., Lai, G.-Y., and Miller, R.A., 1990, "Statistical Evaluation of Form Tolerances Using Discrete Measurement Data," ASME *Advances in Integrated Product Design and Manufacturing*, P.H. Cohen and S.B. Joshi, ed, PED–Vol. 47, pp. 135-149.

57. Montgomery, D. and Altintas, Y., 1991, "Mechanism of Cutting Force and Surface Generation in Dynamic Milling," ASME *J. of Engineering for Industry*, Vol. 113, No. 2, pp. 160-168.

58. Montgomery, D.C., 1991, *Design and Analysis of Experiments*, 3rd edn, John Wiley & Sons, Inc., New York, NY.

59. Murthy, T.S.R. and Abdin, S.Z., 1980, "Minimum Zone Evaluation of Surfaces," *Int. J. of Machine Tool Design & Research*, Vol. 20, No. 2, pp. 123-136.

60. Myers, R.H. and Montgomery, D.C., 1995, *Response Surface Methodology: Process*

and Product Optimization Using Designed Experiments, John Wiley & Sons, Inc., New York, NY.

61. Namboothiri, V.N.N. and Shunmugam, M.S., 1998a, "Form Error Evaluation Using L_1 – Approximation," *Computer Methods in Applied Mechanics and Engineering*, Vol. 162, No. 1-4, pp. 133-149.

62. Namboothiri, V.N.N. and Shunmugam, M.S., 1998b, "Function-Oriented Form Evaluation of Engineering Surfaces," *Precision Engineering*, Vol. 22, No. 2, pp. 98-109.

63. Namboothiri, V.N.N. and Shunmugam, M.S., 1999, "On Determination of Sample Size in Form Error Evaluation Using Coordinate Metrology," *Int. J. of Production Research*, Vol. 37, No. 4, pp. 793-804.

64. Orady, E., Li, S., and Chen, Y., 1996b, "A Fitting Algorithm for Determination of Minimum Zone Form Tolerances," *SAE Transactions*, Vol. 105, Sec. 6, pp. 1502-1508.

65. Orady, E.A., Li, S., and Chen, Y., 1996a, "Evaluation of Minimum Zone Straightness by Nonlinear Optimization Method," ASME *Manufacturing Science and Engineering*, MED-Vol. 4, pp. 413-423.

66. Pham, T.D. and Karaboga, D., 2000, *Intelligent Optimisation Techniques*, Springer-Verlag, London, UK.

67. Prakasvudhisarn, C., 2002, "Dimensional Measurement of Conical Features Using Coodinate Metrology," PhD Dissertation (Advisor: Shivakumar Raman), Univ. of Oklahoma, Norman, OK.

68. Radulescu, R., Kapoor, S.G., and DeVor, R.E., 1997, "An Investigation of Variable Spindle Speed Face Milling for Tool-Work Structures With Complex Dynamics, Part 2: Physical Explanation," ASME *J. of Manufacturing Science and Engineering*, Vol. 119, No. 3, pp. 273-280.

69. Rayward-Smith, V.J., Osman, I.H., Reeves, C.R., and Smith, G.D., 1996, *Modern Heuristic Search Methods*, John Wiley & Sons Ltd, Chichester, UK.

70. Reklaitis, G.V., Ravindran, A., and Ragsdell, K.M., 1983, *Engineering Optimization: Methods and Applications*, John Wiley & Sons, Inc., New York, NY.

71. Requicha, A.A.G., 1993, "Mathematical Definition of Tolerance Specifications," ASME *Manufacturing Review*, Vol. 6, No. 4, pp. 269-274.

72. Roy, U. and Xu, Y., 1994, "Form and Orientation Tolerance Analysis for Cylindrical Surfaces in Computer Aided Inspection," IIE *3rd Industrial Engineering Research Conf. Proceedings*, Atlanta, May 18-19, pp. 46-51.

73. Roy, U. and Zhang, X., 1992, "Establishment of a Pair of Concentric Circles with the Minimum Radial Separation for Assessing Roundness Error," *Computer-Aided Design*, Vol. 24, No. 3, pp. 161-168.

74. Roy, U., 1995, "Computational Methodologies for Evaluating form and Positional

Tolerances in a Computer Integrated Manufacturing System," *The Int. J. of Adv. Manufacturing Technology*, Vol. 10, pp. 110-117.

75. Shao, J. and Tsui, K.-L., 1996, "Form Tolerance Estimation Using Jackknife Methods," ASME *Manufacturing Science and Engineering*, MED-Vol. 4, pp. 433-445.

76. Shen, Y. and Duffie, N.A., 1995, "An Uncertainty Analysis Method for Coordinate Referencing in Manufacturing Systems," ASME *J. of Engineering for Industry*, Vol. 117, No. 1, pp. 42-48.

77. Shunmugam, M.S., 1986, "On Assessment of Geometric Errors," *Int. J. of Production Research*, Vol. 24, No. 2, pp. 413-425.

78. Shunmugam, M.S., 1987a, "Comparison of Linear and Normal Deviations of Forms of Engineering Surfaces," *Precision Engineering*, Vol. 9, No. 2, pp. 96-102.

79. Shunmugam, M.S., 1987b, "New Approach for Evaluating Form Errors of Engineering Surfaces," *Computer-Aided Design*, Vol. 19, No. 7, pp. 368-374.

80. Shunmugam, M.S., 1991a, "Criteria for Computer-Aided Form Evaluation," ASME *J. of Engineering for Industry*, Vol. 113, No. 2, pp. 233-238.

81. Shunmugam, M.S., 1991b, "Establishing Reference Figures for Form Evaluation of Engineering Surfaces," *J. of Manufacturing Systems*, Vol. 10, No. 4, pp. 314-321.

82. Skorin-Kapov, J., 1990, "Tabu Search Applied to the Quadratic Assignment Problem," *ORSA J. on Computing*, Vol. 2, No. 1, pp. 33-45.

83. Stein, M., 1987, "Large Sample Properties of Simulations Using Latin Hypercube Sampling," ASA & ASQC Technometrics, Vol. 29, No. 2, pp. 143-151.

84. Sutherland, J.W. and DeVor, R.E., 1986, "An Improved Method for Cutting Force and Surface Error Prediction in Flexible End Milling Systems," ASME *J. of Engineering for Industry*, Vol. 108, No. 4, pp. 269-279.

85. Sweet, A.L., Noller, D., and Lee, S.-H., 1985, "Statistical Design for the Location of Planes and Circles When Using a Probe," Precision Engineering, Vol. 7, No. 4, pp. 187-194.

86. Torczon, V., 1997, "On the Convergence of Pattern Search Algorithms," SIAM *J. on Optimization*, Vol. 7, No. 1, pp. 1-25.

87. Traband, M.T., Joshi, S., Wysk, R.A., and Cavalier, T.M., 1989, "Evaluation of Straightness and Flatness Tolerances Using the Minimum Zone," ASME *Manufacturing Review*, Vol. 2, No. 3, pp. 189-195.

88. Wang, Y., 1992, "Minimum Zone Evaluation of Form Tolerances," ASME *Manufacturing Review*, Vol. 5, No. 3, pp. 213-220.

89. Weckenmann, A., Eitzert, H., Garmer, M., and Weber, H., 1995, "Functionality-Oriented Evaluation and Sampling Strategy in Coordinate Metrology," *Precision Engineering*, Vol. 17, No. 4, pp. 244-252.

90. Weckenmann, A., Heinrichowski, M., and Mordhorst, H.-J, 1991, "Design of Gauges and Multipoint Measuring Systems Using Coordinate-Measuring-Machine Data and Computer Simulation," *Precision Engineering*, Vol. 13, No. 3, pp. 203-207.

91. Woo, T.C. and Liang, R., 1993, "Dimensional Measurement of Surfaces and Their Sampling," *Computer-Aided Design*, Vol. 25, No. 4, pp. 233-239.

92. Woo, T.C., Liang, R., and Pollock, S.M., 1993, "Hammersley Sampling for Efficient Surface Coordinate Measurements," *Proceedings of the 1993 NSF Design and Manufacturing Systems Conf.*, Charlotte, NC, Jan 6-8, pp. 1489-1495.

93. Woo, T.C., Liang, R., Hsieh, C.C., and Lee, N.K., 1995, "Efficient Sampling for Surface Measurements," *J. of Manufacturing Systems*, Vol. 14, No. 5, pp. 345-354.

94. Yau, H.-T. and Menq, C.-H., 1992, "An Automated Dimensional Inspection Environment for Manufactured Parts Using Coordinate Measuring Machines," *Int. J. of Production Research*, Vol. 30, No. 7, pp. 1517-1536.

95. Yau, H.-T. and Menq, C.-H., 1996, "A Unified Least-Squares Approach to the Evaluation of Geometric Errors Using Discrete Measurement Data," *Int. J. of Machine Tools & Manufacture*, Vol. 36, No. 11, pp. 1269-1290.

96. Yau, H.-T., 1998, "Uncertainty Analysis in Geometric Best Fit," *Int. J. of Machine Tools & Manufacture*, Vol. 38, No. 10-11, pp. 1323-1342.

97. You, S.J. and Ehmann, K.F., 1991, "Synthesis and Generation of Surfaces Milled by Ball Nose End Mills Under Tertiary Cutter Motion," ASME *J. of Engineering for Industry*, Vol. 113, No. 1, pp. 17-24.

98. Zhang, G.M. and Kapoor, S.G., 1991a, "Dynamic Generation of Machined Surfaces, Part 1: Description of a Random Excitation System," ASME *J. of Engineering for Industry*, Vol. 113, No. 2, pp. 137-144.

99. Zhang, G.M. and Kapoor, S.G., 1991b, "Dynamic Generation of Machined Surfaces, Part 2: Construction of Surface Topography," ASME *J. of Engineering for Industry*, Vol. 113, No. 2, pp. 145-153.

100. Zhang, Y.F., Nee, A.Y.C., Fuh, J.Y.H., Neo, K.S., and Loy, H.K., 1996, "A Neural Network Approach to Determining Optimal Inspection Sampling Size for CMM," *Computer-Integrated Manufacturing Systems*, Vol. 9, No. 3, pp. 161-169.

101. Ehmann, K.F., Kapoor, S.G., DeVor, R.E., and Lazoglu, I., 1997, "Machining Process Modeling: a Review," ASME *J. of Manufacturing Science and Engineering*, Vol. 119, pp. 655-663.

102. Tsai, J.-S. and Liao, C.-L., 2000, "Dynamic Finite Element Modeling of Surface Errors in Peripheral Milling of Thin-Walled Workpieces," *J. of the Chinese Society of Mechanical Engineers*, Vol. 21, No. 3, pp. 265-282.

APPENDIX A

POPULATION DATA

A.1 Straightness

A.1.1 End Milling

Table A.1.1.1: Plate #4 (For the population, $N = 121$, $l_0 = -0.001030$, $z_0 = -14.84799$, $h_t = 0.005475$ inch).

n	x	z	n	x	z	n	x	z
0	4.0	-14.85459	1	4.05	-14.85511	2	4.1	-14.85474
3	4.15	-14.85458	4	4.2	-14.85446	5	4.25	-14.8544
6	4.3	-14.85416	7	4.35	-14.85468	8	4.4	-14.85376
9	4.45	-14.85455	10	4.5	-14.85415	11	4.55	-14.85374
12	4.6	-14.85363	13	4.65	-14.85358	14	4.7	-14.85376
15	4.75	-14.85404	16	4.8	-14.85352	17	4.85	-14.85417
18	4.9	-14.85422	19	4.95	-14.85383	20	5	-14.85385
21	5.05	-14.85348	22	5.1	-14.8538	23	5.15	-14.85329
24	5.2	-14.85333	25	5.25	-14.85341	26	5.3	-14.8536
27	5.35	-14.85407	28	5.4	-14.85337	29	5.45	-14.85333
30	5.5	-14.85295	31	5.55	-14.85303	32	5.6	-14.85365
33	5.65	-14.85361	34	5.7	-14.85324	35	5.75	-14.85325
36	5.8	-14.85285	37	5.85	-14.85357	38	5.9	-14.85348
39	5.95	-14.85349	40	6	-14.85361	41	6.05	-14.85302
42	6.1	-14.85355	43	6.15	-14.85335	44	6.2	-14.8534
45	6.25	-14.85307	46	6.3	-14.85363	47	6.35	-14.85297
48	6.4	-14.85358	49	6.45	-14.8531	50	6.5	-14.85378
51	6.55	-14.85346	52	6.6	-14.85303	53	6.65	-14.85306
54	6.7	-14.85376	55	6.75	-14.85343	56	6.8	-14.85405
57	6.85	-14.85302	58	6.9	-14.85371	59	6.95	-14.854
60	7	-14.85396	61	7.05	-14.85343	62	7.1	-14.85431
63	7.15	-14.85431	64	7.2	-14.85365	65	7.25	-14.85368
66	7.3	-14.85398	67	7.35	-14.85377	68	7.4	-14.85426
69	7.45	-14.85397	70	7.5	-14.85463	71	7.55	-14.85395
72	7.6	-14.85392	73	7.65	-14.8551	74	7.7	-14.8552
75	7.75	-14.85435	76	7.8	-14.85494	77	7.85	-14.85439
78	7.9	-14.85479	79	7.95	-14.85546	80	8	-14.85552
81	8.05	-14.85496	82	8.1	-14.85526	83	8.15	-14.85504
84	8.2	-14.85554	85	8.25	-14.856	86	8.3	-14.85545
87	8.35	-14.85619	88	8.4	-14.85633	89	8.45	-14.85672
90	8.5	-14.8556	91	8.55	-14.85644	92	8.6	-14.8568
93	8.65	-14.85645	94	8.7	-14.85659	95	8.75	-14.85705

96	8.8	-14.85733	97	8.85	-14.85688	98	8.9	-14.85733
99	8.95	-14.85806	100	9	-14.85741	101	9.05	-14.85778
102	9.1	-14.85791	103	9.15	-14.85778	104	9.2	-14.85873
105	9.25	-14.85769	106	9.3	-14.85808	107	9.35	-14.85885
108	9.4	-14.85963	109	9.45	-14.85933	110	9.5	-14.85973
111	9.55	-14.85938	112	9.6	-14.85945	113	9.65	-14.86033
114	9.7	-14.85967	115	9.75	-14.8607	116	9.8	-14.8607
117	9.85	-14.86047	118	9.9	-14.86091	119	9.95	-14.86115
120	10	-14.86174						

Table A.1.1.2: Plate #12 ($N = 121$, $l_0 = -0.000979$, $z_0 = -14.8273$, $h_t = 0.005333$).

n	x	z	n	x	z	n	x	z
0	4.0	-14.83429	1	4.05	-14.83445	2	4.1	-14.83432
3	4.15	-14.83436	4	4.2	-14.83333	5	4.25	-14.834
6	4.3	-14.83382	7	4.35	-14.83359	8	4.4	-14.83285
9	4.45	-14.83349	10	4.5	-14.83379	11	4.55	-14.83354
12	4.6	-14.83277	13	4.65	-14.83324	14	4.7	-14.83243
15	4.75	-14.83258	16	4.8	-14.83306	17	4.85	-14.8332
18	4.9	-14.8325	19	4.95	-14.83296	20	5	-14.83222
21	5.05	-14.8326	22	5.1	-14.83218	23	5.15	-14.83273
24	5.2	-14.83236	25	5.25	-14.83243	26	5.3	-14.8319
27	5.35	-14.83189	28	5.4	-14.83245	29	5.45	-14.83246
30	5.5	-14.83187	31	5.55	-14.83228	32	5.6	-14.83271
33	5.65	-14.83177	34	5.7	-14.83219	35	5.75	-14.83234
36	5.8	-14.83236	37	5.85	-14.83224	38	5.9	-14.83246
39	5.95	-14.83167	40	6	-14.83232	41	6.05	-14.83255
42	6.1	-14.83226	43	6.15	-14.83197	44	6.2	-14.83193
45	6.25	-14.83173	46	6.3	-14.83195	47	6.35	-14.83194
48	6.4	-14.83247	49	6.45	-14.83237	50	6.5	-14.83189
51	6.55	-14.83209	52	6.6	-14.83224	53	6.65	-14.83199
54	6.7	-14.83258	55	6.75	-14.83263	56	6.8	-14.8326
57	6.85	-14.83272	58	6.9	-14.83204	59	6.95	-14.83283
60	7	-14.83257	61	7.05	-14.83312	62	7.1	-14.83324
63	7.15	-14.83258	64	7.2	-14.83246	65	7.25	-14.83225
66	7.3	-14.83307	67	7.35	-14.83276	68	7.4	-14.83316
69	7.45	-14.83371	70	7.5	-14.83372	71	7.55	-14.83336
72	7.6	-14.83396	73	7.65	-14.83386	74	7.7	-14.83353
75	7.75	-14.83338	76	7.8	-14.83408	77	7.85	-14.83394
78	7.9	-14.83445	79	7.95	-14.83358	80	8	-14.83388
81	8.05	-14.83423	82	8.1	-14.835	83	8.15	-14.83491
84	8.2	-14.835	85	8.25	-14.8347	86	8.3	-14.83494
87	8.35	-14.83551	88	8.4	-14.8348	89	8.45	-14.83526
90	8.5	-14.83557	91	8.55	-14.83613	92	8.6	-14.83552
93	8.65	-14.83516	94	8.7	-14.83533	95	8.75	-14.83609
96	8.8	-14.83634	97	8.85	-14.83621	98	8.9	-14.83604

99	8.95	-14.83594	100	9	-14.83609	101	9.05	-14.8367
102	9.1	-14.8362	103	9.15	-14.83707	104	9.2	-14.83673
105	9.25	-14.83704	106	9.3	-14.83746	107	9.35	-14.83758
108	9.4	-14.8383	109	9.45	-14.83757	110	9.5	-14.83851
111	9.55	-14.83769	112	9.6	-14.8389	113	9.65	-14.83901
114	9.7	-14.83926	115	9.75	-14.83929	116	9.8	-14.83887
117	9.85	-14.83892	118	9.9	-14.83947	119	9.95	-14.8401
120	10	-14.8402						

Table A.1.1.3: Plate #9 ($N = 121$, $l_0 = -0.000835$, $z_0 = -14.85147$, $h_t = 0.004047$).

n	*x*	*z*	*n*	*x*	*z*	*n*	*x*	*z*
0	4.0	-14.85701	1	4.05	-14.85691	2	4.1	-14.85713
3	4.15	-14.85677	4	4.2	-14.85694	5	4.25	-14.85666
6	4.3	-14.85669	7	4.35	-14.85705	8	4.4	-14.85688
9	4.45	-14.85679	10	4.5	-14.85683	11	4.55	-14.8568
12	4.6	-14.85601	13	4.65	-14.85613	14	4.7	-14.85623
15	4.75	-14.85635	16	4.8	-14.8564	17	4.85	-14.85639
18	4.9	-14.85636	19	4.95	-14.85595	20	5	-14.85624
21	5.05	-14.85588	22	5.1	-14.85628	23	5.15	-14.85586
24	5.2	-14.85566	25	5.25	-14.85578	26	5.3	-14.85602
27	5.35	-14.85593	28	5.4	-14.85552	29	5.45	-14.85552
30	5.5	-14.85553	31	5.55	-14.85604	32	5.6	-14.85554
33	5.65	-14.8557	34	5.7	-14.85544	35	5.75	-14.85532
36	5.8	-14.85563	37	5.85	-14.85537	38	5.9	-14.85569
39	5.95	-14.85574	40	6	-14.85573	41	6.05	-14.85564
42	6.1	-14.85594	43	6.15	-14.85529	44	6.2	-14.856
45	6.25	-14.85599	46	6.3	-14.8555	47	6.35	-14.85584
48	6.4	-14.8557	49	6.45	-14.8554	50	6.5	-14.85572
51	6.55	-14.8557	52	6.6	-14.85604	53	6.65	-14.85597
54	6.7	-14.85586	55	6.75	-14.85593	56	6.8	-14.85626
57	6.85	-14.85585	58	6.9	-14.85631	59	6.95	-14.85614
60	7	-14.85608	61	7.05	-14.85633	62	7.1	-14.85625
63	7.15	-14.8564	64	7.2	-14.85593	65	7.25	-14.85623
66	7.3	-14.85629	67	7.35	-14.85631	68	7.4	-14.85617
69	7.45	-14.85687	70	7.5	-14.8567	71	7.55	-14.85642
72	7.6	-14.85656	73	7.65	-14.8565	74	7.7	-14.85677
75	7.75	-14.85689	76	7.8	-14.85705	77	7.85	-14.85741
78	7.9	-14.85723	79	7.95	-14.85705	80	8	-14.85709
81	8.05	-14.85737	82	8.1	-14.85774	83	8.15	-14.85706
84	8.2	-14.85726	85	8.25	-14.85795	86	8.3	-14.85815
87	8.35	-14.85824	88	8.4	-14.85858	89	8.45	-14.85783
90	8.5	-14.85802	91	8.55	-14.85838	92	8.6	-14.85855
93	8.65	-14.85901	94	8.7	-14.8592	95	8.75	-14.85897
96	8.8	-14.85874	97	8.85	-14.85885	98	8.9	-14.85927
99	8.95	-14.85929	100	9	-14.85958	101	9.05	-14.8594

102	9.1	-14.85967	103	9.15	-14.85973	104	9.2	-14.85979
105	9.25	-14.86012	106	9.3	-14.8598	107	9.35	-14.86039
108	9.4	-14.86033	109	9.45	-14.86085	110	9.5	-14.86039
111	9.55	-14.86088	112	9.6	-14.86113	113	9.65	-14.86116
114	9.7	-14.86108	115	9.75	-14.86153	116	9.8	-14.86153
117	9.85	-14.86219	118	9.9	-14.86146	119	9.95	-14.86196
120	10	-14.86207						

Table A.1.1.4: Plate #2 ($N = 121$, $l_0 = -0.00116$, $z_0 = -14.64718$, $h_t = 0.005188$).

n	*x*	*z*	*n*	*x*	*z*	*n*	*x*	*z*
0	4.0	-14.65411	1	4.05	-14.65425	2	4.1	-14.65396
3	4.15	-14.65415	4	4.2	-14.65453	5	4.25	-14.65381
6	4.3	-14.65411	7	4.35	-14.65371	8	4.4	-14.65398
9	4.45	-14.65405	10	4.5	-14.65376	11	4.55	-14.65388
12	4.6	-14.65397	13	4.65	-14.65338	14	4.7	-14.65413
15	4.75	-14.65388	16	4.8	-14.65348	17	4.85	-14.65325
18	4.9	-14.6538	19	4.95	-14.6533	20	5	-14.65378
21	5.05	-14.65333	22	5.1	-14.65372	23	5.15	-14.65352
24	5.2	-14.65313	25	5.25	-14.65311	26	5.3	-14.6528
27	5.35	-14.65296	28	5.4	-14.65343	29	5.45	-14.65298
30	5.5	-14.65295	31	5.55	-14.65369	32	5.6	-14.65276
33	5.65	-14.65355	34	5.7	-14.65335	35	5.75	-14.65354
36	5.8	-14.65311	37	5.85	-14.65264	38	5.9	-14.65263
39	5.95	-14.65335	40	6	-14.65261	41	6.05	-14.65311
42	6.1	-14.65315	43	6.15	-14.65357	44	6.2	-14.65331
45	6.25	-14.6536	46	6.3	-14.65342	47	6.35	-14.65383
48	6.4	-14.65344	49	6.45	-14.65373	50	6.5	-14.65403
51	6.55	-14.65377	52	6.6	-14.6533	53	6.65	-14.65413
54	6.7	-14.65397	55	6.75	-14.65322	56	6.8	-14.65362
57	6.85	-14.654	58	6.9	-14.65354	59	6.95	-14.6535
60	7	-14.65434	61	7.05	-14.65379	62	7.1	-14.65393
63	7.15	-14.65285	64	7.2	-14.65399	65	7.25	-14.65442
66	7.3	-14.65406	67	7.35	-14.65493	68	7.4	-14.65509
69	7.45	-14.65435	70	7.5	-14.65451	71	7.55	-14.65418
72	7.6	-14.65471	73	7.65	-14.6551	74	7.7	-14.65473
75	7.75	-14.65535	76	7.8	-14.65567	77	7.85	-14.65496
78	7.9	-14.6548	79	7.95	-14.65583	80	8	-14.65599
81	8.05	-14.6557	82	8.1	-14.65524	83	8.15	-14.6562
84	8.2	-14.65584	85	8.25	-14.65609	86	8.3	-14.65571
87	8.35	-14.65626	88	8.4	-14.65681	89	8.45	-14.65689
90	8.5	-14.65627	91	8.55	-14.65691	92	8.6	-14.65736
93	8.65	-14.65759	94	8.7	-14.65705	95	8.75	-14.65818
96	8.8	-14.65783	97	8.85	-14.65796	98	8.9	-14.65815
99	8.95	-14.65787	100	9	-14.65777	101	9.05	-14.65819
102	9.1	-14.65857	103	9.15	-14.65829	104	9.2	-14.65841

105	9.25	-14.65911	106	9.3	-14.65964	107	9.35	-14.6589
108	9.4	-14.65935	109	9.45	-14.65984	110	9.5	-14.65989
111	9.55	-14.6599	112	9.6	-14.65984	113	9.65	-14.65997
114	9.7	-14.65987	115	9.75	-14.66017	116	9.8	-14.6609
117	9.85	-14.66117	118	9.9	-14.6605	119	9.95	-14.66055
120	10	-14.66031						

A.1.2 Face Milling

Table A.1.2.1: Plate #3 ($N = 56$, $l_0 = -0.000039$, $z_0 = -14.15268$, $h_t = 0.00241$).

n	*x*	*z*	*n*	*x*	*z*	*n*	*x*	*z*
0	2.00	-14.15425	1	2.05	-14.15435	2	2.10	-14.1543
3	2.15	-14.15409	4	2.20	-14.15403	5	2.25	-14.15413
6	2.30	-14.154	7	3.40	-14.15262	8	3.45	-14.15272
9	3.50	-14.15265	10	3.55	-14.15255	11	3.60	-14.15265
12	3.65	-14.15257	13	3.70	-14.1525	14	3.75	-14.15248
15	3.80	-14.15243	16	3.85	-14.15235	17	3.90	-14.15233
18	3.95	-14.15224	19	4.00	-14.15223	20	4.05	-14.15234
21	4.10	-14.15228	22	4.15	-14.1523	23	4.20	-14.15219
24	4.25	-14.15207	25	4.30	-14.15209	26	4.35	-14.15203
27	4.40	-14.15211	28	5.60	-14.15216	29	5.65	-14.15224
30	5.70	-14.15218	31	5.75	-14.15214	32	5.80	-14.15218
33	5.85	-14.15232	34	5.90	-14.15232	35	5.95	-14.15231
36	6.00	-14.15231	37	6.05	-14.15239	38	6.10	-14.15243
39	6.15	-14.15234	40	6.20	-14.15256	41	6.25	-14.15234
42	6.30	-14.1526	43	6.35	-14.15266	44	6.40	-14.15258
45	6.45	-14.15257	46	6.50	-14.15292	47	6.55	-14.15296
48	6.60	-14.15287	49	7.70	-14.15428	50	7.75	-14.15427
51	7.80	-14.15433	52	7.85	-14.1544	53	7.90	-14.15437
54	7.95	-14.15454	55	8.00	-14.15457			

Table A.1.2.2: Plate #1 ($N = 56$, $l_0 = 0.000041$, $z_0 = -14.15048$, $h_t = 0.001404$).

n	x	z	n	x	z	n	x	z
0	2.00	-14.15101	1	2.05	-14.151	2	2.10	-14.15105
3	2.15	-14.15095	4	2.20	-14.15076	5	2.25	-14.1509
6	2.30	-14.15101	7	3.40	-14.15042	8	3.45	-14.15036
9	3.50	-14.15033	10	3.55	-14.15025	11	3.60	-14.1503
12	3.65	-14.15033	13	3.70	-14.15019	14	3.75	-14.15013
15	3.80	-14.1501	16	3.85	-14.15013	17	3.90	-14.15033
18	3.95	-14.15011	19	4.00	-14.15016	20	4.05	-14.15006
21	4.10	-14.15009	22	4.15	-14.15012	23	4.20	-14.15028
24	4.25	-14.15006	25	4.30	-14.15001	26	4.35	-14.15002
27	4.40	-14.14999	28	5.60	-14.15002	29	5.65	-14.1499

30	5.70	-14.1499	31	5.75	-14.14978	32	5.80	-14.14988
33	5.85	-14.14973	34	5.90	-14.14995	35	5.95	-14.15015
36	6.00	-14.14988	37	6.05	-14.14972	38	6.10	-14.15012
39	6.15	-14.1498	40	6.20	-14.15004	41	6.25	-14.14988
42	6.30	-14.14993	43	6.35	-14.14994	44	6.40	-14.1499
45	6.45	-14.15013	46	6.50	-14.14996	47	6.55	-14.15
48	6.60	-14.15007	49	7.70	-14.15089	50	7.75	-14.15082
51	7.80	-14.15105	52	7.85	-14.15105	53	7.90	-14.15075
54	7.95	-14.15086	55	8.00	-14.15095			

Table A.1.2.3: Plate #11 ($N = 56$, $l_0 = -0.000041$, $z_0 = -14.14394$, $h_t = 0.002382$).

n	x	z	n	x	z	n	x	z
0	2.00	-14.14548	1	2.05	-14.14553	2	2.10	-14.14554
3	2.15	-14.14548	4	2.20	-14.14529	5	2.25	-14.14518
6	2.30	-14.14524	7	3.40	-14.14415	8	3.45	-14.14398
9	3.50	-14.14399	10	3.55	-14.14421	11	3.60	-14.14367
12	3.65	-14.14369	13	3.70	-14.1437	14	3.75	-14.14354
15	3.80	-14.14353	16	3.85	-14.1435	17	3.90	-14.14362
18	3.95	-14.14354	19	4.00	-14.14366	20	4.05	-14.14341
21	4.10	-14.14341	22	4.15	-14.14331	23	4.20	-14.14341
24	4.25	-14.14336	25	4.30	-14.14337	26	4.35	-14.14325
27	4.40	-14.14344	28	5.60	-14.1436	29	5.65	-14.14348
30	5.70	-14.14357	31	5.75	-14.14348	32	5.80	-14.14362
33	5.85	-14.14353	34	5.90	-14.14363	35	5.95	-14.14368
36	6.00	-14.14364	37	6.05	-14.14372	38	6.10	-14.14372
39	6.15	-14.14372	40	6.20	-14.14387	41	6.25	-14.14393
42	6.30	-14.14393	43	6.35	-14.14389	44	6.40	-14.14412
45	6.45	-14.1442	46	6.50	-14.14414	47	6.55	-14.14428
48	6.60	-14.14418	49	7.70	-14.14539	50	7.75	-14.14536
51	7.80	-14.14536	52	7.85	-14.14555	53	7.90	-14.14557
54	7.95	-14.14562	55	8.00	-14.14568			

Table A.1.2.4: Plate #7 ($N = 56$, $l_0 = -0.000022$, $z_0 = -14.1446$, $h_t = 0.001418$).

n	x	z	n	x	z	n	x	z
0	2.00	-14.14549	1	2.05	-14.14549	2	2.10	-14.14531
3	2.15	-14.14552	4	2.20	-14.14528	5	2.25	-14.14531
6	2.30	-14.14537	7	3.40	-14.14467	8	3.45	-14.14469
9	3.50	-14.14457	10	3.55	-14.14443	11	3.60	-14.14451
12	3.65	-14.1446	13	3.70	-14.1446	14	3.75	-14.14441
15	3.80	-14.14428	16	3.85	-14.14438	17	3.90	-14.14439
18	3.95	-14.14438	19	4.00	-14.14432	20	4.05	-14.14422
21	4.10	-14.14423	22	4.15	-14.14431	23	4.20	-14.14435
24	4.25	-14.14427	25	4.30	-14.14431	26	4.35	-14.14415

27	4.40	-14.14437	28	5.60	-14.1444	29	5.65	-14.1445
30	5.70	-14.14439	31	5.75	-14.14443	32	5.80	-14.1446
33	5.85	-14.14446	34	5.90	-14.14452	35	5.95	-14.14448
36	6.00	-14.14451	37	6.05	-14.14446	38	6.10	-14.14467
39	6.15	-14.14456	40	6.20	-14.14466	41	6.25	-14.14471
42	6.30	-14.14461	43	6.35	-14.14453	44	6.40	-14.14466
45	6.45	-14.14467	46	6.50	-14.1447	47	6.55	-14.14474
48	6.60	-14.14462	49	7.70	-14.14549	50	7.75	-14.14533
51	7.80	-14.1453	52	7.85	-14.14544	53	7.90	-14.14531
54	7.95	-14.14548	55	8.00	-14.14549			

A.2 Flatness

A.2.1 End Milling

Table A.2.1.1: Plate #10 (For the population, $N = 651$, $l_0 = -0.001607$, $m_0 = 0.001907$, $z_0 = -14.85619$, $h_t = 0.007936$ inch).

y	*n*	*x*	*z*	*x*	*z*	*x*	*z*
5.0	0	5.0	-14.85583	5.1	-14.85683	5.2	-14.85633
	3	5.3	-14.85633	5.4	-14.85633	5.5	-14.85633
	6	5.6	-14.85558	5.7	-14.85558	5.8	-14.85488
	9	5.9	-14.8561	6	-14.8561	6.1	-14.8561
	12	6.2	-14.8561	6.3	-14.8591	6.4	-14.86051
	15	6.5	-14.8615	6.6	-14.86145	6.7	-14.85607
	18	6.8	-14.85688	6.9	-14.85679	7	-14.85694
	21	7.1	-14.85706	7.2	-14.86021	7.3	-14.86046
	24	7.4	-14.86046	7.5	-14.86074	7.6	-14.86074
	27	7.7	-14.86014	7.8	-14.86034	7.9	-14.86123
	30	8	-14.86234				
5.1	31	5	-14.85484	5.1	-14.85771	5.2	-14.85722
	34	5.3	-14.85748	5.4	-14.85761	5.5	-14.8548
	37	5.6	-14.8548	5.7	-14.85767	5.8	-14.85736
	40	5.9	-14.85643	6	-14.85534	6.1	-14.85726
	43	6.2	-14.85726	6.3	-14.85515	6.4	-14.85805
	46	6.5	-14.85818	6.6	-14.85818	6.7	-14.85716
	49	6.8	-14.85827	6.9	-14.85888	7	-14.85931
	52	7.1	-14.85931	7.2	-14.85955	7.3	-14.85955
	55	7.4	-14.85761	7.5	-14.85769	7.6	-14.8586
	58	7.7	-14.86047	7.8	-14.85807	7.9	-14.86176
	61	8	-14.86074				
5.2	62	5	-14.85422	5.1	-14.85584	5.2	-14.85584
	65	5.3	-14.8528	5.4	-14.85405	5.5	-14.85458
	68	5.6	-14.85463	5.7	-14.85463	5.8	-14.85463
	71	5.9	-14.85463	6	-14.85463	6.1	-14.85626

	74	6.2	-14.85309	6.3	-14.85313	6.4	-14.85495
	77	6.5	-14.85436	6.6	-14.85414	6.7	-14.8569
	80	6.8	-14.8569	6.9	-14.85621	7	-14.85494
	83	7.1	-14.85494	7.2	-14.85494	7.3	-14.85494
	86	7.4	-14.85555	7.5	-14.85555	7.6	-14.85864
	89	7.7	-14.85643	7.8	-14.85819	7.9	-14.85819
	92	8	-14.85736				
5.3	93	5	-14.85484	5.1	-14.85257	5.2	-14.85257
	96	5.3	-14.85257	5.4	-14.85282	5.5	-14.85465
	99	5.6	-14.85475	5.7	-14.85441	5.8	-14.85441
	102	5.9	-14.85304	6	-14.85468	6.1	-14.85242
	105	6.2	-14.85267	6.3	-14.85267	6.4	-14.853
	108	6.5	-14.853	6.6	-14.85551	6.7	-14.85551
	111	6.8	-14.85563	6.9	-14.85563	7	-14.85642
	114	7.1	-14.85623	7.2	-14.85623	7.3	-14.85726
	117	7.4	-14.85743	7.5	-14.85743	7.6	-14.85649
	120	7.7	-14.85617	7.8	-14.85821	7.9	-14.85736
	123	8	-14.85869				
5.4	124	5	-14.85527	5.1	-14.85517	5.2	-14.85498
	127	5.3	-14.85498	5.4	-14.85293	5.5	-14.85511
	130	5.6	-14.85298	5.7	-14.85481	5.8	-14.85481
	133	5.9	-14.85297	6	-14.8529	6.1	-14.85522
	136	6.2	-14.85522	6.3	-14.85542	6.4	-14.85542
	139	6.5	-14.8537	6.6	-14.8537	6.7	-14.8537
	142	6.8	-14.85602	6.9	-14.85399	7	-14.85674
	145	7.1	-14.85726	7.2	-14.85726	7.3	-14.855
	148	7.4	-14.855	7.5	-14.8581	7.6	-14.8581
	151	7.7	-14.85887	7.8	-14.85889	7.9	-14.8593
	154	8	-14.85774				
5.5	155	5	-14.85292	5.1	-14.85345	5.2	-14.85342
	158	5.3	-14.85352	5.4	-14.85462	5.5	-14.85315
	161	5.6	-14.85481	5.7	-14.85481	5.8	-14.85481
	164	5.9	-14.85481	6	-14.8556	6.1	-14.8556
	167	6.2	-14.85585	6.3	-14.85585	6.4	-14.85555
	170	6.5	-14.85576	6.6	-14.85576	6.7	-14.85672
	173	6.8	-14.85452	6.9	-14.85449	7	-14.85449
	176	7.1	-14.85761	7.2	-14.8577	7.3	-14.8582
	179	7.4	-14.85546	7.5	-14.85546	7.6	-14.85915
	182	7.7	-14.85915	7.8	-14.8596	7.9	-14.85697
	185	8	-14.85795				
5.6	186	5	-14.85342	5.1	-14.85342	5.2	-14.8551
	189	5.3	-14.8551	5.4	-14.85537	5.5	-14.85332
	192	5.6	-14.85332	5.7	-14.85332	5.8	-14.85332
	195	5.9	-14.85332	6	-14.85383	6.1	-14.85383
	198	6.2	-14.85341	6.3	-14.85341	6.4	-14.85373
	201	6.5	-14.85373	6.6	-14.85373	6.7	-14.85663

	204	6.8	-14.85645	6.9	-14.85645	7	-14.85549
	207	7.1	-14.85549	7.2	-14.85549	7.3	-14.85549
	210	7.4	-14.85797	7.5	-14.85832	7.6	-14.85832
	213	7.7	-14.85882	7.8	-14.85673	7.9	-14.85966
	216	8	-14.85794				
5.7	217	5	-14.85531	5.1	-14.85374	5.2	-14.85291
	220	5.3	-14.85301	5.4	-14.85541	5.5	-14.85541
	223	5.6	-14.85541	5.7	-14.85541	5.8	-14.85284
	226	5.9	-14.85353	6	-14.85572	6.1	-14.85572
	229	6.2	-14.85539	6.3	-14.85568	6.4	-14.85365
	232	6.5	-14.85365	6.6	-14.85365	6.7	-14.85365
	235	6.8	-14.8568	6.9	-14.8568	7	-14.85751
	238	7.1	-14.85719	7.2	-14.85741	7.3	-14.85741
	241	7.4	-14.85775	7.5	-14.85797	7.6	-14.85636
	244	7.7	-14.85683	7.8	-14.8592	7.9	-14.85768
	247	8	-14.85953				
5.8	248	5	-14.85522	5.1	-14.85522	5.2	-14.85522
	251	5.3	-14.85523	5.4	-14.85542	5.5	-14.85301
	254	5.6	-14.85466	5.7	-14.85509	5.8	-14.85318
	257	5.9	-14.85318	6	-14.8531	6.1	-14.8531
	260	6.2	-14.85299	6.3	-14.85299	6.4	-14.85586
	263	6.5	-14.85586	6.6	-14.85382	6.7	-14.85657
	266	6.8	-14.85427	6.9	-14.85444	7	-14.85729
	269	7.1	-14.85524	7.2	-14.85524	7.3	-14.85524
	272	7.4	-14.85632	7.5	-14.85632	7.6	-14.85795
	275	7.7	-14.8589	7.8	-14.8589	7.9	-14.85964
	278	8	-14.85737				
5.9	279	5	-14.85354	5.1	-14.85292	5.2	-14.8533
	282	5.3	-14.85532	5.4	-14.85307	5.5	-14.85539
	285	5.6	-14.85558	5.7	-14.85558	5.8	-14.85572
	288	5.9	-14.85555	6	-14.85555	6.1	-14.85363
	291	6.2	-14.85557	6.3	-14.85587	6.4	-14.85587
	294	6.5	-14.85547	6.6	-14.85547	6.7	-14.8568
	297	6.8	-14.8568	6.9	-14.85677	7	-14.85677
	300	7.1	-14.85677	7.2	-14.85677	7.3	-14.85595
	303	7.4	-14.85595	7.5	-14.85816	7.6	-14.85816
	306	7.7	-14.85816	7.8	-14.85816	7.9	-14.85748
	309	8	-14.85772				
6	310	5	-14.85373	5.1	-14.85373	5.2	-14.85373
	313	5.3	-14.85373	5.4	-14.85599	5.5	-14.854
	316	5.6	-14.85582	5.7	-14.85582	5.8	-14.85563
	319	5.9	-14.85539	6	-14.85646	6.1	-14.85646
	322	6.2	-14.85646	6.3	-14.85614	6.4	-14.85614
	325	6.5	-14.85461	6.6	-14.85461	6.7	-14.85461
	328	6.8	-14.85461	6.9	-14.85765	7	-14.85765
	331	7.1	-14.85626	7.2	-14.85775	7.3	-14.85755

	334	7.4	-14.85851	7.5	-14.85851	7.6	-14.85898
	337	7.7	-14.85898	7.8	-14.85954	7.9	-14.85834
	340	8	-14.86018				
6.1	341	5	-14.85418	5.1	-14.8551	5.2	-14.8554
	344	5.3	-14.8554	5.4	-14.8536	5.5	-14.85487
	347	5.6	-14.85344	5.7	-14.85344	5.8	-14.85344
	350	5.9	-14.85525	6	-14.85551	6.1	-14.85551
	353	6.2	-14.85535	6.3	-14.85385	6.4	-14.85435
	356	6.5	-14.85367	6.6	-14.85367	6.7	-14.85367
	359	6.8	-14.85577	6.9	-14.85577	7	-14.85738
	362	7.1	-14.85591	7.2	-14.85591	7.3	-14.85796
	365	7.4	-14.85796	7.5	-14.85796	7.6	-14.85732
	368	7.7	-14.85965	7.8	-14.85996	7.9	-14.85787
	371	8	-14.85787				
6.2	372	5	-14.85314	5.1	-14.85332	5.2	-14.85505
	375	5.3	-14.85505	5.4	-14.85314	5.5	-14.85314
	378	5.6	-14.8549	5.7	-14.85324	5.8	-14.85512
	381	5.9	-14.85512	6	-14.85558	6.1	-14.85332
	384	6.2	-14.85565	6.3	-14.85565	6.4	-14.85383
	387	6.5	-14.85355	6.6	-14.85641	6.7	-14.85666
	390	6.8	-14.85695	6.9	-14.85695	7	-14.85695
	393	7.1	-14.85695	7.2	-14.85581	7.3	-14.85807
	396	7.4	-14.85807	7.5	-14.85837	7.6	-14.85853
	399	7.7	-14.85903	7.8	-14.85931	7.9	-14.86006
	402	8	-14.85828				
6.3	403	5	-14.853	5.1	-14.85346	5.2	-14.85277
	406	5.3	-14.85277	5.4	-14.85277	5.5	-14.85277
	409	5.6	-14.8542	5.7	-14.85301	5.8	-14.85301
	412	5.9	-14.85301	6	-14.85473	6.1	-14.85485
	415	6.2	-14.85485	6.3	-14.85509	6.4	-14.85509
	418	6.5	-14.85383	6.6	-14.85609	6.7	-14.8544
	421	6.8	-14.8544	6.9	-14.85478	7	-14.85478
	424	7.1	-14.85712	7.2	-14.85712	7.3	-14.85764
	427	7.4	-14.85764	7.5	-14.85732	7.6	-14.85732
	430	7.7	-14.85721	7.8	-14.859	7.9	-14.85772
	433	8	-14.85952				
6.4	434	5	-14.85263	5.1	-14.85236	5.2	-14.85489
	437	5.3	-14.85463	5.4	-14.85463	5.5	-14.85463
	440	5.6	-14.85416	5.7	-14.85416	5.8	-14.85472
	443	5.9	-14.85472	6	-14.85276	6.1	-14.85276
	446	6.2	-14.85276	6.3	-14.85276	6.4	-14.85353
	449	6.5	-14.85583	6.6	-14.85509	6.7	-14.85344
	452	6.8	-14.85604	6.9	-14.85626	7	-14.85618
	455	7.1	-14.85618	7.2	-14.85485	7.3	-14.85485
	458	7.4	-14.85721	7.5	-14.85573	7.6	-14.85835
	461	7.7	-14.85835	7.8	-14.85884	7.9	-14.85747

	464	8	-14.85747				
6.5	465	5	-14.85226	5.1	-14.85435	5.2	-14.85225
	468	5.3	-14.85177	5.4	-14.85429	5.5	-14.85429
	471	5.6	-14.8522	5.7	-14.85209	5.8	-14.85209
	474	5.9	-14.85279	6	-14.85279	6.1	-14.85459
	477	6.2	-14.85459	6.3	-14.85459	6.4	-14.85455
	480	6.5	-14.85455	6.6	-14.85294	6.7	-14.85389
	483	6.8	-14.85389	6.9	-14.85389	7	-14.85426
	486	7.1	-14.85426	7.2	-14.85716	7.3	-14.85716
	489	7.4	-14.85721	7.5	-14.85732	7.6	-14.85732
	492	7.7	-14.85732	7.8	-14.85669	7.9	-14.85669
	495	8	-14.85884				
6.6	496	5	-14.8514	5.1	-14.8514	5.2	-14.85177
	499	5.3	-14.85371	5.4	-14.85371	5.5	-14.85386
	502	5.6	-14.85193	5.7	-14.85193	5.8	-14.85395
	505	5.9	-14.85395	6	-14.85447	6.1	-14.85447
	508	6.2	-14.85199	6.3	-14.85427	6.4	-14.8531
	511	6.5	-14.8531	6.6	-14.85491	6.7	-14.85491
	514	6.8	-14.85491	6.9	-14.85491	7	-14.85545
	517	7.1	-14.85545	7.2	-14.85545	7.3	-14.85464
	520	7.4	-14.85481	7.5	-14.85682	7.6	-14.85657
	523	7.7	-14.85788	7.8	-14.85833	7.9	-14.85586
	526	8	-14.85586				
6.7	527	5	-14.85136	5.1	-14.85126	5.2	-14.85099
	530	5.3	-14.85099	5.4	-14.85359	5.5	-14.85359
	533	5.6	-14.85286	5.7	-14.85286	5.8	-14.85286
	536	5.9	-14.85286	6	-14.85328	6.1	-14.85336
	539	6.2	-14.85391	6.3	-14.85216	6.4	-14.852
	542	6.5	-14.85446	6.6	-14.85446	6.7	-14.85446
	545	6.8	-14.8552	6.9	-14.8552	7	-14.85498
	548	7.1	-14.85498	7.2	-14.85407	7.3	-14.85565
	551	7.4	-14.85565	7.5	-14.8549	7.6	-14.85679
	554	7.7	-14.85679	7.8	-14.85821	7.9	-14.85586
	557	8	-14.85624				
6.8	558	5	-14.8529	5.1	-14.8529	5.2	-14.85291
	561	5.3	-14.85233	5.4	-14.85233	5.5	-14.85124
	564	5.6	-14.85091	5.7	-14.85264	5.8	-14.85264
	567	5.9	-14.85359	6	-14.85359	6.1	-14.85359
	570	6.2	-14.85359	6.3	-14.85388	6.4	-14.85388
	573	6.5	-14.85335	6.6	-14.85245	6.7	-14.85245
	576	6.8	-14.85245	6.9	-14.85471	7	-14.85471
	579	7.1	-14.85501	7.2	-14.8553	7.3	-14.85384
	582	7.4	-14.85608	7.5	-14.85449	7.6	-14.85449
	585	7.7	-14.85661	7.8	-14.85506	7.9	-14.85677
	588	8	-14.8583				
6.9	589	5	-14.85189	5.1	-14.8496	5.2	-14.85211

	592	5.3	-14.85196	5.4	-14.84949	5.5	-14.84949
	595	5.6	-14.84979	5.7	-14.85229	5.8	-14.84934
	598	5.9	-14.84991	6	-14.85032	6.1	-14.85228
	601	6.2	-14.85042	6.3	-14.85042	6.4	-14.85036
	604	6.5	-14.85036	6.6	-14.85173	6.7	-14.85173
	607	6.8	-14.85152	6.9	-14.85152	7	-14.85208
	610	7.1	-14.85234	7.2	-14.85451	7.3	-14.85405
	613	7.4	-14.85253	7.5	-14.85253	7.6	-14.85601
	616	7.7	-14.85296	7.8	-14.85468	7.9	-14.85407
	619	8	-14.85407				
7	620	5	-14.84942	5.1	-14.84965	5.2	-14.84945
	623	5.3	-14.85102	5.4	-14.85144	5.5	-14.85103
	626	5.6	-14.85149	5.7	-14.85167	5.8	-14.8497
	629	5.9	-14.85167	6	-14.85041	6.1	-14.85037
	632	6.2	-14.8515	6.3	-14.85252	6.4	-14.85234
	635	6.5	-14.85025	6.6	-14.85032	6.7	-14.8537
	638	6.8	-14.8537	6.9	-14.85137	7	-14.85307
	641	7.1	-14.85141	7.2	-14.85206	7.3	-14.8541
	644	7.4	-14.85393	7.5	-14.85398	7.6	-14.85235
	647	7.7	-14.85566	7.8	-14.85566	7.9	-14.85593
	650	8	-14.85408				

Table A.2.1.2: Plate #5 (N = 651, l_0 = -0.001266, m_0 = 0.002185, z_0 = -14.86072, h_t = 0.005959).

x	z	x	z	x	z	x	z
$y = 5.0$							
5.0	-14.85722	5.1	-14.85807	5.2	-14.85718	5.3	-14.85781
5.4	-14.85853	5.5	-14.85736	5.6	-14.8585	5.7	-14.85892
5.8	-14.85653	5.9	-14.85653	6	-14.85653	6.1	-14.85734
6.2	-14.85745	6.3	-14.85808	6.4	-14.8585	6.5	-14.85881
6.6	-14.85881	6.7	-14.86032	6.8	-14.86032	6.9	-14.86041
7	-14.85897	7.1	-14.85897	7.2	-14.86038	7.3	-14.8595
7.4	-14.85952	7.5	-14.86206	7.6	-14.86191	7.7	-14.85953
7.8	-14.86252	7.9	-14.86151	8	-14.86218		
$y = 5.1$							
5	-14.85535	5.1	-14.85752	5.2	-14.85621	5.3	-14.85671
5.4	-14.85721	5.5	-14.85541	5.6	-14.85751	5.7	-14.85577
5.8	-14.85667	5.9	-14.85667	6	-14.85564	6.1	-14.85564
6.2	-14.85734	6.3	-14.85761	6.4	-14.85812	6.5	-14.85812
6.6	-14.85828	6.7	-14.85829	6.8	-14.85829	6.9	-14.85835
7	-14.85737	7.1	-14.85737	7.2	-14.85764	7.3	-14.85764
7.4	-14.85939	7.5	-14.86001	7.6	-14.86009	7.7	-14.85854
7.8	-14.86184	7.9	-14.85952	8	-14.86014		
$y = 5.2$							

5	-14.85466	5.1	-14.85567	5.2	-14.85438	5.3	-14.85677
5.4	-14.85581	5.5	-14.85441	5.6	-14.85469	5.7	-14.85668
5.8	-14.85668	5.9	-14.85668	6	-14.85668	6.1	-14.85668
6.2	-14.85665	6.3	-14.85665	6.4	-14.85482	6.5	-14.85549
6.6	-14.85779	6.7	-14.85487	6.8	-14.85487	6.9	-14.85738
7	-14.85613	7.1	-14.85579	7.2	-14.85638	7.3	-14.85638
7.4	-14.85651	7.5	-14.85833	7.6	-14.85858	7.7	-14.8592
7.8	-14.85967	7.9	-14.85907	8	-14.85958		
$y = 5.3$							
5	-14.85447	5.1	-14.85454	5.2	-14.85454	5.3	-14.85454
5.4	-14.85426	5.5	-14.85447	5.6	-14.85591	5.7	-14.85591
5.8	-14.8547	5.9	-14.8547	6	-14.85464	6.1	-14.85464
6.2	-14.85443	6.3	-14.85443	6.4	-14.857	6.5	-14.8571
6.6	-14.85684	6.7	-14.8548	6.8	-14.85739	6.9	-14.85596
7	-14.85736	7.1	-14.85655	7.2	-14.85574	7.3	-14.85801
7.4	-14.8585	7.5	-14.85738	7.6	-14.85738	7.7	-14.85722
7.8	-14.858	7.9	-14.8586	8	-14.85797		
$y = 5.4$							
5	-14.85603	5.1	-14.85468	5.2	-14.85468	5.3	-14.85456
5.4	-14.85387	5.5	-14.85664	5.6	-14.85641	5.7	-14.85438
5.8	-14.85438	5.9	-14.85658	6	-14.85658	6.1	-14.85658
6.2	-14.85479	6.3	-14.85479	6.4	-14.85638	6.5	-14.85584
6.6	-14.85584	6.7	-14.85572	6.8	-14.85572	6.9	-14.85572
7	-14.85655	7.1	-14.85829	7.2	-14.85616	7.3	-14.85628
7.4	-14.85742	7.5	-14.85742	7.6	-14.85775	7.7	-14.85972
7.8	-14.85851	7.9	-14.85851	8	-14.85841		
$y = 5.5$							
5	-14.85468	5.1	-14.8565	5.2	-14.8565	5.3	-14.85635
5.4	-14.85487	5.5	-14.85487	5.6	-14.85475	5.7	-14.85475
5.8	-14.85475	5.9	-14.85475	6	-14.85475	6.1	-14.85475
6.2	-14.85475	6.3	-14.85559	6.4	-14.85559	6.5	-14.85546
6.6	-14.85546	6.7	-14.85725	6.8	-14.85725	6.9	-14.85725
7	-14.85725	7.1	-14.85725	7.2	-14.85868	7.3	-14.85868
7.4	-14.85715	7.5	-14.85958	7.6	-14.85958	7.7	-14.8582
7.8	-14.85853	7.9	-14.85854	8	-14.85854		
$y = 5.6$							
5	-14.85459	5.1	-14.85487	5.2	-14.85478	5.3	-14.85489
5.4	-14.855	5.5	-14.855	5.6	-14.85504	5.7	-14.85504
5.8	-14.85638	5.9	-14.85638	6	-14.85638	6.1	-14.85521
6.2	-14.8556	6.3	-14.85555	6.4	-14.85555	6.5	-14.85762
6.6	-14.85762	6.7	-14.85638	6.8	-14.85638	6.9	-14.85656
7	-14.85656	7.1	-14.85696	7.2	-14.85696	7.3	-14.85705
7.4	-14.85705	7.5	-14.85705	7.6	-14.85705	7.7	-14.85839
7.8	-14.85839	7.9	-14.85966	8	-14.8592		
$y = 5.7$							
5	-14.85478	5.1	-14.85478	5.2	-14.85478	5.3	-14.85605

5.4	-14.85487	5.5	-14.85487	5.6	-14.85475	5.7	-14.85665
5.8	-14.85636	5.9	-14.85636	6	-14.85539	6.1	-14.85539
6.2	-14.85521	6.3	-14.8558	6.4	-14.85701	6.5	-14.85706
6.6	-14.85594	6.7	-14.85594	6.8	-14.85825	6.9	-14.85825
7	-14.85821	7.1	-14.85666	7.2	-14.85752	7.3	-14.85752
7.4	-14.85821	7.5	-14.85778	7.6	-14.85778	7.7	-14.85827
7.8	-14.85827	7.9	-14.86033	8	-14.85819		
$y = 5.8$							
5	-14.85487	5.1	-14.856	5.2	-14.85494	5.3	-14.8548
5.4	-14.85664	5.5	-14.85664	5.6	-14.85452	5.7	-14.85636
5.8	-14.85474	5.9	-14.85474	6	-14.85474	6.1	-14.85474
6.2	-14.8558	6.3	-14.85558	6.4	-14.85562	6.5	-14.85562
6.6	-14.85577	6.7	-14.85598	6.8	-14.85598	6.9	-14.85766
7	-14.85649	7.1	-14.85649	7.2	-14.85841	7.3	-14.85841
7.4	-14.85712	7.5	-14.85876	7.6	-14.85876	7.7	-14.85876
7.8	-14.85876	7.9	-14.85847	8	-14.85847		
$y = 5.9$							
5	-14.85499	5.1	-14.85487	5.2	-14.85487	5.3	-14.8562
5.4	-14.85706	5.5	-14.85706	5.6	-14.85706	5.7	-14.85525
5.8	-14.85525	5.9	-14.85488	6	-14.85669	6.1	-14.85669
6.2	-14.85554	6.3	-14.85751	6.4	-14.85751	6.5	-14.85513
6.6	-14.85513	6.7	-14.85794	6.8	-14.85613	6.9	-14.8561
7	-14.85714	7.1	-14.85714	7.2	-14.85641	7.3	-14.85641
7.4	-14.8575	7.5	-14.85703	7.6	-14.85809	7.7	-14.85786
7.8	-14.85819	7.9	-14.85819	8	-14.85809		
$y = 6$							
5	-14.85472	5.1	-14.85474	5.2	-14.85474	5.3	-14.85467
5.4	-14.85526	5.5	-14.85567	5.6	-14.85559	5.7	-14.85686
5.8	-14.85686	5.9	-14.85694	6	-14.85694	6.1	-14.85554
6.2	-14.85554	6.3	-14.85578	6.4	-14.85578	6.5	-14.85731
6.6	-14.85731	6.7	-14.85581	6.8	-14.85606	6.9	-14.85804
7	-14.85804	7.1	-14.85679	7.2	-14.85846	7.3	-14.85846
7.4	-14.85712	7.5	-14.85753	7.6	-14.85931	7.7	-14.86022
7.8	-14.86022	7.9	-14.85815	8	-14.85913		
$y = 6.1$							
5	-14.85494	5.1	-14.85487	5.2	-14.85573	5.3	-14.85573
5.4	-14.85567	5.5	-14.85567	5.6	-14.85591	5.7	-14.85591
5.8	-14.85551	5.9	-14.85551	6	-14.85441	6.1	-14.85441
6.2	-14.85441	6.3	-14.85441	6.4	-14.85441	6.5	-14.85441
6.6	-14.85581	6.7	-14.85684	6.8	-14.85684	6.9	-14.85778
7	-14.85778	7.1	-14.85684	7.2	-14.85684	7.3	-14.85801
7.4	-14.85801	7.5	-14.85785	7.6	-14.85785	7.7	-14.85887
7.8	-14.85887	7.9	-14.86024	8	-14.85851		
$y = 6.2$							
5	-14.85387	5.1	-14.85519	5.2	-14.85437	5.3	-14.85437
5.4	-14.8547	5.5	-14.8547	5.6	-14.8547	5.7	-14.85597

5.8	-14.85597	5.9	-14.85441	6	-14.85464	6.1	-14.85536
6.2	-14.855	6.3	-14.855	6.4	-14.85645	6.5	-14.85645
6.6	-14.85545	6.7	-14.85545	6.8	-14.85545	6.9	-14.85709
7	-14.85709	7.1	-14.85731	7.2	-14.85783	7.3	-14.85632
7.4	-14.85632	7.5	-14.85854	7.6	-14.85854	7.7	-14.85723
7.8	-14.85767	7.9	-14.85829	8	-14.85913		
$y = 6.3$							
5	-14.8535	5.1	-14.85498	5.2	-14.85394	5.3	-14.85506
5.4	-14.85533	5.5	-14.85555	5.6	-14.85456	5.7	-14.85456
5.8	-14.85578	5.9	-14.85578	6	-14.85404	6.1	-14.85632
6.2	-14.85632	6.3	-14.85632	6.4	-14.85668	6.5	-14.85668
6.6	-14.85488	6.7	-14.85712	6.8	-14.8555	6.9	-14.85595
7	-14.8561	7.1	-14.85783	7.2	-14.85553	7.3	-14.85779
7.4	-14.85663	7.5	-14.85808	7.6	-14.85808	7.7	-14.85808
7.8	-14.85808	7.9	-14.85841	8	-14.85755		
$y = 6.4$							
5	-14.85379	5.1	-14.85387	5.2	-14.85391	5.3	-14.85391
5.4	-14.85407	5.5	-14.85407	5.6	-14.85407	5.7	-14.85407
5.8	-14.85407	5.9	-14.85404	6	-14.85433	6.1	-14.85445
6.2	-14.85627	6.3	-14.85627	6.4	-14.8545	6.5	-14.8545
6.6	-14.8545	6.7	-14.85505	6.8	-14.85505	6.9	-14.85505
7	-14.85566	7.1	-14.85566	7.2	-14.85566	7.3	-14.85566
7.4	-14.85566	7.5	-14.85667	7.6	-14.85667	7.7	-14.85709
7.8	-14.85709	7.9	-14.85885	8	-14.85725		
$y = 6.5$							
5	-14.85459	5.1	-14.85379	5.2	-14.85379	5.3	-14.85364
5.4	-14.85356	5.5	-14.85356	5.6	-14.8536	5.7	-14.85489
5.8	-14.85338	5.9	-14.85338	6	-14.85442	6.1	-14.85542
6.2	-14.85419	6.3	-14.8547	6.4	-14.85616	6.5	-14.85451
6.6	-14.85451	6.7	-14.8548	6.8	-14.8548	6.9	-14.85658
7	-14.85658	7.1	-14.85662	7.2	-14.85506	7.3	-14.85576
7.4	-14.85563	7.5	-14.85629	7.6	-14.85648	7.7	-14.85648
7.8	-14.85746	7.9	-14.8575	8	-14.85875		
$y = 6.6$							
5	-14.85461	5.1	-14.85459	5.2	-14.85252	5.3	-14.85341
5.4	-14.85256	5.5	-14.8549	5.6	-14.85482	5.7	-14.8535
5.8	-14.8535	5.9	-14.85391	6	-14.85391	6.1	-14.85359
6.2	-14.85502	6.3	-14.85373	6.4	-14.85373	6.5	-14.85487
6.6	-14.85487	6.7	-14.85592	6.8	-14.85592	6.9	-14.85592
7	-14.85592	7.1	-14.85538	7.2	-14.85508	7.3	-14.85557
7.4	-14.85659	7.5	-14.85611	7.6	-14.85559	7.7	-14.85617
7.8	-14.85617	7.9	-14.85624	8	-14.85648		
$y = 6.7$							
5	-14.85349	5.1	-14.85301	5.2	-14.85344	5.3	-14.85373
5.4	-14.8535	5.5	-14.8535	5.6	-14.8535	5.7	-14.85311
5.8	-14.85311	5.9	-14.85448	6	-14.85458	6.1	-14.85409

6.2	-14.85409	6.3	-14.85482	6.4	-14.85415	6.5	-14.85415
6.6	-14.85415	6.7	-14.85453	6.8	-14.85453	6.9	-14.85412
7	-14.85538	7.1	-14.85436	7.2	-14.85671	7.3	-14.85671
7.4	-14.85435	7.5	-14.85527	7.6	-14.85541	7.7	-14.85692
7.8	-14.85457	7.9	-14.85552	8	-14.85622		
$y = 6.8$							
5	-14.85156	5.1	-14.85303	5.2	-14.85171	5.3	-14.85297
5.4	-14.84995	5.5	-14.85201	5.6	-14.85287	5.7	-14.85223
5.8	-14.85186	5.9	-14.85186	6	-14.85261	6.1	-14.85217
6.2	-14.85308	6.3	-14.85361	6.4	-14.85229	6.5	-14.85229
6.6	-14.85229	6.7	-14.85353	6.8	-14.85353	6.9	-14.85441
7	-14.85462	7.1	-14.85418	7.2	-14.85497	7.3	-14.85518
7.4	-14.85415	7.5	-14.85381	7.6	-14.85381	7.7	-14.85526
7.8	-14.85526	7.9	-14.85538	8	-14.8561		
$y = 6.9$							
5	-14.8518	5.1	-14.85355	5.2	-14.85153	5.3	-14.85145
5.4	-14.85139	5.5	-14.85153	5.6	-14.85153	5.7	-14.85155
5.8	-14.85155	5.9	-14.85106	6	-14.85267	6.1	-14.85143
6.2	-14.85321	6.3	-14.85238	6.4	-14.85183	6.5	-14.85405
6.6	-14.85405	6.7	-14.85296	6.8	-14.85296	6.9	-14.85296
7	-14.85296	7.1	-14.85497	7.2	-14.85327	7.3	-14.85505
7.4	-14.85505	7.5	-14.85427	7.6	-14.85427	7.7	-14.85427
7.8	-14.85454	7.9	-14.85454	8	-14.85436		
$y = 7$							
5	-14.85122	5.1	-14.8516	5.2	-14.85132	5.3	-14.85132
5.4	-14.85143	5.5	-14.85143	5.6	-14.85157	5.7	-14.85157
5.8	-14.85157	5.9	-14.85267	6	-14.85143	6.1	-14.85205
6.2	-14.85195	6.3	-14.85195	6.4	-14.85195	6.5	-14.85195
6.6	-14.85195	6.7	-14.8526	6.8	-14.8526	6.9	-14.8526
7	-14.8526	7.1	-14.85327	7.2	-14.85327	7.3	-14.85327
7.4	-14.85371	7.5	-14.85371	7.6	-14.85371	7.7	-14.85383
7.8	-14.85411	7.9	-14.85411	8	-14.85411		

Table A.2.1.3: Plate #7 (N = 651, l_0 = -0.001212, m_0 = 0.000789, z_0 = -14.85619, h_t = 0.006047).

x	z	x	z	x	z	x	z
$y = 5$							
5	-14.86073	5.1	-14.86078	5.2	-14.86078	5.3	-14.85988
5.4	-14.86013	5.5	-14.8591	5.6	-14.8591	5.7	-14.8591
5.8	-14.8591	5.9	-14.8591	6	-14.8591	6.1	-14.8591
6.2	-14.8591	6.3	-14.8591	6.4	-14.86075	6.5	-14.86075
6.6	-14.86116	6.7	-14.86116	6.8	-14.86116	6.9	-14.86156
7	-14.86156	7.1	-14.86156	7.2	-14.86156	7.3	-14.86232
7.4	-14.86232	7.5	-14.86232	7.6	-14.86232	7.7	-14.86232
7.8	-14.86356	7.9	-14.86325	8	-14.86396		

$y = 5.1$							
5	-14.85968	5.1	-14.86051	5.2	-14.85997	5.3	-14.85857
5.4	-14.8596	5.5	-14.86011	5.6	-14.86011	5.7	-14.85995
5.8	-14.85956	5.9	-14.85881	6	-14.85945	6.1	-14.86005
6.2	-14.86005	6.3	-14.85974	6.4	-14.85947	6.5	-14.85947
6.6	-14.86026	6.7	-14.85996	6.8	-14.86078	6.9	-14.8607
7	-14.8607	7.1	-14.85983	7.2	-14.86137	7.3	-14.86072
7.4	-14.86116	7.5	-14.86263	7.6	-14.86203	7.7	-14.86278
7.8	-14.86354	7.9	-14.86354	8	-14.86345		
$y = 5.2$							
5	-14.85777	5.1	-14.85777	5.2	-14.85771	5.3	-14.85796
5.4	-14.8576	5.5	-14.8575	5.6	-14.85822	5.7	-14.85822
5.8	-14.85822	5.9	-14.85822	6	-14.85822	6.1	-14.85765
6.2	-14.85765	6.3	-14.85765	6.4	-14.85765	6.5	-14.85765
6.6	-14.8586	6.7	-14.8586	6.8	-14.8586	6.9	-14.85855
7	-14.85855	7.1	-14.85925	7.2	-14.85925	7.3	-14.85935
7.4	-14.86058	7.5	-14.86051	7.6	-14.86051	7.7	-14.86044
7.8	-14.86042	7.9	-14.86117	8	-14.86117		
$y = 5.3$							
5	-14.85828	5.1	-14.85748	5.2	-14.85748	5.3	-14.8583
5.4	-14.85774	5.5	-14.85774	5.6	-14.85749	5.7	-14.8582
5.8	-14.8582	5.9	-14.85751	6	-14.85751	6.1	-14.85775
6.2	-14.85775	6.3	-14.85817	6.4	-14.85817	6.5	-14.85767
6.6	-14.85856	6.7	-14.85856	6.8	-14.85801	6.9	-14.859
7	-14.859	7.1	-14.85968	7.2	-14.85968	7.3	-14.85929
7.4	-14.85953	7.5	-14.85953	7.6	-14.86009	7.7	-14.86009
7.8	-14.86023	7.9	-14.86142	8	-14.86067		
$y = 5.4$							
5	-14.85792	5.1	-14.8581	5.2	-14.8583	5.3	-14.85854
5.4	-14.85809	5.5	-14.85809	5.6	-14.85814	5.7	-14.85784
5.8	-14.85826	5.9	-14.85826	6	-14.85835	6.1	-14.85835
6.2	-14.8583	6.3	-14.8583	6.4	-14.85886	6.5	-14.85881
6.6	-14.85881	6.7	-14.85872	6.8	-14.85939	6.9	-14.85939
7	-14.85918	7.1	-14.85925	7.2	-14.85957	7.3	-14.85957
7.4	-14.86045	7.5	-14.86045	7.6	-14.86045	7.7	-14.86105
7.8	-14.86105	7.9	-14.86148	8	-14.86142		
$y = 5.5$							
5	-14.85824	5.1	-14.85824	5.2	-14.8582	5.3	-14.8582
5.4	-14.85817	5.5	-14.85817	5.6	-14.85817	5.7	-14.85835
5.8	-14.85872	5.9	-14.85872	6	-14.85872	6.1	-14.85796
6.2	-14.85796	6.3	-14.85883	6.4	-14.85883	6.5	-14.85868
6.6	-14.85868	6.7	-14.85868	6.8	-14.85934	6.9	-14.85934
7	-14.85901	7.1	-14.85983	7.2	-14.86024	7.3	-14.86071
7.4	-14.86071	7.5	-14.86063	7.6	-14.86063	7.7	-14.86063
7.8	-14.86063	7.9	-14.86063	8	-14.86063		
$y = 5.6$							

5	-14.85853	5.1	-14.85888	5.2	-14.85873	5.3	-14.85803
5.4	-14.85868	5.5	-14.85821	5.6	-14.85821	5.7	-14.85827
5.8	-14.858	5.9	-14.858	6	-14.86178	6.1	-14.85869
6.2	-14.85842	6.3	-14.85842	6.4	-14.85842	6.5	-14.85842
6.6	-14.8595	6.7	-14.8595	6.8	-14.85931	6.9	-14.85931
7	-14.86001	7.1	-14.86001	7.2	-14.86038	7.3	-14.86038
7.4	-14.85998	7.5	-14.86072	7.6	-14.86073	7.7	-14.86073
7.8	-14.86127	7.9	-14.86148	8	-14.86148		
$y = 5.7$							
5	-14.85837	5.1	-14.85823	5.2	-14.85811	5.3	-14.85811
5.4	-14.85818	5.5	-14.85818	5.6	-14.85814	5.7	-14.85812
5.8	-14.8584	5.9	-14.85885	6	-14.85885	6.1	-14.85864
6.2	-14.85864	6.3	-14.85864	6.4	-14.85912	6.5	-14.85912
6.6	-14.85877	6.7	-14.85938	6.8	-14.85938	6.9	-14.85961
7	-14.85961	7.1	-14.85961	7.2	-14.85984	7.3	-14.85984
7.4	-14.8606	7.5	-14.8606	7.6	-14.8613	7.7	-14.8613
7.8	-14.86128	7.9	-14.86151	8	-14.86168		
$y = 5.8$							
5	-14.85801	5.1	-14.85801	5.2	-14.85801	5.3	-14.85801
5.4	-14.85801	5.5	-14.85766	5.6	-14.85766	5.7	-14.85804
5.8	-14.85853	5.9	-14.85814	6	-14.85814	6.1	-14.85897
6.2	-14.8584	6.3	-14.85872	6.4	-14.85872	6.5	-14.85926
6.6	-14.85926	6.7	-14.85859	6.8	-14.85859	6.9	-14.85891
7	-14.85944	7.1	-14.85975	7.2	-14.86037	7.3	-14.8607
7.4	-14.8607	7.5	-14.86104	7.6	-14.8612	7.7	-14.86194
7.8	-14.86194	7.9	-14.86151	8	-14.86151		
$y = 5.9$							
5	-14.85868	5.1	-14.85868	5.2	-14.85772	5.3	-14.85772
5.4	-14.8585	5.5	-14.859	5.6	-14.859	5.7	-14.85909
5.8	-14.85909	5.9	-14.85862	6	-14.85862	6.1	-14.85887
6.2	-14.85887	6.3	-14.85934	6.4	-14.85934	6.5	-14.85934
6.6	-14.85974	6.7	-14.85974	6.8	-14.85955	6.9	-14.85972
7	-14.85972	7.1	-14.86063	7.2	-14.86063	7.3	-14.86032
7.4	-14.86032	7.5	-14.86032	7.6	-14.86136	7.7	-14.86136
7.8	-14.86193	7.9	-14.86193	8	-14.86193		
$y = 6$							
5	-14.85858	5.1	-14.85858	5.2	-14.85923	5.3	-14.85923
5.4	-14.85844	5.5	-14.85844	5.6	-14.85844	5.7	-14.85891
5.8	-14.85995	5.9	-14.85951	6	-14.85951	6.1	-14.8599
6.2	-14.8599	6.3	-14.8599	6.4	-14.8599	6.5	-14.8599
6.6	-14.8599	6.7	-14.8599	6.8	-14.86105	6.9	-14.86105
7	-14.86015	7.1	-14.86029	7.2	-14.86029	7.3	-14.86099
7.4	-14.86099	7.5	-14.86138	7.6	-14.86138	7.7	-14.86174
7.8	-14.86174	7.9	-14.86174	8	-14.86174		
$y = 6.1$							
5	-14.85942	5.1	-14.85927	5.2	-14.85951	5.3	-14.85951

5.4	-14.85949	5.5	-14.85949	5.6	-14.8592	5.7	-14.8592
5.8	-14.85907	5.9	-14.85907	6	-14.85956	6.1	-14.85983
6.2	-14.85996	6.3	-14.85996	6.4	-14.8601	6.5	-14.85993
6.6	-14.86044	6.7	-14.86044	6.8	-14.86068	6.9	-14.86068
7	-14.86029	7.1	-14.86069	7.2	-14.86043	7.3	-14.8611
7.4	-14.8611	7.5	-14.86217	7.6	-14.86194	7.7	-14.86193
7.8	-14.86276	7.9	-14.86292	8	-14.8627		
$y = 6.2$							
5	-14.85925	5.1	-14.85925	5.2	-14.85925	5.3	-14.85925
5.4	-14.85851	5.5	-14.85858	5.6	-14.85858	5.7	-14.8588
5.8	-14.8588	5.9	-14.8594	6	-14.85928	6.1	-14.85928
6.2	-14.85928	6.3	-14.85996	6.4	-14.85996	6.5	-14.85996
6.6	-14.85996	6.7	-14.86033	6.8	-14.86033	6.9	-14.86054
7	-14.86056	7.1	-14.86124	7.2	-14.86124	7.3	-14.86108
7.4	-14.86105	7.5	-14.86234	7.6	-14.86234	7.7	-14.86181
7.8	-14.86181	7.9	-14.86181	8	-14.86181		
$y = 6.3$							
5	-14.8587	5.1	-14.85909	5.2	-14.85909	5.3	-14.85849
5.4	-14.85871	5.5	-14.85871	5.6	-14.859	5.7	-14.859
5.8	-14.859	5.9	-14.859	6	-14.859	6.1	-14.85944
6.2	-14.85917	6.3	-14.85917	6.4	-14.85917	6.5	-14.85917
6.6	-14.85934	6.7	-14.85934	6.8	-14.85934	6.9	-14.86043
7	-14.86043	7.1	-14.86043	7.2	-14.8607	7.3	-14.8607
7.4	-14.8614	7.5	-14.86121	7.6	-14.862	7.7	-14.86219
7.8	-14.8616	7.9	-14.8616	8	-14.8616		
$y = 6.4$							
5	-14.85901	5.1	-14.85929	5.2	-14.85929	5.3	-14.85929
5.4	-14.85812	5.5	-14.85856	5.6	-14.85856	5.7	-14.85877
5.8	-14.85877	5.9	-14.85894	6	-14.85835	6.1	-14.85959
6.2	-14.85959	6.3	-14.8597	6.4	-14.8597	6.5	-14.8597
6.6	-14.85982	6.7	-14.85982	6.8	-14.8597	6.9	-14.8597
7	-14.86014	7.1	-14.86014	7.2	-14.86139	7.3	-14.86139
7.4	-14.86102	7.5	-14.86109	7.6	-14.86193	7.7	-14.8613
7.8	-14.86213	7.9	-14.86213	8	-14.86213		
$y = 6.5$							
5	-14.85784	5.1	-14.85792	5.2	-14.85792	5.3	-14.85869
5.4	-14.85823	5.5	-14.85889	5.6	-14.85777	5.7	-14.85834
5.8	-14.85834	5.9	-14.85809	6	-14.85809	6.1	-14.8592
6.2	-14.85906	6.3	-14.85936	6.4	-14.85885	6.5	-14.85939
6.6	-14.85939	6.7	-14.85939	6.8	-14.85939	6.9	-14.86006
7	-14.86006	7.1	-14.86092	7.2	-14.86092	7.3	-14.86098
7.4	-14.86098	7.5	-14.86149	7.6	-14.86183	7.7	-14.86183
7.8	-14.86206	7.9	-14.86153	8	-14.86153		
$y = 6.6$							
5	-14.85791	5.1	-14.85791	5.2	-14.85791	5.3	-14.85809
5.4	-14.85809	5.5	-14.85775	5.6	-14.85775	5.7	-14.85775

5.8	-14.85775	5.9	-14.85775	6	-14.85775	6.1	-14.85848
6.2	-14.85899	6.3	-14.85854	6.4	-14.85854	6.5	-14.85935
6.6	-14.85935	6.7	-14.85935	6.8	-14.85935	6.9	-14.85997
7	-14.85997	7.1	-14.85996	7.2	-14.86055	7.3	-14.86055
7.4	-14.86026	7.5	-14.86026	7.6	-14.86124	7.7	-14.86138
7.8	-14.86138	7.9	-14.86202	8	-14.86202		
$y = 6.7$							
5	-14.85725	5.1	-14.85725	5.2	-14.85729	5.3	-14.85769
5.4	-14.85717	5.5	-14.85729	5.6	-14.85807	5.7	-14.85807
5.8	-14.8574	5.9	-14.8574	6	-14.8574	6.1	-14.85827
6.2	-14.85827	6.3	-14.85827	6.4	-14.85827	6.5	-14.85876
6.6	-14.85876	6.7	-14.85858	6.8	-14.85858	6.9	-14.85946
7	-14.85959	7.1	-14.85959	7.2	-14.85964	7.3	-14.86004
7.4	-14.86004	7.5	-14.86035	7.6	-14.86035	7.7	-14.86069
7.8	-14.85982	7.9	-14.86091	8	-14.86091		
$y = 6.8$							
5	-14.85693	5.1	-14.85693	5.2	-14.85693	5.3	-14.85693
5.4	-14.85654	5.5	-14.85654	5.6	-14.85673	5.7	-14.85643
5.8	-14.85643	5.9	-14.85716	6	-14.85716	6.1	-14.85716
6.2	-14.85758	6.3	-14.85758	6.4	-14.85758	6.5	-14.85758
6.6	-14.85758	6.7	-14.85758	6.8	-14.8582	6.9	-14.85847
7	-14.85861	7.1	-14.85861	7.2	-14.85888	7.3	-14.859
7.4	-14.85957	7.5	-14.85906	7.6	-14.85968	7.7	-14.86018
7.8	-14.86018	7.9	-14.85996	8	-14.85996		
$y = 6.9$							
5	-14.85631	5.1	-14.85586	5.2	-14.8563	5.3	-14.85589
5.4	-14.85589	5.5	-14.85638	5.6	-14.85638	5.7	-14.85611
5.8	-14.85619	5.9	-14.85627	6	-14.85623	6.1	-14.857
6.2	-14.85659	6.3	-14.85659	6.4	-14.85709	6.5	-14.85709
6.6	-14.85728	6.7	-14.85688	6.8	-14.85751	6.9	-14.85751
7	-14.85761	7.1	-14.85794	7.2	-14.85853	7.3	-14.85855
7.4	-14.85875	7.5	-14.85934	7.6	-14.85811	7.7	-14.85957
7.8	-14.85957	7.9	-14.86006	8	-14.86006		
$y = 7$							
5	-14.85638	5.1	-14.85494	5.2	-14.85577	5.3	-14.85577
5.4	-14.8563	5.5	-14.8563	5.6	-14.8563	5.7	-14.85652
5.8	-14.85652	5.9	-14.85641	6	-14.85641	6.1	-14.85641
6.2	-14.85641	6.3	-14.85715	6.4	-14.85715	6.5	-14.85715
6.6	-14.85744	6.7	-14.85759	6.8	-14.85774	6.9	-14.85742
7	-14.85794	7.1	-14.85794	7.2	-14.85816	7.3	-14.85821
7.4	-14.85818	7.5	-14.85923	7.6	-14.85923	7.7	-14.85923
7.8	-14.85681	7.9	-14.85993	8	-14.85919		

Table A.2.1.4: Plate #4 (N = 651, l_0 = -0.001013, m_0 = 0.001365, z_0 = -14.8597, h_t = 0.008214).

x	z	x	z	x	z	x	z
$y = 5$							
5	-14.85969	5.1	-14.86062	5.2	-14.85885	5.3	-14.85926
5.4	-14.86033	5.5	-14.86036	5.6	-14.85898	5.7	-14.85898
5.8	-14.85922	5.9	-14.85922	6	-14.86065	6.1	-14.85941
6.2	-14.85941	6.3	-14.85954	6.4	-14.85954	6.5	-14.85954
6.6	-14.85954	6.7	-14.86056	6.8	-14.86056	6.9	-14.86056
7	-14.86056	7.1	-14.86078	7.2	-14.86078	7.3	-14.86122
7.4	-14.86283	7.5	-14.86118	7.6	-14.86207	7.7	-14.86207
7.8	-14.86469	7.9	-14.86263	8	-14.86323		
$y = 5.1$							
5	-14.86048	5.1	-14.85857	5.2	-14.85808	5.3	-14.85808
5.4	-14.85857	5.5	-14.85842	5.6	-14.85893	5.7	-14.86054
5.8	-14.86054	5.9	-14.85904	6	-14.85888	6.1	-14.85888
6.2	-14.85935	6.3	-14.85935	6.4	-14.85917	6.5	-14.85917
6.6	-14.858	6.7	-14.85874	6.8	-14.86148	6.9	-14.86148
7	-14.86009	7.1	-14.86009	7.2	-14.85887	7.3	-14.86124
7.4	-14.86063	7.5	-14.85984	7.6	-14.86096	7.7	-14.86102
7.8	-14.86102	7.9	-14.86249	8	-14.86387		
$y = 5.2$							
5	-14.85809	5.1	-14.85658	5.2	-14.85641	5.3	-14.85597
5.4	-14.85788	5.5	-14.85774	5.6	-14.85588	5.7	-14.85605
5.8	-14.85622	5.9	-14.85612	6	-14.85612	6.1	-14.85615
6.2	-14.85634	6.3	-14.85634	6.4	-14.85694	6.5	-14.85694
6.6	-14.85768	6.7	-14.85634	6.8	-14.85634	6.9	-14.85874
7	-14.85874	7.1	-14.85731	7.2	-14.85752	7.3	-14.85825
7.4	-14.85925	7.5	-14.85925	7.6	-14.85905	7.7	-14.85905
7.8	-14.85905	7.9	-14.85975	8	-14.85975		
$y = 5.3$							
5	-14.85652	5.1	-14.85659	5.2	-14.8579	5.3	-14.85604
5.4	-14.85642	5.5	-14.85642	5.6	-14.85604	5.7	-14.85636
5.8	-14.85669	5.9	-14.85669	6	-14.85804	6.1	-14.85697
6.2	-14.85697	6.3	-14.85641	6.4	-14.85641	6.5	-14.85842
6.6	-14.85741	6.7	-14.85706	6.8	-14.85718	6.9	-14.85891
7	-14.85785	7.1	-14.85785	7.2	-14.86033	7.3	-14.86033
7.4	-14.86033	7.5	-14.86033	7.6	-14.86033	7.7	-14.85902
7.8	-14.85895	7.9	-14.85905	8	-14.8616		
$y = 5.4$							
5	-14.85707	5.1	-14.8589	5.2	-14.8589	5.3	-14.85693
5.4	-14.85651	5.5	-14.85651	5.6	-14.85853	5.7	-14.85667
5.8	-14.85636	5.9	-14.85856	6	-14.85685	6.1	-14.85685
6.2	-14.85685	6.3	-14.85685	6.4	-14.85742	6.5	-14.85742
6.6	-14.8569	6.7	-14.85673	6.8	-14.85673	6.9	-14.85793

7	-14.85793	7.1	-14.85816	7.2	-14.85816	7.3	-14.85891
7.4	-14.85891	7.5	-14.85922	7.6	-14.85922	7.7	-14.86134
7.8	-14.86182	7.9	-14.86003	8	-14.86031		
$y = 5.5$							
5	-14.85707	5.1	-14.85769	5.2	-14.85742	5.3	-14.85669
5.4	-14.85658	5.5	-14.85658	5.6	-14.85689	5.7	-14.85689
5.8	-14.857	5.9	-14.85706	6	-14.85709	6.1	-14.85709
6.2	-14.85726	6.3	-14.85751	6.4	-14.85751	6.5	-14.85928
6.6	-14.85928	6.7	-14.85975	6.8	-14.85975	6.9	-14.85997
7	-14.85997	7.1	-14.85997	7.2	-14.85871	7.3	-14.85871
7.4	-14.85938	7.5	-14.85938	7.6	-14.85938	7.7	-14.85965
7.8	-14.8606	7.9	-14.86164	8	-14.86084		
$y = 5.6$							
5	-14.85941	5.1	-14.85941	5.2	-14.8571	5.3	-14.85738
5.4	-14.85747	5.5	-14.85748	5.6	-14.85748	5.7	-14.85926
5.8	-14.85926	5.9	-14.85693	6	-14.85693	6.1	-14.85765
6.2	-14.85765	6.3	-14.85717	6.4	-14.85764	6.5	-14.85818
6.6	-14.85792	6.7	-14.85782	6.8	-14.85782	6.9	-14.8584
7	-14.8584	7.1	-14.85998	7.2	-14.85841	7.3	-14.85841
7.4	-14.85865	7.5	-14.86081	7.6	-14.86081	7.7	-14.86081
7.8	-14.8601	7.9	-14.8601	8	-14.8606		
$y = 5.7$							
5	-14.85734	5.1	-14.8571	5.2	-14.85717	5.3	-14.85726
5.4	-14.8571	5.5	-14.85723	5.6	-14.85894	5.7	-14.85707
5.8	-14.85707	5.9	-14.85906	6	-14.85906	6.1	-14.85718
6.2	-14.85718	6.3	-14.85768	6.4	-14.85937	6.5	-14.85937
6.6	-14.85818	6.7	-14.85818	6.8	-14.85842	6.9	-14.85842
7	-14.85842	7.1	-14.85842	7.2	-14.85883	7.3	-14.85865
7.4	-14.86087	7.5	-14.86087	7.6	-14.86195	7.7	-14.86195
7.8	-14.856056	7.9	-14.86195	8	-14.856056		
$y = 5.8$							
5	-14.85726	5.1	-14.85742	5.2	-14.85758	5.3	-14.85709
5.4	-14.85877	5.5	-14.85877	5.6	-14.85742	5.7	-14.85855
5.8	-14.85724	5.9	-14.85763	6	-14.85763	6.1	-14.85907
6.2	-14.85756	6.3	-14.85811	6.4	-14.85811	6.5	-14.85756
6.6	-14.85756	6.7	-14.85756	6.8	-14.85842	6.9	-14.85842
7	-14.85842	7.1	-14.85849	7.2	-14.86044	7.3	-14.85897
7.4	-14.85918	7.5	-14.85918	7.6	-14.86052	7.7	-14.86052
7.8	-14.86052	7.9	-14.86016	8	-14.86016		
$y = 5.9$							
5	-14.85752	5.1	-14.85881	5.2	-14.85761	5.3	-14.85779
5.4	-14.85779	5.5	-14.85931	5.6	-14.85931	5.7	-14.85931
5.8	-14.85743	5.9	-14.85739	6	-14.85739	6.1	-14.8576
6.2	-14.85798	6.3	-14.8581	6.4	-14.8581	6.5	-14.8581
6.6	-14.85843	6.7	-14.85843	6.8	-14.85843	6.9	-14.85841
7	-14.85841	7.1	-14.8606	7.2	-14.8606	7.3	-14.85993

7.4	-14.85993	7.5	-14.85993	7.6	-14.85993	7.7	-14.85993
7.8	-14.85993	7.9	-14.85993	8	-14.85993		
$y = 6$							
5	-14.85777	5.1	-14.85777	5.2	-14.85772	5.3	-14.85772
5.4	-14.85772	5.5	-14.85772	5.6	-14.85772	5.7	-14.85772
5.8	-14.85772	5.9	-14.85777	6	-14.85777	6.1	-14.85717
6.2	-14.85767	6.3	-14.85998	6.4	-14.85998	6.5	-14.85998
6.6	-14.85903	6.7	-14.85903	6.8	-14.85903	6.9	-14.85903
7	-14.8589	7.1	-14.8589	7.2	-14.85915	7.3	-14.85909
7.4	-14.85909	7.5	-14.86133	7.6	-14.85989	7.7	-14.86176
7.8	-14.86095	7.9	-14.8607	8	-14.86272		
$y = 6.1$							
5	-14.858	5.1	-14.85979	5.2	-14.85911	5.3	-14.85928
5.4	-14.85928	5.5	-14.85774	5.6	-14.85774	5.7	-14.85774
5.8	-14.85774	5.9	-14.86003	6	-14.86003	6.1	-14.85792
6.2	-14.85792	6.3	-14.85786	6.4	-14.85811	6.5	-14.85811
6.6	-14.85992	6.7	-14.85979	6.8	-14.85852	6.9	-14.85889
7	-14.85924	7.1	-14.85924	7.2	-14.86116	7.3	-14.86053
7.4	-14.856035	7.5	-14.856035	7.6	-14.85986	7.7	-14.86051
7.8	-14.86286	7.9	-14.86286	8	-14.86286		
$y = 6.2$							
5	-14.85933	5.1	-14.85698	5.2	-14.85776	5.3	-14.85776
5.4	-14.85729	5.5	-14.85729	5.6	-14.857	5.7	-14.85729
5.8	-14.85678	5.9	-14.85678	6	-14.85678	6.1	-14.85716
6.2	-14.85722	6.3	-14.85764	6.4	-14.85686	6.5	-14.85686
6.6	-14.85979	6.7	-14.85827	6.8	-14.85827	6.9	-14.85827
7	-14.85827	7.1	-14.85827	7.2	-14.86053	7.3	-14.85911
7.4	-14.85911	7.5	-14.85982	7.6	-14.85985	7.7	-14.8602
7.8	-14.8602	7.9	-14.86013	8	-14.8602		
$y = 6.3$							
5	-14.85677	5.1	-14.85677	5.2	-14.8567	5.3	-14.85683
5.4	-14.85685	5.5	-14.85685	5.6	-14.85685	5.7	-14.85643
5.8	-14.85643	5.9	-14.85729	6	-14.8588	6.1	-14.8588
6.2	-14.85891	6.3	-14.85688	6.4	-14.85718	6.5	-14.85718
6.6	-14.85776	6.7	-14.85776	6.8	-14.85776	6.9	-14.85776
7	-14.85776	7.1	-14.85798	7.2	-14.85846	7.3	-14.85866
7.4	-14.85934	7.5	-14.86057	7.6	-14.85885	7.7	-14.85946
7.8	-14.85946	7.9	-14.85961	8	-14.85962		
$y = 6.4$							
5	-14.85703	5.1	-14.85703	5.2	-14.85685	5.3	-14.85628
5.4	-14.85723	5.5	-14.85874	5.6	-14.85874	5.7	-14.85673
5.8	-14.85725	5.9	-14.85697	6	-14.85676	6.1	-14.85676
6.2	-14.85743	6.3	-14.85743	6.4	-14.85732	6.5	-14.85927
6.6	-14.85797	6.7	-14.85797	6.8	-14.85797	6.9	-14.85787
7	-14.85972	7.1	-14.85841	7.2	-14.85824	7.3	-14.86029
7.4	-14.85892	7.5	-14.85892	7.6	-14.85892	7.7	-14.85907

7.8	-14.85952	7.9	-14.85981	8	-14.85981		
$y = 6.5$							
5	-14.85657	5.1	-14.85651	5.2	-14.85651	5.3	-14.85651
5.4	-14.85651	5.5	-14.85651	5.6	-14.85673	5.7	-14.85693
5.8	-14.85656	5.9	-14.8586	6	-14.85677	6.1	-14.8591
6.2	-14.8591	6.3	-14.85693	6.4	-14.85736	6.5	-14.85845
6.6	-14.85698	6.7	-14.85764	6.8	-14.85764	6.9	-14.85729
7	-14.86	7.1	-14.86	7.2	-14.86	7.3	-14.86
7.4	-14.85846	7.5	-14.8592	7.6	-14.8592	7.7	-14.85907
7.8	-14.86063	7.9	-14.85966	8	-14.85966		
$y = 6.6$							
5	-14.85626	5.1	-14.85626	5.2	-14.85638	5.3	-14.85591
5.4	-14.85627	5.5	-14.85589	5.6	-14.85589	5.7	-14.85662
5.8	-14.85654	5.9	-14.85639	6	-14.85843	6.1	-14.85843
6.2	-14.85843	6.3	-14.85736	6.4	-14.85684	6.5	-14.85684
6.6	-14.85698	6.7	-14.85698	6.8	-14.85719	6.9	-14.85719
7	-14.85719	7.1	-14.85719	7.2	-14.85815	7.3	-14.85785
7.4	-14.8583	7.5	-14.85827	7.6	-14.85903	7.7	-14.85897
7.8	-14.85916	7.9	-14.85901	8	-14.85906		
$y = 6.7$							
5	-14.85637	5.1	-14.8563	5.2	-14.85612	5.3	-14.85612
5.4	-14.85818	5.5	-14.85818	5.6	-14.85566	5.7	-14.85566
5.8	-14.85584	5.9	-14.85584	6	-14.85584	6.1	-14.85603
6.2	-14.85603	6.3	-14.85603	6.4	-14.85689	6.5	-14.85689
6.6	-14.85641	6.7	-14.85641	6.8	-14.8574	6.9	-14.85641
7	-14.85719	7.1	-14.85695	7.2	-14.85695	7.3	-14.85799
7.4	-14.85773	7.5	-14.85978	7.6	-14.85741	7.7	-14.85877
7.8	-14.85877	7.9	-14.85863	8	-14.86035		
$y = 6.8$							
5	-14.85548	5.1	-14.85483	5.2	-14.85568	5.3	-14.85568
5.4	-14.85568	5.5	-14.85568	5.6	-14.85664	5.7	-14.85664
5.8	-14.85664	5.9	-14.85664	6	-14.85664	6.1	-14.85664
6.2	-14.85641	6.3	-14.85549	6.4	-14.85549	6.5	-14.85805
6.6	-14.85805	6.7	-14.85594	6.8	-14.85666	6.9	-14.85639
7	-14.85639	7.1	-14.85671	7.2	-14.85671	7.3	-14.85691
7.4	-14.85699	7.5	-14.85815	7.6	-14.85815	7.7	-14.85815
7.8	-14.85815	7.9	-14.85815	8	-14.85815		
$y = 6.9$							
5	-14.85495	5.1	-14.85498	5.2	-14.85543	5.3	-14.85679
5.4	-14.85495	5.5	-14.85453	5.6	-14.85499	5.7	-14.85488
5.8	-14.85488	5.9	-14.8553	6	-14.8553	6.1	-14.85476
6.2	-14.85698	6.3	-14.85578	6.4	-14.85577	6.5	-14.85548
6.6	-14.85519	6.7	-14.85555	6.8	-14.85606	6.9	-14.85719
7	-14.8562	7.1	-14.8562	7.2	-14.85809	7.3	-14.85622
7.4	-14.85622	7.5	-14.85674	7.6	-14.85674	7.7	-14.85889
7.8	-14.85685	7.9	-14.85695	8	-14.85695		

$y = 7$							
5	-14.85457	5.1	-14.85491	5.2	-14.85529	5.3	-14.85658
5.4	-14.85427	5.5	-14.85509	5.6	-14.85509	5.7	-14.85499
5.8	-14.85499	5.9	-14.85455	6	-14.85455	6.1	-14.85519
6.2	-14.85443	6.3	-14.85613	6.4	-14.85655	6.5	-14.85452
6.6	-14.85253	6.7	-14.85552	6.8	-14.85694	6.9	-14.85694
7	-14.85694	7.1	-14.85595	7.2	-14.85595	7.3	-14.85595
7.4	-14.85818	7.5	-14.85658	7.6	-14.85638	7.7	-14.85584
7.8	-14.85696	7.9	-14.85642	8	-14.85642		

A.2.2 Face Milling

Table A.2.2.1: Plate #2 (For the population, $N = 410$, $l_0 = 0.000021$, $m_0 = 0.000857$, $z_0 = -14.15217$, $h_t = 0.001866$ in).

x	z	x	z	x	z	x	z
$y = 1.0$							
2.0	-14.15171	2.1	-14.15163	2.2	-14.15158	2.3	-14.15158
7.7	-14.15151	7.8	-14.15169	7.9	-14.15169	8.0	-14.15177
$y = 1.1$							
2.0	-14.15164	2.1	-14.15171	2.2	-14.1515	2.3	-14.1515
7.7	-14.15148	7.8	-14.15144	7.9	-14.15147	8.0	-14.15177
$y = 1.2$							
2.0	-14.15163	2.1	-14.15152	2.2	-14.15148	2.3	-14.15148
7.7	-14.15132	7.8	-14.15144	7.9	-14.15144	8.0	-14.1518
$y = 1.3$							
2.0	-14.15136	2.1	-14.15147	2.2	-14.15139	2.3	-14.15139
7.7	-14.15118	7.8	-14.15138	7.9	-14.1512	8.0	-14.15165
$y = 1.4$							
2.0	-14.15139	2.1	-14.15124	2.2	-14.15126	2.3	-14.15126
7.7	-14.15115	7.8	-14.1512	7.9	-14.15112	8.0	-14.15162
$y = 1.5$							
2.0	-14.15128	2.1	-14.15125	2.2	-14.15131	2.3	-14.15125
3.4	-14.15081	3.5	-14.15063	3.6	-14.1506	3.7	-14.15067
3.8	-14.15055	3.9	-14.15079	4.0	-14.15079	4.1	-14.15043
4.2	-14.15043	4.3	-14.15043	4.4	-14.15043	5.6	-14.15047
5.7	-14.15051	5.8	-14.15025	5.9	-14.15025	6.0	-14.15027
6.1	-14.1503	6.2	-14.1502	6.3	-14.15043	6.4	-14.15024
6.5	-14.15039	6.6	-14.15044	7.7	-14.15097	7.8	-14.15135
7.9	-14.15113	8.0	-14.15171				
$y = 1.6$							
2.0	-14.15123	2.1	-14.15107	2.2	-14.15125	2.3	-14.15125
3.4	-14.1507	3.5	-14.15063	3.6	-14.15068	3.7	-14.15035
3.8	-14.15045	3.9	-14.15052	4.0	-14.15048	4.1	-14.15038
4.2	-14.15037	4.3	-14.15042	4.4	-14.15043	5.6	-14.15034

5.7	-14.15046	5.8	-14.15041	5.9	-14.15022	6.0	-14.15027
6.1	-14.15036	6.2	-14.15032	6.3	-14.1504	6.4	-14.15022
6.5	-14.15039	6.6	-14.15057	7.7	-14.15084	7.8	-14.1511
7.9	-14.15125	8.0	-14.15151				
$y = 1.7$							
2.0	-14.15134	2.1	-14.1511	2.2	-14.15097	2.3	-14.15112
3.4	-14.15072	3.5	-14.15039	3.6	-14.15054	3.7	-14.15036
3.8	-14.15034	3.9	-14.15046	4.0	-14.15037	4.1	-14.15037
4.2	-14.15041	4.3	-14.15041	4.4	-14.15042	5.6	-14.15029
5.7	-14.15031	5.8	-14.15019	5.9	-14.1502	6.0	-14.1502
6.1	-14.15018	6.2	-14.15017	6.3	-14.15028	6.4	-14.1501
6.5	-14.15031	6.6	-14.15044	7.7	-14.15089	7.8	-14.15111
7.9	-14.15118	8.0	-14.15164				
$y = 1.8$							
2.0	-14.15119	2.1	-14.15092	2.2	-14.15107	2.3	-14.15089
3.4	-14.15053	3.5	-14.15022	3.6	-14.15035	3.7	-14.1501
3.8	-14.15046	3.9	-14.1504	4.0	-14.15046	4.1	-14.15028
4.2	-14.15025	4.3	-14.15025	4.4	-14.15045	5.6	-14.15002
5.7	-14.15013	5.8	-14.14978	5.9	-14.15001	6.0	-14.14995
6.1	-14.15017	6.2	-14.14988	6.3	-14.15007	6.4	-14.14994
6.5	-14.15024	6.6	-14.15036	7.7	-14.15105	7.8	-14.15089
7.9	-14.15105	8.0	-14.15138				
$y = 1.9$							
2.0	-14.15101	2.1	-14.15095	2.2	-14.15101	2.3	-14.15105
3.4	-14.15042	3.5	-14.15003	3.6	-14.15039	3.7	-14.15001
3.8	-14.15042	3.9	-14.15011	4.0	-14.15033	4.1	-14.15015
4.2	-14.15012	4.3	-14.15028	4.4	-14.15032	5.6	-14.1499
5.7	-14.15025	5.8	-14.14973	5.9	-14.14973	6.0	-14.15015
6.1	-14.15012	6.2	-14.14964	6.3	-14.14993	6.4	-14.14987
6.5	-14.1503	6.6	-14.15015	7.7	-14.15082	7.8	-14.15075
7.9	-14.15075	8.0	-14.15119				
$y = 2.0$							
2.0	-14.15109	2.1	-14.15105	2.2	-14.15076	2.3	-14.1509
3.4	-14.1503	3.5	-14.14993	3.6	-14.15019	3.7	-14.14999
3.8	-14.15013	3.9	-14.15033	4.0	-14.15016	4.1	-14.15009
4.2	-14.15009	4.3	-14.15006	4.4	-14.15025	5.6	-14.14968
5.7	-14.15002	5.8	-14.14988	5.9	-14.14988	6.0	-14.14995
6.1	-14.1498	6.2	-14.14979	6.3	-14.15004	6.4	-14.14976
6.5	-14.1499	6.6	-14.15013	7.7	-14.1508	7.8	-14.15086
7.9	-14.15086	8.0	-14.15095				
$y = 2.1$							
2.0	-14.15086	2.1	-14.15092	2.2	-14.15065	2.3	-14.15065
3.4	-14.15036	3.5	-14.14972	3.6	-14.15013	3.7	-14.14974
3.8	-14.15021	3.9	-14.15021	4.0	-14.14999	4.1	-14.15002
4.2	-14.15002	4.3	-14.1498	4.4	-14.15022	5.6	-14.14956
5.7	-14.1499	5.8	-14.14953	5.9	-14.14953	6.0	-14.14988

6.1	-14.14969	6.2	-14.1496	6.3	-14.14968	6.4	-14.14965
6.5	-14.14975	6.6	-14.14996	7.7	-14.15053	7.8	-14.15079
7.9	-14.15079	8.0	-14.15088				
$y = 2.2$							
2.0	-14.1509	2.1	-14.15095	2.2	-14.15046	2.3	-14.15046
3.4	-14.15033	3.5	-14.14969	3.6	-14.15009	3.7	-14.14963
3.8	-14.15006	3.9	-14.15002	4.0	-14.15006	4.1	-14.15008
4.2	-14.14985	4.3	-14.14973	4.4	-14.15001	5.6	-14.14947
5.7	-14.14975	5.8	-14.14953	5.9	-14.14969	6.0	-14.14972
6.1	-14.14975	6.2	-14.14965	6.3	-14.1496	6.4	-14.14965
6.5	-14.14972	6.6	-14.15	7.7	-14.1504	7.8	-14.15082
7.9	-14.15085	8.0	-14.15089				
$y = 2.3$							
2.0	-14.15075	2.1	-14.15073	2.2	-14.15026	2.3	-14.15072
3.4	-14.15009	3.5	-14.14958	3.6	-14.15013	3.7	-14.14955
3.8	-14.14955	3.9	-14.14989	4.0	-14.14986	4.1	-14.14985
4.2	-14.14964	4.3	-14.14962	4.4	-14.14999	5.6	-14.14932
5.7	-14.14969	5.8	-14.14936	5.9	-14.14963	6.0	-14.14966
6.1	-14.14968	6.2	-14.14932	6.3	-14.14961	6.4	-14.1495
6.5	-14.14959	6.6	-14.14974	7.7	-14.15057	7.8	-14.15065
7.9	-14.15083	8.0	-14.15072				
$y = 2.4$							
2.0	-14.15054	2.1	-14.15066	2.2	-14.15001	2.3	-14.15059
3.4	-14.15011	3.5	-14.14983	3.6	-14.15001	3.7	-14.14961
3.8	-14.15004	3.9	-14.14984	4.0	-14.14962	4.1	-14.14962
4.2	-14.14958	4.3	-14.14957	4.4	-14.14995	5.6	-14.14936
5.7	-14.14925	5.8	-14.14924	5.9	-14.14927	6.0	-14.14924
6.1	-14.14927	6.2	-14.14925	6.3	-14.14953	6.4	-14.14936
6.5	-14.14966	6.6	-14.14944	7.7	-14.15049	7.8	-14.15049
7.9	-14.15053	8.0	-14.15078				
$y = 2.5$							
2.0	-14.15047	2.1	-14.15071	2.2	-14.15042	2.3	-14.15056
3.4	-14.14977	3.5	-14.14967	3.6	-14.14963	3.7	-14.1496
3.8	-14.14962	3.9	-14.14972	4.0	-14.14961	4.1	-14.14967
4.2	-14.14943	4.3	-14.14941	4.4	-14.14982	5.6	-14.14931
5.7	-14.14929	5.8	-14.14919	5.9	-14.14926	6.0	-14.14936
6.1	-14.14934	6.2	-14.1492	6.3	-14.14925	6.4	-14.14931
6.5	-14.14919	6.6	-14.14941	7.7	-14.1502	7.8	-14.15047
7.9	-14.15061	8.0	-14.15064				
$y = 2.6$							
2.0	-14.15057	2.1	-14.15041	2.2	-14.15025	2.3	-14.15049
7.7	-14.15022	7.8	-14.15039	7.9	-14.15059	8.0	-14.15045
$y = 2.7$							
2.0	-14.15027	2.1	-14.15021	2.2	-14.15039	2.3	-14.15039
7.7	-14.15002	7.8	-14.1503	7.9	-14.15037	8.0	-14.15047
$y = 2.8$							

2.0	-14.15023	2.1	-14.15041	2.2	-14.15029	2.3	-14.15029
7.7	-14.14989	7.8	-14.15035	7.9	-14.15038	8.0	-14.15042
$y = 2.9$							
2.0	-14.15025	2.1	-14.15013	2.2	-14.15008	2.3	-14.15008
7.7	-14.14997	7.8	-14.15029	7.9	-14.15016	8.0	-14.15019
$y = 3.0$							
2.0	-14.15011	2.1	-14.14989	2.2	-14.14999	2.3	-14.14993
7.7	-14.14992	7.8	-14.15011	7.9	-14.15017	8.0	-14.15013

Table A.2.2.2: Plate #9 (N = 410, l_0 = -0.000022, m_0 = -0.000467, z_0 = -14.14393, h_t = 0.001707).

x	z	x	z	x	z	x	z
$y = 1.0$							
2.0	-14.1451	2.1	-14.14488	2.2	-14.14494	2.3	-14.14483
7.7	-14.14523	7.8	-14.14523	7.9	-14.14547	8.0	-14.1453
$y = 1.1$							
2.0	-14.14511	2.1	-14.14497	2.2	-14.14491	2.3	-14.1449
7.7	-14.14517	7.8	-14.14513	7.9	-14.1455	8.0	-14.14557
$y = 1.2$							
2.0	-14.14516	2.1	-14.14518	2.2	-14.14524	2.3	-14.14504
7.7	-14.14524	7.8	-14.14524	7.9	-14.14548	8.0	-14.14527
$y = 1.3$							
2.0	-14.14504	2.1	-14.14507	2.2	-14.14497	2.3	-14.14497
7.7	-14.14521	7.8	-14.14521	7.9	-14.14556	8.0	-14.14526
$y = 1.4$							
2.0	-14.14535	2.1	-14.14507	2.2	-14.14518	2.3	-14.14498
7.7	-14.14526	7.8	-14.14526	7.9	-14.14542	8.0	-14.14537
$y = 1.5$							
2.0	-14.14525	2.1	-14.14528	2.2	-14.14502	2.3	-14.14497
3.4	-14.1442	3.5	-14.14419	3.6	-14.14412	3.7	-14.14414
3.8	-14.14411	3.9	-14.14411	4.0	-14.14413	4.1	-14.14422
4.2	-14.14408	4.3	-14.1441	4.4	-14.14416	5.6	-14.14408
5.7	-14.14411	5.8	-14.1442	5.9	-14.14421	6.0	-14.14435
6.1	-14.14435	6.2	-14.14436	6.3	-14.14436	6.4	-14.14436
6.5	-14.14434	6.6	-14.14448	7.7	-14.14544	7.8	-14.14553
7.9	-14.1456	8.0	-14.14573				
$y = 1.6$							
2.0	-14.14525	2.1	-14.14509	2.2	-14.14532	2.3	-14.14513
3.4	-14.14433	3.5	-14.14426	3.6	-14.14424	3.7	-14.14418
3.8	-14.14434	3.9	-14.1442	4.0	-14.14438	4.1	-14.14438
4.2	-14.14419	4.3	-14.14415	4.4	-14.14415	5.6	-14.14433
5.7	-14.14428	5.8	-14.14431	5.9	-14.14432	6.0	-14.14448
6.1	-14.14448	6.2	-14.14451	6.3	-14.14451	6.4	-14.14451
6.5	-14.14467	6.6	-14.14467	7.7	-14.14555	7.8	-14.14558

7.9	-14.1456	8.0	-14.14546				
$y = 1.7$							
2.0	-14.14538	2.1	-14.14528	2.2	-14.14526	2.3	-14.14537
3.4	-14.14457	3.5	-14.1446	3.6	-14.1446	3.7	-14.14428
3.8	-14.14438	3.9	-14.14432	4.0	-14.14432	4.1	-14.14423
4.2	-14.14406	4.3	-14.14417	4.4	-14.14437	5.6	-14.1444
5.7	-14.1445	5.8	-14.14443	5.9	-14.14443	6.0	-14.14446
6.1	-14.14456	6.2	-14.14453	6.3	-14.14453	6.4	-14.14453
6.5	-14.14466	6.6	-14.14462	7.7	-14.14533	7.8	-14.14526
7.9	-14.1455	8.0	-14.14553				
$y = 1.8$							
2.0	-14.14549	2.1	-14.14531	2.2	-14.14526	2.3	-14.14537
3.4	-14.14443	3.5	-14.14469	3.6	-14.14441	3.7	-14.14439
3.8	-14.14439	3.9	-14.14439	4.0	-14.14431	4.1	-14.14431
4.2	-14.14412	4.3	-14.14435	4.4	-14.14427	5.6	-14.14439
5.7	-14.14439	5.8	-14.1446	5.9	-14.14452	6.0	-14.14451
6.1	-14.14459	6.2	-14.14488	6.3	-14.14488	6.4	-14.14488
6.5	-14.14488	6.6	-14.14488	7.7	-14.1453	7.8	-14.14531
7.9	-14.14573	8.0	-14.14575				
$y = 1.9$							
2.0	-14.14549	2.1	-14.14561	2.2	-14.14531	2.3	-14.14548
3.4	-14.14451	3.5	-14.1448	3.6	-14.1446	3.7	-14.14448
3.8	-14.14448	3.9	-14.14448	4.0	-14.14448	4.1	-14.14448
4.2	-14.14431	4.3	-14.14431	4.4	-14.14431	5.6	-14.14459
5.7	-14.14467	5.8	-14.14446	5.9	-14.14467	6.0	-14.14466
6.1	-14.1446	6.2	-14.14467	6.3	-14.14461	6.4	-14.1447
6.5	-14.1447	6.6	-14.14474	7.7	-14.14549	7.8	-14.14548
7.9	-14.14549	8.0	-14.14535				
$y = 2.0$							
2.0	-14.14555	2.1	-14.14534	2.2	-14.1454	2.3	-14.14534
3.4	-14.1446	3.5	-14.1448	3.6	-14.14452	3.7	-14.14455
3.8	-14.14452	3.9	-14.14442	4.0	-14.14442	4.1	-14.14454
4.2	-14.1444	4.3	-14.14443	4.4	-14.1444	5.6	-14.14459
5.7	-14.14455	5.8	-14.1446	5.9	-14.14453	6.0	-14.14453
6.1	-14.14473	6.2	-14.14473	6.3	-14.14471	6.4	-14.14473
6.5	-14.14465	6.6	-14.14479	7.7	-14.14534	7.8	-14.14564
7.9	-14.14575	8.0	-14.14575				
$y = 2.1$							
2.0	-14.14552	2.1	-14.14555	2.2	-14.1454	2.3	-14.14555
3.4	-14.14472	3.5	-14.14472	3.6	-14.14465	3.7	-14.14472
3.8	-14.14472	3.9	-14.14458	4.0	-14.14445	4.1	-14.14445
4.2	-14.14446	4.3	-14.14457	4.4	-14.14446	5.6	-14.14467
5.7	-14.14459	5.8	-14.14468	5.9	-14.1447	6.0	-14.1447
6.1	-14.14477	6.2	-14.14474	6.3	-14.1447	6.4	-14.14483
6.5	-14.14492	6.6	-14.14484	7.7	-14.14561	7.8	-14.14539
7.9	-14.14571	8.0	-14.14547				

$y = 2.2$							
2.0	-14.14574	2.1	-14.14557	2.2	-14.14559	2.3	-14.14551
3.4	-14.14471	3.5	-14.14486	3.6	-14.14464	3.7	-14.14466
3.8	-14.14466	3.9	-14.14466	4.0	-14.14457	4.1	-14.14457
4.2	-14.14458	4.3	-14.14458	4.4	-14.14458	5.6	-14.14477
5.7	-14.14473	5.8	-14.14468	5.9	-14.14458	6.0	-14.14463
6.1	-14.14474	6.2	-14.14491	6.3	-14.1449	6.4	-14.14487
6.5	-14.14485	6.6	-14.14507	7.7	-14.14531	7.8	-14.14569
7.9	-14.14542	8.0	-14.14542				
$y = 2.3$							
2.0	-14.14566	2.1	-14.14557	2.2	-14.14565	2.3	-14.14551
3.4	-14.14493	3.5	-14.14483	3.6	-14.14474	3.7	-14.14475
3.8	-14.14478	3.9	-14.1447	4.0	-14.1447	4.1	-14.1449
4.2	-14.14476	4.3	-14.14468	4.4	-14.14468	5.6	-14.14485
5.7	-14.14485	5.8	-14.14476	5.9	-14.14477	6.0	-14.1448
6.1	-14.14491	6.2	-14.14488	6.3	-14.14491	6.4	-14.14495
6.5	-14.14495	6.6	-14.14495	7.7	-14.14546	7.8	-14.14566
7.9	-14.14556	8.0	-14.14579				
$y = 2.4$							
2.0	-14.14586	2.1	-14.14574	2.2	-14.14564	2.3	-14.14553
3.4	-14.14499	3.5	-14.14506	3.6	-14.145	3.7	-14.14489
3.8	-14.14486	3.9	-14.14484	4.0	-14.14484	4.1	-14.14486
4.2	-14.14472	4.3	-14.14478	4.4	-14.14472	5.6	-14.1449
5.7	-14.14476	5.8	-14.14492	5.9	-14.14496	6.0	-14.14477
6.1	-14.14493	6.2	-14.14498	6.3	-14.14507	6.4	-14.14491
6.5	-14.14503	6.6	-14.14503	7.7	-14.1456	7.8	-14.14554
7.9	-14.14589	8.0	-14.14551				
$y = 2.5$							
2.0	-14.14595	2.1	-14.14572	2.2	-14.14595	2.3	-14.14567
3.4	-14.14505	3.5	-14.14502	3.6	-14.1448	3.7	-14.14495
3.8	-14.14475	3.9	-14.14476	4.0	-14.14476	4.1	-14.14477
4.2	-14.14474	4.3	-14.14476	4.4	-14.14487	5.6	-14.14491
5.7	-14.14494	5.8	-14.14502	5.9	-14.14491	6.0	-14.14498
6.1	-14.14487	6.2	-14.14494	6.3	-14.14502	6.4	-14.14493
6.5	-14.14519	6.6	-14.14511	7.7	-14.14575	7.8	-14.14585
7.9	-14.14555	8.0	-14.14589				
$y = 2.6$							
2.0	-14.14583	2.1	-14.1459	2.2	-14.14582	2.3	-14.1459
7.7	-14.14558	7.8	-14.14584	7.9	-14.1456	8.0	-14.1456
$y = 2.7$							
2.0	-14.14596	2.1	-14.14592	2.2	-14.14572	2.3	-14.14589
7.7	-14.14575	7.8	-14.14577	7.9	-14.14566	8.0	-14.1459
$y = 2.8$							
2.0	-14.14595	2.1	-14.14606	2.2	-14.14595	2.3	-14.14582
7.7	-14.14551	7.8	-14.14546	7.9	-14.14592	8.0	-14.14592
$y = 2.9$							

2.0	-14.14607	2.1	-14.1461	2.2	-14.1458	2.3	-14.14596
7.7	-14.14574	7.8	-14.14562	7.9	-14.14581	8.0	-14.14562
$y = 3.0$							
2.0	-14.14604	2.1	-14.14597	2.2	-14.14581	2.3	-14.14579
7.7	-14.14566	7.8	-14.14557	7.9	-14.14588	8.0	-14.14587

Table A.2.2.3: Plate #11 ($N = 410$, $l_0 = -0.000028$, $m_0 = -0.000086$, $z_0 = -14.14412$, $h_t = 0.002758$).

x	z	x	z	x	z	x	z
$y = 1.0$							
2.0	-14.14519	2.1	-14.14524	2.2	-14.14508	2.3	-14.14501
7.7	-14.14543	7.8	-14.14543	7.9	-14.14563	8.0	-14.14588
$y = 1.1$							
2.0	-14.14523	2.1	-14.14498	2.2	-14.14513	2.3	-14.14499
7.7	-14.14553	7.8	-14.14553	7.9	-14.14559	8.0	-14.1458
$y = 1.2$							
2.0	-14.1453	2.1	-14.1453	2.2	-14.14517	2.3	-14.14507
7.7	-14.14551	7.8	-14.14551	7.9	-14.14563	8.0	-14.14578
$y = 1.3$							
2.0	-14.14517	2.1	-14.14512	2.2	-14.14512	2.3	-14.14512
7.7	-14.14553	7.8	-14.14553	7.9	-14.14577	8.0	-14.14577
$y = 1.4$							
2.0	-14.14534	2.1	-14.14534	2.2	-14.14534	2.3	-14.14505
7.7	-14.14544	7.8	-14.14544	7.9	-14.14584	8.0	-14.14584
$y = 1.5$							
2.0	-14.14539	2.1	-14.14539	2.2	-14.14518	2.3	-14.14514
3.4	-14.14388	3.5	-14.14368	3.6	-14.144	3.7	-14.14361
3.8	-14.14361	3.9	-14.14361	4.0	-14.14361	4.1	-14.14333
4.2	-14.14333	4.3	-14.14333	4.4	-14.14322	5.6	-14.14333
5.7	-14.14352	5.8	-14.14347	5.9	-14.14343	6.0	-14.14343
6.1	-14.1437	6.2	-14.1437	6.3	-14.1438	6.4	-14.14398
6.5	-14.14394	6.6	-14.14394	7.7	-14.14556	7.8	-14.14556
7.9	-14.1456	8.0	-14.14572				
$y = 1.6$							
2.0	-14.1439	2.1	-14.1439	2.2	-14.1439	2.3	-14.14342
3.4	-14.14353	3.5	-14.14338	3.6	-14.14346	3.7	-14.14335
3.8	-14.14335	3.9	-14.14317	4.0	-14.1434	4.1	-14.1454
4.2	-14.14534	4.3	-14.14507	4.4	-14.14507	5.6	-14.14342
5.7	-14.14337	5.8	-14.14358	5.9	-14.14346	6.0	-14.14364
6.1	-14.14364	6.2	-14.14378	6.3	-14.14379	6.4	-14.14374
6.5	-14.14398	6.6	-14.14406	7.7	-14.14546	7.8	-14.14546
7.9	-14.14557	8.0	-14.14579				
$y = 1.7$							
2.0	-14.14514	2.1	-14.14518	2.2	-14.14518	2.3	-14.14518

3.4	-14.14405	3.5	-14.14405	3.6	-14.14365	3.7	-14.14365
3.8	-14.1435	3.9	-14.14345	4.0	-14.14345	4.1	-14.14339
4.2	-14.14339	4.3	-14.14339	4.4	-14.14327	5.6	-14.14347
5.7	-14.14355	5.8	-14.14347	5.9	-14.14347	6.0	-14.14377
6.1	-14.14378	6.2	-14.14378	6.3	-14.1438	6.4	-14.14406
6.5	-14.14406	6.6	-14.14407	7.7	-14.14557	7.8	-14.14557
7.9	-14.14552	8.0	-14.1457				
$y = 1.8$							
2.0	-14.14548	2.1	-14.14548	2.2	-14.14538	2.3	-14.14523
3.4	-14.144	3.5	-14.144	3.6	-14.14372	3.7	-14.14372
3.8	-14.14362	3.9	-14.14344	4.0	-14.14339	4.1	-14.14339
4.2	-14.14339	4.3	-14.14339	4.4	-14.14324	5.6	-14.14341
5.7	-14.14341	5.8	-14.14341	5.9	-14.14352	6.0	-14.14392
6.1	-14.14392	6.2	-14.14392	6.3	-14.14381	6.4	-14.14427
6.5	-14.14427	6.6	-14.1443	7.7	-14.14528	7.8	-14.14542
7.9	-14.1456	8.0	-14.1456				
$y = 1.9$							
2.0	-14.14548	2.1	-14.14548	2.2	-14.14529	2.3	-14.14521
3.4	-14.14401	3.5	-14.14401	3.6	-14.14401	3.7	-14.14401
3.8	-14.1437	3.9	-14.14348	4.0	-14.14331	4.1	-14.14331
4.2	-14.14331	4.3	-14.14327	4.4	-14.14325	5.6	-14.1436
5.7	-14.14357	5.8	-14.14368	5.9	-14.14372	6.0	-14.14372
6.1	-14.14372	6.2	-14.14393	6.3	-14.14393	6.4	-14.14529
6.5	-14.14529	6.6	-14.14539	7.7	-14.14539	7.8	-14.14541
7.9	-14.14557	8.0	-14.14568				
$y = 2.0$							
2.0	-14.14553	2.1	-14.14553	2.2	-14.14553	2.3	-14.14524
3.4	-14.14415	3.5	-14.14415	3.6	-14.14415	3.7	-14.14369
3.8	-14.14354	3.9	-14.14354	4.0	-14.14341	4.1	-14.14341
4.2	-14.14341	4.3	-14.14344	4.4	-14.14335	5.6	-14.14348
5.7	-14.14362	5.8	-14.14353	5.9	-14.14364	6.0	-14.14378
6.1	-14.14382	6.2	-14.14393	6.3	-14.14393	6.4	-14.14393
6.5	-14.14412	6.6	-14.14428	7.7	-14.14536	7.8	-14.14536
7.9	-14.14562	8.0	-14.14554				
$y = 2.1$							
2.0	-14.14554	2.1	-14.14554	2.2	-14.14551	2.3	-14.14542
3.4	-14.14398	3.5	-14.14421	3.6	-14.14421	3.7	-14.14367
3.8	-14.14354	3.9	-14.14354	4.0	-14.14341	4.1	-14.14341
4.2	-14.14341	4.3	-14.14337	4.4	-14.14329	5.6	-14.14348
5.7	-14.14361	5.8	-14.14363	5.9	-14.14372	6.0	-14.14372
6.1	-14.14372	6.2	-14.14381	6.3	-14.14381	6.4	-14.14389
6.5	-14.14414	6.6	-14.14418	7.7	-14.14536	7.8	-14.14536
7.9	-14.14548	8.0	-14.1456				
$y = 2.2$							
2.0	-14.1455	2.1	-14.1455	2.2	-14.14551	2.3	-14.14528
3.4	-14.14406	3.5	-14.14399	3.6	-14.14399	3.7	-14.14373

3.8	-14.14366	3.9	-14.14349	4.0	-14.14346	4.1	-14.14349
4.2	-14.14336	4.3	-14.14341	4.4	-14.14337	5.6	-14.14356
5.7	-14.14359	5.8	-14.14369	5.9	-14.1436	6.0	-14.14387
6.1	-14.14387	6.2	-14.14393	6.3	-14.14393	6.4	-14.1442
6.5	-14.1442	6.6	-14.14421	7.7	-14.1453	7.8	-14.1453
7.9	-14.14555	8.0	-14.14567				
$y = 2.3$							
2.0	-14.14552	2.1	-14.14552	2.2	-14.14558	2.3	-14.14538
3.4	-14.14423	3.5	-14.14423	3.6	-14.14423	3.7	-14.14372
3.8	-14.14353	3.9	-14.14346	4.0	-14.14361	4.1	-14.14346
4.2	-14.14334	4.3	-14.14345	4.4	-14.14323	5.6	-14.14353
5.7	-14.14352	5.8	-14.1436	5.9	-14.14358	6.0	-14.14381
6.1	-14.14381	6.2	-14.14387	6.3	-14.14387	6.4	-14.14407
6.5	-14.14407	6.6	-14.14438	7.7	-14.1453	7.8	-14.14531
7.9	-14.14556	8.0	-14.14556				
$y = 2.4$							
2.0	-14.1457	2.1	-14.1457	2.2	-14.14553	2.3	-14.14542
3.4	-14.14408	3.5	-14.14414	3.6	-14.14414	3.7	-14.14368
3.8	-14.14381	3.9	-14.14349	4.0	-14.14349	4.1	-14.14337
4.2	-14.14337	4.3	-14.14348	4.4	-14.14336	5.6	-14.14366
5.7	-14.14368	5.8	-14.14358	5.9	-14.14372	6.0	-14.14375
6.1	-14.14374	6.2	-14.14392	6.3	-14.14392	6.4	-14.1441
6.5	-14.1441	6.6	-14.14443	7.7	-14.14523	7.8	-14.14536
7.9	-14.1456	8.0	-14.1456				
$y = 2.5$							
2.0	-14.14569	2.1	-14.14569	2.2	-14.1455	2.3	-14.14548
3.4	-14.14404	3.5	-14.14397	3.6	-14.14371	3.7	-14.14373
3.8	-14.14364	3.9	-14.1434	4.0	-14.14364	4.1	-14.1434
4.2	-14.14354	4.3	-14.14337	4.4	-14.1433	5.6	-14.14352
5.7	-14.14366	5.8	-14.14371	5.9	-14.14374	6.0	-14.14372
6.1	-14.14374	6.2	-14.14384	6.3	-14.14388	6.4	-14.14387
6.5	-14.1441	6.6	-14.14436	7.7	-14.14533	7.8	-14.14538
7.9	-14.14537	8.0	-14.14552				
$y = 2.6$							
2.0	-14.14571	2.1	-14.14571	2.2	-14.14557	2.3	-14.1455
7.7	-14.14536	7.8	-14.14536	7.9	-14.14539	8.0	-14.1454
$y = 2.7$							
2.0	-14.14572	2.1	-14.14572	2.2	-14.14551	2.3	-14.1455
7.7	-14.14525	7.8	-14.14525	7.9	-14.1454	8.0	-14.14545
$y = 2.8$							
2.0	-14.14575	2.1	-14.1456	2.2	-14.14564	2.3	-14.14546
7.7	-14.14524	7.8	-14.14524	7.9	-14.14534	8.0	-14.14556
$y = 2.9$							
2.0	-14.14587	2.1	-14.14587	2.2	-14.1456	2.3	-14.14555
7.7	-14.14522	7.8	-14.14522	7.9	-14.14527	8.0	-14.14549
$y = 3.0$							

2.0	-14.14598	2.1	-14.14579	2.2	-14.14561	2.3	-14.14543
7.7	-14.14507	7.8	-14.14523	7.9	-14.14531	8.0	-14.14547

Table A.2.2.4: Plate #5 (N = 410, l_0 = -0.000055, m_0 = 0.000130, z_0 = -14.15313, h_t = 0.002863).

x	z	x	z	x	z	x	z
$y = 1.0$							
2.0	-14.15414	2.1	-14.15415	2.2	-14.15394	2.3	-14.15382
7.7	-14.15452	7.8	-14.15457	7.9	-14.15478	8.0	-14.1549
$y = 1.1$							
2.0	-14.15425	2.1	-14.15413	2.2	-14.15405	2.3	-14.15397
7.7	-14.15471	7.8	-14.15469	7.9	-14.15469	8.0	-14.1549
$y = 1.2$							
2.0	-14.15422	2.1	-14.15411	2.2	-14.15411	2.3	-14.15387
7.7	-14.15459	7.8	-14.15459	7.9	-14.155	8.0	-14.155
$y = 1.3$							
2.0	-14.15418	2.1	-14.15409	2.2	-14.15401	2.3	-14.15402
7.7	-14.15443	7.8	-14.15443	7.9	-14.1547	8.0	-14.15484
$y = 1.4$							
2.0	-14.15412	2.1	-14.15411	2.2	-14.15403	2.3	-14.15388
7.7	-14.15437	7.8	-14.15446	7.9	-14.15446	8.0	-14.15482
$y = 1.5$							
2.0	-14.15425	2.1	-14.15403	2.2	-14.15399	2.3	-14.15396
3.4	-14.15272	3.5	-14.15261	3.6	-14.15241	3.7	-14.15221
3.8	-14.15226	3.9	-14.15213	4.0	-14.15208	4.1	-14.15206
4.2	-14.15204	4.3	-14.1519	4.4	-14.152	5.6	-14.15216
5.7	-14.15231	5.8	-14.15236	5.9	-14.15227	6.0	-14.15228
6.1	-14.15251	6.2	-14.15243	6.3	-14.15267	6.4	-14.15276
6.5	-14.15289	6.6	-14.15306	7.7	-14.15434	7.8	-14.15461
7.9	-14.15461	8.0	-14.15484				
$y = 1.6$							
2.0	-14.1542	2.1	-14.1542	2.2	-14.15409	2.3	-14.15403
3.4	-14.15276	3.5	-14.15241	3.6	-14.15241	3.7	-14.15225
3.8	-14.15232	3.9	-14.15226	4.0	-14.15205	4.1	-14.15211
4.2	-14.15213	4.3	-14.152	4.4	-14.15199	5.6	-14.1522
5.7	-14.1523	5.8	-14.15236	5.9	-14.15239	6.0	-14.15235
6.1	-14.15247	6.2	-14.15246	6.3	-14.15276	6.4	-14.15276
6.5	-14.15278	6.6	-14.15293	7.7	-14.1543	7.8	-14.15456
7.9	-14.15484	8.0	-14.15488				
$y = 1.7$							
2.0	-14.15433	2.1	-14.15419	2.2	-14.15402	2.3	-14.1539
3.4	-14.15256	3.5	-14.15256	3.6	-14.15243	3.7	-14.1523
3.8	-14.15223	3.9	-14.15232	4.0	-14.15216	4.1	-14.15214
4.2	-14.15211	4.3	-14.15219	4.4	-14.15203	5.6	-14.15229
5.7	-14.15229	5.8	-14.15229	5.9	-14.15232	6.0	-14.1525

6.1	-14.15238	6.2	-14.15259	6.3	-14.15268	6.4	-14.15268
6.5	-14.15278	6.6	-14.15302	7.7	-14.15433	7.8	-14.15439
7.9	-14.15456	8.0	-14.15491				
$y = 1.8$							
2.0	-14.15435	2.1	-14.15418	2.2	-14.15403	2.3	-14.15391
3.4	-14.15278	3.5	-14.15265	3.6	-14.15272	3.7	-14.15243
3.8	-14.1523	3.9	-14.15229	4.0	-14.15221	4.1	-14.15217
4.2	-14.15206	4.3	-14.15199	4.4	-14.15204	5.6	-14.1521
5.7	-14.1522	5.8	-14.15224	5.9	-14.15224	6.0	-14.15246
6.1	-14.15252	6.2	-14.15252	6.3	-14.15256	6.4	-14.15293
6.5	-14.15293	6.6	-14.15293	7.7	-14.15434	7.8	-14.15434
7.9	-14.15457	8.0	-14.15462				
$y = 1.9$							
2.0	-14.15425	2.1	-14.15417	2.2	-14.1541	2.3	-14.15413
3.4	-14.15273	3.5	-14.15257	3.6	-14.15241	3.7	-14.15242
3.8	-14.15242	3.9	-14.15223	4.0	-14.1523	4.1	-14.15217
4.2	-14.15215	4.3	-14.15209	4.4	-14.15213	5.6	-14.15216
5.7	-14.15218	5.8	-14.15229	5.9	-14.15231	6.0	-14.15231
6.1	-14.15256	6.2	-14.15245	6.3	-14.15258	6.4	-14.15281
6.5	-14.15281	6.6	-14.15296	7.7	-14.15442	7.8	-14.15442
7.9	-14.15437	8.0	-14.15473				
$y = 2.0$							
2.0	-14.15418	2.1	-14.15418	2.2	-14.15405	2.3	-14.154
3.4	-14.1527	3.5	-14.15255	3.6	-14.1525	3.7	-14.15233
3.8	-14.15233	3.9	-14.15224	4.0	-14.15228	4.1	-14.15219
4.2	-14.15206	4.3	-14.15203	4.4	-14.15211	5.6	-14.15218
5.7	-14.15224	5.8	-14.15232	5.9	-14.15232	6.0	-14.15239
6.1	-14.15234	6.2	-14.15256	6.3	-14.1526	6.4	-14.1526
6.5	-14.15258	6.6	-14.15287	7.7	-14.15428	7.8	-14.15441
7.9	-14.15454	8.0	-14.15466				
$y = 2.1$							
2.0	-14.1543	2.1	-14.15434	2.2	-14.15409	2.3	-14.15416
3.4	-14.15265	3.5	-14.15265	3.6	-14.15248	3.7	-14.15235
3.8	-14.15233	3.9	-14.15234	4.0	-14.15224	4.1	-14.15207
4.2	-14.15204	4.3	-14.15213	4.4	-14.15197	5.6	-14.15212
5.7	-14.15218	5.8	-14.1521	5.9	-14.15214	6.0	-14.15243
6.1	-14.15243	6.2	-14.15234	6.3	-14.15266	6.4	-14.15266
6.5	-14.15257	6.6	-14.15285	7.7	-14.15427	7.8	-14.1544
7.9	-14.15451	8.0	-14.15454				
$y = 2.2$							
2.0	-14.15424	2.1	-14.15424	2.2	-14.15407	2.3	-14.15414
3.4	-14.15266	3.5	-14.15259	3.6	-14.15243	3.7	-14.15241
3.8	-14.15229	3.9	-14.15229	4.0	-14.1522	4.1	-14.15218
4.2	-14.15218	4.3	-14.15209	4.4	-14.15203	5.6	-14.15208
5.7	-14.15223	5.8	-14.15216	5.9	-14.15227	6.0	-14.15223
6.1	-14.15234	6.2	-14.15234	6.3	-14.15254	6.4	-14.15254

6.5	-14.15282	6.6	-14.15293	7.7	-14.15433	7.8	-14.15456
7.9	-14.15456	8.0	-14.15451				
$y = 2.3$							
2.0	-14.15431	2.1	-14.15424	2.2	-14.15418	2.3	-14.15403
3.4	-14.15271	3.5	-14.15264	3.6	-14.15245	3.7	-14.15242
3.8	-14.15227	3.9	-14.15227	4.0	-14.15222	4.1	-14.15216
4.2	-14.15216	4.3	-14.15214	4.4	-14.15216	5.6	-14.15223
5.7	-14.15219	5.8	-14.15217	5.9	-14.15217	6.0	-14.15221
6.1	-14.15231	6.2	-14.15231	6.3	-14.15234	6.4	-14.15247
6.5	-14.15248	6.6	-14.15273	7.7	-14.154	7.8	-14.15439
7.9	-14.15443	8.0	-14.15454				
$y = 2.4$							
2.0	-14.1544	2.1	-14.15436	2.2	-14.15424	2.3	-14.15407
3.4	-14.1526	3.5	-14.1526	3.6	-14.15243	3.7	-14.15251
3.8	-14.15227	3.9	-14.15227	4.0	-14.15214	4.1	-14.15209
4.2	-14.15209	4.3	-14.15194	4.4	-14.15204	5.6	-14.15215
5.7	-14.15223	5.8	-14.15216	5.9	-14.15216	6.0	-14.15214
6.1	-14.15214	6.2	-14.15228	6.3	-14.15242	6.4	-14.15246
6.5	-14.15244	6.6	-14.15269	7.7	-14.15391	7.8	-14.15433
7.9	-14.15433	8.0	-14.15433				
$y = 2.5$							
2.0	-14.15423	2.1	-14.15424	2.2	-14.15417	2.3	-14.15408
3.4	-14.15262	3.5	-14.15243	3.6	-14.1524	3.7	-14.15243
3.8	-14.15231	3.9	-14.15227	4.0	-14.15218	4.1	-14.15211
4.2	-14.152	4.3	-14.15198	4.4	-14.15206	5.6	-14.15203
5.7	-14.15201	5.8	-14.15214	5.9	-14.15217	6.0	-14.1521
6.1	-14.15202	6.2	-14.15221	6.3	-14.1523	6.4	-14.15232
6.5	-14.15245	6.6	-14.15267	7.7	-14.15409	7.8	-14.15446
7.9	-14.15446	8.0	-14.15442				
$y = 2.6$							
2.0	-14.15435	2.1	-14.15435	2.2	-14.15413	2.3	-14.15414
7.7	-14.15415	7.8	-14.15415	7.9	-14.15447	8.0	-14.15452
$y = 2.7$							
2.0	-14.15433	2.1	-14.15433	2.2	-14.15416	2.3	-14.15404
7.7	-14.15385	7.8	-14.15405	7.9	-14.1543	8.0	-14.15451
$y = 2.8$							
2.0	-14.15423	2.1	-14.15423	2.2	-14.15407	2.3	-14.15393
7.7	-14.15378	7.8	-14.15406	7.9	-14.1543	8.0	-14.15439
$y = 2.9$							
2.0	-14.15428	2.1	-14.15413	2.2	-14.1541	2.3	-14.15392
7.7	-14.15388	7.8	-14.1539	7.9	-14.15429	8.0	-14.15429
$y = 3.0$							
2.0	-14.15424	2.1	-14.1543	2.2	-14.15407	2.3	-14.15391
7.7	-14.1539	7.8	-14.15392	7.9	-14.15425	8.0	-14.15423

APPENDIX B

JAVA CODES

B.1 Least Squares Line

```
/* Least Squares Fit Line
 * This program uses a class containing 3 instance variables l0, y0, & r, and
 * method calcLine that returns an object of that class. The method is returning
 * a single object, but it is actually calculating & returning all 3 values.
 */

import chapman.io.*;
class LSquaresLine {
   //declare variables
   private double l0, y0, r;
   // Constructor that does nothing
   public LSquaresLine() {
   }
   // Method to get slope
   public double getSlope(){
      return l0;
   }
   // Method to get intercept
   public double getIntercept(){
      return y0;
   }
   // Method to get correlation coefficient
   public double getCorrelation(){
      return r;
   }

   // Method calcLine
   public LSquaresLine calcLine( double x[], double y[], int nval) {

      double sumX = 0.0, sumY = 0.0;
      double sumX2 = 0.0, sumXY = 0.0, sumY2 = 0.0;
      double xbar, ybar, term;
      LSquaresLine result = new LSquaresLine();

      for (int i = 0; i < nval; i++) {
         sumX += x[i]; sumY += y[i];
         sumX2 += x[i] * x[i];
         sumXY += x[i] * y[i];
```

```
            sumY2 += y[i] * y[i];
        }

        xbar = sumX / nval;
        ybar = sumY / nval;
        l0 = (sumXY - sumX*ybar)/(sumX2 - sumX*xbar);
        y0 = ybar - l0*xbar;
        term = (nval*sumX2 - sumX*sumX)*(nval*sumY2 - sumY*sumY);
        r = (nval*sumXY - sumX*sumY)/Math.sqrt(term);
        return result;
    } //end method calcLine
}//end class LSquaresLine

public class LeastSquaresLine {
    public static void main(String[] args) {
        final int MAXVAL = 150; //max array size

        //Variables Declaration
        double x[] = new double[MAXVAL], y[] = new double[MAXVAL];//data arrays
        String fileName1; //input file name
        int j = 0; //index
        int n;//number of data points

        StdIn in = new StdIn(); //creates a StdIn object (Chapman.io)

        //Get input file name
        System.out.print("Enter data file name under forte4j dir: ");
        fileName1 = in.readString();
        System.out.println(fileName1);

        //Open file (Chapman.io package)
        FileIn in1 = new FileIn(fileName1);

        //check for valid data file open
        if (in1.readStatus != in1.FILE_NOT_FOUND) {
            //Read numbers into array
            while (in1.readStatus != in1.EOF) {
                j = in1.readInt();
                x[j] = in1.readDouble();
                y[j] = in1.readDouble();
            }//end while
            n = j + 1;
            in1.close();//Close data file

            LSquaresLine ls = new LSquaresLine(); //Create a LSquaresLine object
            ls.calcLine(x, y, n);//invoke method calcLine
```

```
        //Write the results in the format specified
        System.out.println("data points = "+ n);
        Fmt.printf("The slope l0 = %9.6f\n" , ls.getSlope());
        Fmt.printf("The intercept z0 = %9.5f\n" , ls.getIntercept());
        Fmt.printf("The correlation coefficient r = %6.3f\n" , ls.getCorrelation());

      } //end if
      else {//data file not found
        System.out.println("File " + fileName1 + " not found!");
      }
    } //end main method
} //end class LeastSquaresLine
```

B.2 Least Squares Plane

```
/* Least Squares Fit Plane
 * This program uses a class containing 3 instance variables l0, m0, & z0, and
 * method calcPlane that returns an object of that class. The method is returning
 * a single object, but it is actually calculating & returning all 3 values.
 */

import chapman.io.*;
class LSquaresPlane {
  //declare variables
  private double l0, m0, z0;
  // Constructor that does nothing
  public LSquaresPlane() {
  }
  // Method to get slopex
  public double getSlopex(){
    return l0;
  }
  // Method to get slopey
  public double getSlopey(){
    return m0;
  }
  // Method to get intercept
  public double getIntercept(){
    return z0;
  }

  // Method calcPlane
  public LSquaresPlane calcPlane(double x[],double y[],double z[],int nval) {
```

```
      double sumX = 0.0, sumY = 0.0, sumZ = 0.0;
      double sumX2 = 0.0, sumXY = 0.0, sumXZ = 0.0;
      double sumY2 = 0.0, sumYZ = 0.0;
      double term, term1, term2, term3;
      LSquaresPlane result = new LSquaresPlane();

      for (int i = 0; i < nval; i++) {
         sumX += x[i]; sumY += y[i];
         sumZ += z[i]; sumX2 += x[i] * x[i];
         sumXY += x[i] * y[i]; sumXZ += x[i] * z[i];
         sumY2 += y[i] * y[i]; sumYZ += y[i] * z[i];
      }
      term = nval*sumX2 - sumX*sumX;
      term1 = (nval*sumXZ - sumX*sumZ)*(nval*sumXY - sumX*sumY)/term;
      term2 = Math.pow((sumX*sumY - nval*sumXY),2)/(nval*sumX2 - sumX*sumX);
      m0 = (term1 + sumY*sumZ - nval*sumYZ)/(term2 + sumY*sumY - nval*sumY2);
      term3 = m0*(sumX*sumY - nval*sumXY) + nval*sumXZ - sumX*sumZ;
      l0 = term3/(nval*sumX2 - sumX*sumX);
      z0 = (sumZ - l0*sumX - m0*sumY)/nval;

      return result;
   } //end method calcPlane
}//end class LSquaresPlane

public class LeastSquaresPlane {
   public static void main(String[] args) {
      final int MAXVAL = 700; //Array size

      //Variables Declaration
      double x[] = new double[MAXVAL];
      double y[] = new double[MAXVAL];
      double z[] = new double[MAXVAL];
      String fileName1; //input file name
      int i = 0; //Array index
      int p, q;//row & col index in the data file
      int n;//number of data points

      StdIn in = new StdIn(); //creates a StdIn object (Chapman.io)

      //Get input file name
      System.out.print("Enter data file name under forte4j dir: ");
      fileName1 = in.readString();
      System.out.println(fileName1);

      //Open file (Chapman.io package)
      FileIn in1 = new FileIn(fileName1);
```

```
      //check for valid data file open
      if (in1.readStatus != in1.FILE_NOT_FOUND) {
        //Read numbers into array
        while (in1.readStatus != in1.EOF) {
          p = in1.readInt(); q = in1.readInt();
          x[i] = in1.readDouble();
          y[i] = in1.readDouble();
          z[i] = in1.readDouble();
          i += 1;
        }//end while
        n = i;
        in1.close();//Close data file

        LSquaresPlane ls = new LSquaresPlane(); //Create a LSquaresPlane object
        ls.calcPlane(x, y, z, n);//invoke method calcPlane

        //Write the results in the format specified
        System.out.println();
        Fmt.printf("The slope l0 = %9.6f\n" , ls.getSlopex());
        Fmt.printf("The slope m0 = %9.6f\n" , ls.getSlopey());
        Fmt.printf("The intercept z0 = %9.5f\n" , ls.getIntercept());

      } //end if loop
      else {//data file not found
        System.out.println("File " + fileName1 + " not found!");
      }
    } //end main method
} //end class LeastSquaresPlane
```

B.3 Region Elimination Search

```
/* This program finds solution point corresponding to emax(-ve direction) and
 * emax(+ve dir) using region elimination (RE) search method.
 * Arrays to hold (x,y) data points of the population.
 * File access to read data from & write to files
 */
import chapman.io.*;
public class RESearch {
  public static void main(String[] args) {
    final int MAXVAL = 150; //Array size
    final double err = 1e-5;
    final double XLL = 4.0 - err, XUL = 10.0 + err; //boundary points
    final double l0 = -0.000773, y0 = -14.82977; //LS line
```

```
//Variables Declaration
double x[] = new double[MAXVAL], y[] = new double[MAXVAL];
String infile = "dataStAlPlate12End.txt"; //data file name
String outfile = "outRE.txt"; //output file name
int i=0; //index
int N; //Number of data values
double delta = 0.05; //step size for population data file
double xnow, xopt, enow, emaxNeg;
double xl, yl, xu, yu, el, eu;
char reply;

//Instantiate objects from the classes in chapman.io package
StdIn in = new StdIn(); //To read from keyboard
FileIn in1 = new FileIn(infile);//Opens data file
FileOut out1 = new FileOut(outfile);//Opens output file

//check for valid data file open
if (in1.readStatus != in1.FILE_NOT_FOUND) {
   //Read points into arrays
   while (in1.readStatus != in1.EOF) {
      i = in1.readInt();
      x[i] = in1.readDouble();
      y[i] = in1.readDouble();
   } //end while
   N = i+1;
   in1.close();//Close data file
} //end if
else {
   System.out.println("File " + infile + " not found!");
}

//Finding a point with an optimum value of emax(-)
System.out.println("Search for a solution with emax(-)");

do {
   int p, k = 4; //k = 16, 8, 4 (depending on the step size to be used)
   System.out.println("Enter the index [p] of the start point: ");
   p = in.readInt();
   System.out.println("Enter the value of e: ");
   enow = in.readDouble();
   xnow = x[p];
   out1.printf("%21s\n", infile);
   out1.printf("%36s", "Starting point to find emax(-) was: ");
   out1.printf("x = %4.2f", xnow);
   out1.printf(" with e = %9.6f\n", enow);
```

```
for (int j = 0; j < 3; j++) {
   if ((xnow - k*delta) >= XLL) {//within the boundary
      xl = x[p-k]; yl = y[p-k];
      el = yl - (y0 + l0*xl);
   }//end if
   else {//outside the boundary
      xl = 100.0; el = 100.0; //a hypothetical value
   } //end else

   if ((xnow + k*delta) <= XUL) {//within boundary
      xu = x[p+k]; yu = y[p+k];
      eu = yu - (y0 + l0*xu);
   }//end if
   else {//outside the boundary
      xu = 100.0; eu = 100.0; //a hypothetical value
   } //end else

   //Compare e vlues
   if (el < enow) {
      if (el <= eu) {
      enow = el;
      xnow = xl; p = p - k;
      }
      else {
         enow = eu;
         xnow = xu; p = p + k;
      } //end inner if/else
   } // outer if
   else {
      if (eu < enow) {
         enow = eu;
         xnow = xu; p = p + k;
      }
   }//end outer if/else

   //Write in output file
   out1.printf("xl = %4.2f", xl);
   out1.printf(" el = %9.6f", el);
   out1.printf(", xu = %4.2f", xu);
   out1.printf(" eu = %9.6f", eu);
   out1.printf(", xnow = %4.2f", xnow);
   out1.printf(" enow = %9.6f\n", enow);

   k = k/2;
} //end for loop
```

```
    xopt = xnow;
    emaxNeg = enow;
    out1.printf("%23s\n", "Hence, the solution is:");
    out1.printf("xopt = %4.2f", xopt);
    out1.printf("   emax(-) = %9.6f\n", emaxNeg);

    System.out.println("Another starting point? Y/N");
    reply = in.readChar();

} while (reply == 'Y'); //end do/while loop

//Finding a point with an optimum value of emax(+)
System.out.println("Search for a solution with emax(+)");

do {
    int p, k = 4;
    System.out.println("Enter the index [p] of the start point: ");
    p = in.readInt();
    System.out.println("Enter the value of e: ");
    enow = in.readDouble();
    xnow = x[p];
    out1.printf("%30s\n", "------------------------------");
    out1.printf("%36s", "Starting point to find emax(+) was: ");
    out1.printf("x = %4.2f", xnow);
    out1.printf(" with e = %9.6f\n", enow);

    for (int j = 0; j < 3; j++) {
      if ((xnow - k*delta) >= XLL) {
        xl = x[p-k];
        yl = y[p-k];
        el = yl - (y0 + l0*xl);
      }
      else {//outside the boundary
        xl = 100.0; el = -100.0; //a hypothetical value
      } //end if/else

      if ((xnow + k*delta) <= XUL) {
        xu = x[p+k];
        yu = y[p+k];
        eu = yu - (y0 + l0*xu);
      }
      else {//outside the boundary
        xu = 100.0; eu = -100.0; //a hypothetical value
      } //end if/else

      //Compare e vlues
```

```
            if (el > enow) {
               if (el >= eu) {
               enow = el;
               xnow = xl; p = p - k;
               }
               else {
                  enow = eu;
                  xnow = xu; p = p + k;
               } //end inner if/else
            } // outer if
            else {
               if (eu > enow) {
                  enow = eu;
                  xnow = xu; p = p + k;
               }
            }//end outer if/else

            //Write in output file
            out1.printf("xl = %4.2f", xl);
            out1.printf(" el = %9.6f", el);
            out1.printf(", xu = %4.2f", xu);
            out1.printf(" eu = %9.6f", eu);
            out1.printf(", xnow = %4.2f", xnow);
            out1.printf(" enow = %9.6f\n", enow);

            k = k/2;
         } //end for loop

         xopt = xnow;
         emaxNeg = enow;
         out1.printf("%23s\n", "Hence, the solution is:");
         out1.printf("xopt = %4.2f", xopt);
         out1.printf("   emax(+) = %9.6f\n", emaxNeg);

         System.out.println("Another starting point? Y/N");
         reply = in.readChar();

      } while (reply == 'Y'); //end do/while loop

      out1.close();//Close output file

   } //end main method
} //end RESearch class
```

B.4 Tabu Search

```
/* This program finds solution points corresponding to emax(-ve direction) and
 * emax(+ve dir) using Tabu search method.
 * Arrays to hold (x,y,z) data points of the population.
 * File access to read data from & write to files
 */
import chapman.io.*;
public class TabuSearch {
   public static void main(String[] args) {
      final int MAXVAL = 31; //Array size
      final double err = 1.0e-5;
      final double XLL = 5. - err, XUL = 8. + err; //boundary in x-direction
      final double YLL = 5. - err, YUL = 7. + err; //boundary in y-direction
      final double l0 = -0.00187, m0 = 0.000193, z0 = -14.84665; //LS Plane

      //Variables Declaration
      double x[][] = new double[MAXVAL][MAXVAL];
      double y[][] = new double[MAXVAL][MAXVAL];
      double z[][] = new double[MAXVAL][MAXVAL];
      String infile = "dataFlAlPlate4.txt"; //input file name
      String outfile = "outTS.txt"; //output file name
      int p = 0, q = 0; //row & col index
      int pmin, pmax, qmin, qmax;//to track search area
      int iter, itermax = 35; /*iteration max includes intensification iter,
       which results no move bec' either bad or outside the boundary */
      int N; //Number of data values = # of row * # of col
      int badmove, badmovemax = 1;
      int direc[] = new int[itermax];/* direction of move in each iteration
        = 1 (+x), 2 (+y), 3 (-x), 4 (-y), 0 (no move) */
      double tabux, tabuy; // x & y value of tabu point
      double x1 = 0.0, y1 = 0.0, z1 = 0.0, x2 = 0.0, y2 = 0.0, z2 = 0.0;
      double x3 = 0.0, y3 = 0.0, z3 = 0.0, x4 = 0.0, y4 = 0.0, z4 = 0.0;
      double e1, e2, e3, e4;
      double delta = 0.1; //step size
      double xnow, ynow, znow, xopt = 0.0, yopt = 0.0, zopt = 0.0;
      double xtemp = 0.0, ytemp = 0.0, ztemp = 0.0, etemp = 0.0;
      double enow, ebest;
      char reply = 'N'; //Re-start for the search: Y/N

      //Instantiate objects from the classes in chapman.io package
      StdIn in = new StdIn(); //To read from keyboard
      FileIn in1 = new FileIn(infile);//Opens data file
      FileOut out1 = new FileOut(outfile);//Opens output file

      //check for valid data file open
```

```
    if (in1.readStatus != in1.FILE_NOT_FOUND) {
      //Read points into arrays
      while (in1.readStatus != in1.EOF) {
        p = in1.readInt();
        q = in1.readInt();
        x[p][q] = in1.readDouble();
        y[p][q] = in1.readDouble();
        z[p][q] = in1.readDouble();
      } //end while
      N = (p+1)*(q+1);
      in1.close();//Close data file
    } //end if
    else {
      System.out.println("File " + infile + " not found!");
    }

    //Write a note in the output file
    out1.printf("%5s\n", "Note:");
    out1.printf("%46s", "1. From iteration, subtract the iteration for ");
    out1.printf("%44s\n", "which intensification was not good/feasible.");
    out1.printf("%50s", "2. Iter starts w/ 0. If stopped bec' of badmovemax");
    out1.printf("%42s\n", ", the last iteration point is not visited.");
    out1.printf("%53s\n", "Either case, points visited = iter - failed intensif.");
    out1.printf("%55s\n", "Sampled points = visited in emax (- & +) + initial set.");

  //Finding a point with an optimum value of emax(-)
    System.out.println("Search for a solution for emax(-): ");
    //initial values
    ebest = 100.0; tabux = 100.0; tabuy = 100.0;
    pmin = 20; pmax = 0; qmin = 30; qmax = 0;

  do {
    badmove = 0;
    System.out.println("Enter row index [p] of the start point: ");
    p = in.readInt();
    System.out.println("Enter column index [q] of the start point: ");
    q = in.readInt();
    if((p <= pmax && p >= pmin) && (q <= qmax && q >= qmin)) {
     System.out.println("Enter a point outside p = " + pmin + " to " + pmax
       + " & q = " + qmin + " to " + qmax);
     continue;//do loop
    }
    System.out.println("Enter e value of the start point: ");
      enow = in.readDouble();
      xnow = x[p][q]; ynow = y[p][q]; znow = z[p][q];
      out1.printf("%2s\n", "  ");
```

```
    out1.printf("%35s\n", "Starting point to find emax(-) was ");
    out1.printf("x = %4.1f", xnow);
    out1.printf(", y = %4.1f", ynow);
    out1.printf(", z = %9.5f", znow);
    out1.printf(" with e = %9.6f\n", enow);
    if (p < pmin) pmin = p;
    if (p > pmax) pmax = p;
    if (q < qmin) qmin = q;
    if (q > qmax) qmax = q;
    if (enow < ebest) {
      ebest = enow; xopt = xnow;
      yopt = ynow; zopt = znow;
    }

  for (iter = 0; iter < itermax; iter++) {
    if (iter <= 1 || (direc[iter - 1] != direc[iter - 2])) {

      //Neighborhood search
      if ((xnow + delta) <= XUL) {
        y1 = ynow; x1 = x[p][q+1];
        if ((Math.abs(x1 - tabux) <= 0.1e-5) && (Math.abs(y1 - tabuy) <= 0.1e-5))
          e1 = 100.0; //tabu
        else {
          z1 = z[p][q+1];
          e1 = z1 - (z0 + l0*x1 + m0*y1);
        }
      }
      else e1 = 100.0;//outside the boundary

      if ((ynow + delta) <= YUL) {
        x2 = xnow; y2 = y[p+1][q];
        if ((Math.abs(x2 - tabux) <= 0.1e-5) && (Math.abs(y2 - tabuy) <= 0.1e-5))
          e2 = 100.0; //tabu
        else {
          z2 = z[p+1][q];
          e2 = z2 - (z0 + l0*x2 + m0*y2);
        }
      }
      else e2 = 100.0;//outside the boundary

      if ((xnow - delta) >= XLL) {
        y3 = ynow; x3 = x[p][q-1];
        if ((Math.abs(x3 - tabux) <= 0.1e-5) && (Math.abs(y3 - tabuy) <= 0.1e-5))
          e3 = 100.0; //tabu
        else {
          z3 = z[p][q-1];
```

```
      e3 = z3 - (z0 + l0*x3 + m0*y3);
    }
  }
  else e3 = 100.0;//outside the boundary

  if ((ynow - delta) >= YLL) {
    x4 = xnow; y4 = y[p-1][q];
    if ((Math.abs(x4 - tabux) <= 0.1e-5) && (Math.abs(y4 - tabuy) <= 0.1e-5))
      e4 = 100.0; //tabu
    else {
      z4 = z[p-1][q];
      e4 = z4 - (z0 + l0*x4 + m0*y4);
    }
  }
  else e4 = 100.0;//outside the boundary

  if (e1 <= e2){
    if (e1 <= e3) {
      if (e1 <= e4) {
        etemp = e1; xtemp = x1;
        ytemp = y1; ztemp = z1;
        q = q + 1; direc[iter] = 1;
      }//end if e1 <= e4
      else {
        etemp = e4; xtemp = x4;
        ytemp = y4; ztemp = z4;
        p = p - 1; direc[iter] = 4;
      }//end else
    }//end if e1 <= e3
    else {
      if (e3 <= e4) {
        etemp = e3; xtemp = x3;
        ytemp = y3; ztemp = z3;
        q = q - 1; direc[iter] = 3;
      }//end if e3 <= e4
      else {
        etemp = e4; xtemp = x4;
        ytemp = y4; ztemp = z4;
        p = p - 1; direc[iter] = 4;
      }//end else
    }//end else e1 > e3
  }//end if e1 <= e2
  else {
    if (e2 <= e3) {
      if (e2 <= e4) {
        etemp = e2; xtemp = x2;
```

```
            ytemp = y2; ztemp = z2;
            p = p + 1; direc[iter] = 2;
          }//end if e2 <= e4
          else {
            etemp = e4; xtemp = x4;
            ytemp = y4; ztemp = z4;
            p = p - 1; direc[iter] = 4;
          }//end else
        }//end if e2 <= e3
        else {
          if (e3 <= e4) {
            etemp = e3; xtemp = x3;
            ytemp = y3; ztemp = z3;
            q = q - 1; direc[iter] = 3;
          }//end if e3 <= e4
          else {
            etemp = e4; xtemp = x4;
            ytemp = y4; ztemp = z4;
            p = p - 1; direc[iter] = 4;
          }//end else e3 > e4
        }//end else e2 > e3
      }//end else e1 > e2

      if (etemp > enow) { //bad move
        badmove = badmove + 1;
        if (badmove > badmovemax){
          out1.printf("%32s\n", "Bad moves exceeded the max limit");
          break;
        }//exit from the for loop
      }
    } //end if iter <= 1 or direc[iter-1] != direc[iter-2]

    else { // No need to check for tabu as moving ahead in the direction
        //  of the 2 prev moves (intensification)
      if (direc[iter-1] == 1) {
        if ((xnow + delta) <= XUL) {
          ytemp = ynow;
          xtemp = x[p][q+1];
          ztemp = z[p][q+1];
          etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
          if (etemp < enow) {//good move
            direc[iter] = 1;
            q = q + 1;
          }
          else {
            direc[iter] = 0;
```

```
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": Intensification not good!");
            continue; //for loop
         }
      }
      else {//outside the boundary
         direc[iter] = 0;
         out1.printf("Iteration %2i", iter);
         out1.printf("%30s\n", ": Intensification not feasible!");
         continue; //for loop
      }
   }
   else if (direc[iter-1] == 2) {
      if ((ynow + delta) <= YUL) {
         xtemp = xnow;
         ytemp = y[p+1][q];
         ztemp = z[p+1][q];
         etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp < enow) {//good move
            direc[iter] = 2;
            p = p + 1;
         }
         else {
            direc[iter] = 0;
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": Intensification not good!");
            continue; //for loop
         }
      }
      else {//outside the boundary
         direc[iter] = 0;
         out1.printf("Iteration %2i", iter);
         out1.printf("%30s\n", ": Intensification not feasible!");
         continue; //for loop
      }
   }
   else if (direc[iter-1] == 3) {
      if ((xnow - delta) >= XLL) {
         ytemp = ynow;
         xtemp = x[p][q-1];
         ztemp = z[p][q-1];
         etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
         if (etemp < enow) {//good move
            direc[iter] = 3;
            q = q - 1;
         }
```

```
            else {
               direc[iter] = 0;
               out1.printf("Iteration %2i", iter);
               out1.printf("%30s\n", ": Intensification not good!");
               continue; //for loop
            }
         }
         else {//outside the boundary
            direc[iter] = 0;
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": Intensification not feasible!");
            continue; //for loop
         }
      }
      else if (direc[iter-1] == 4) {
         if ((ynow - delta) >= YLL) {
            xtemp = xnow;
            ytemp = y[p-1][q];
            ztemp = z[p-1][q];
            etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
            if (etemp < enow) {//good move
               direc[iter] = 4;
               p = p - 1;
            }
            else {
               direc[iter] = 0;
               out1.printf("Iteration %2i", iter);
               out1.printf("%30s\n", ": Intensification not good!");
               continue; //for loop
            }
         }
         else {//outside the boundary
            direc[iter] = 0;
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": Intensification not feasible!");
            continue; //for loop
         }
      }
   }//end else, i.e. direc[iter-1] == direc[iter-2]

   tabux = xnow; tabuy = ynow;
   enow = etemp; xnow = xtemp;
   ynow = ytemp; znow = ztemp;
   if (p < pmin) pmin = p;
   if (p > pmax) pmax = p;
   if (q < qmin) qmin = q;
```

```
        if (q > qmax) qmax = q;
        if (enow < ebest) {
           ebest = enow; xopt = xnow;
           yopt = ynow; zopt = znow;
        }

     } //end for loop

     //Write in output file
        out1.printf("%35s", "Hence, the solution obtained after ");
        out1.printf("iteration %2i", iter);
        out1.printf("%1s\n", ":");
        out1.printf("x = %4.1f", xopt);
        out1.printf(", y = %4.1f", yopt);
        out1.printf(", z = %9.5f", zopt);
        out1.printf(" with emax(-) = %9.6f\n", ebest);
        out1.printf("%36s\n", "Plate is divided in 21 row x 31 col.");
        out1.printf("%15s", "Area searched: ");
        out1.printf(" row = %2i", pmin+1);
        out1.printf(" to row = %2i", pmax+1);
        out1.printf(" and col = %2i", qmin+1);
        out1.printf(" to col = %2i\n", qmax+1);

     System.out.println("Another starting point for emax(-)? Y/N");
     reply = in.readChar();
  } while (reply == 'Y'); //end do/while loop

  //Finding a point with an optimum value of emax(+)
     System.out.println("Search for a solution with emax(+)");
     ebest = -100.0; tabux = 100.0; tabuy = 100.0;
     pmin = 20; pmax = 0; qmin = 30; qmax = 0;

  do {
     badmove = 0;
     System.out.println("Enter row index [p] of the start point: ");
     p = in.readInt();
     System.out.println("Enter column index [q] of the start point: ");
     q = in.readInt();
     if((p <= pmax && p >= pmin) && (q <= qmax && q >= qmin)) {
      System.out.println("Enter a point outside p = " + pmin + " to " + pmax
        + " & q = " + qmin + " to " + qmax);
      continue;//do loop
     }
     System.out.println("Enter e value of the start point: ");
        enow = in.readDouble();
        xnow = x[p][q];
```

```
    ynow = y[p][q];
    znow = z[p][q];
    out1.printf("%30s\n", "------------------------------");
    out1.printf("%35s\n", "Starting point to find emax(+) was ");
    out1.printf("x = %4.1f", xnow);
    out1.printf(", y = %4.1f", ynow);
    out1.printf(", z = %9.5f", znow);
    out1.printf(" with e = %9.6f\n", enow);
    if (p < pmin) pmin = p;
    if (p > pmax) pmax = p;
    if (q < qmin) qmin = q;
    if (q > qmax) qmax = q;
    if (enow > ebest) {
       ebest = enow; xopt = xnow;
       yopt = ynow; zopt = znow;
    }

for (iter = 0; iter < itermax; iter++) {
   if (iter <= 1 || (direc[iter - 1] != direc[iter - 2])) {

      //Neighborhood search
      if ((xnow + delta) <= XUL) {
         y1 = ynow; x1 = x[p][q+1];
         if ((Math.abs(x1 - tabux) <= 0.1e-5) && (Math.abs(y1 - tabuy) <= 0.1e-5))
            e1 = -100.0; //tabu
         else {
            z1 = z[p][q+1];
            e1 = z1 - (z0 + l0*x1 + m0*y1);
         }
      }
      else e1 = -100.0;//outside the boundary

      if ((ynow + delta) <= YUL) {
         x2 = xnow; y2 = y[p+1][q];
         if ((Math.abs(x2 - tabux) <= 0.1e-5) && (Math.abs(y2 - tabuy) <= 0.1e-5))
            e2 = -100.0; //tabu
         else {
            z2 = z[p+1][q];
            e2 = z2 - (z0 + l0*x2 + m0*y2);
         }
      }
      else e2 = -100.0;//outside the boundary

      if ((xnow - delta) >= XLL) {
         y3 = ynow; x3 = x[p][q-1];
         if ((Math.abs(x3 - tabux) <= 0.1e-5) && (Math.abs(y3 - tabuy) <= 0.1e-5))
```

```
        e3 = -100.0; //tabu
      else {
        z3 = z[p][q-1];
        e3 = z3 - (z0 + l0*x3 + m0*y3);
      }
   }
   else e3 = -100.0;//outside the boundary

   if ((ynow - delta) >= YLL) {
      x4 = xnow; y4 = y[p-1][q];
      if ((Math.abs(x4 - tabux) <= 0.1e-5) && (Math.abs(y4 - tabuy) <= 0.1e-5))
        e4 = -100.0; //tabu
      else {
        z4 = z[p-1][q];
        e4 = z4 - (z0 + l0*x4 + m0*y4);
      }
   }
   else e4 = -100.0;//outside the boundary

   if (e1 >= e2){
      if (e1 >= e3) {
        if (e1 >= e4) {
          etemp = e1; xtemp = x1;
          ytemp = y1; ztemp = z1;
          q = q + 1; direc[iter] = 1;
        }//end if e1 >= e4
        else {
          etemp = e4; xtemp = x4;
          ytemp = y4; ztemp = z4;
          p = p - 1; direc[iter] = 4;
        }//end else e1 < e4
      }//end if e1 >= e3
      else {
        if (e3 >= e4) {
          etemp = e3; xtemp = x3;
          ytemp = y3; ztemp = z3;
          q = q - 1; direc[iter] = 3;
        }//end if e3 >= e4
        else {
          etemp = e4; xtemp = x4;
          ytemp = y4; ztemp = z4;
          p = p - 1; direc[iter] = 4;
        }//end else e3 < e4
      }//end else e1 < e3
   }//end if e1 >= e2
   else {//e1 < e2
```

```
      if (e2 >= e3) {
        if (e2 >= e4) {
          etemp = e2; xtemp = x2;
          ytemp = y2; ztemp = z2;
          p = p + 1; direc[iter] = 2;
        }//end if e2 >= e4
        else {
          etemp = e4; xtemp = x4;
          ytemp = y4; ztemp = z4;
          p = p - 1; direc[iter] = 4;
        }//end else e2 < e4
      }//end if e2 >= e3
      else {
        if (e3 >= e4) {
          etemp = e3; xtemp = x3;
          ytemp = y3; ztemp = z3;
          q = q - 1; direc[iter] = 3;
        }//end if e3 >= e4
        else {
          etemp = e4; xtemp = x4;
          ytemp = y4; ztemp = z4;
          p = p - 1; direc[iter] = 4;
        }//end else e3 < e4
      }//end else e2 < e3
    }//end else e1 < e2

    if (etemp < enow) { //bad move
      badmove = badmove + 1;
      if (badmove > badmovemax){
        out1.printf("%32s\n", "Bad moves exceeded the max limit");
        break;
      }//exit from the for loop
    }
  } //end if iter <= 1 or direc[iter-1] != direc[iter-2]

  else { // No need to check for tabu as moving ahead in the direction
      // of the 2 prev moves (intensification)
    if (direc[iter-1] == 1) {
      if ((xnow + delta) <= XUL) {
        ytemp = ynow;
        xtemp = x[p][q+1];
        ztemp = z[p][q+1];
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp > enow) {//good move
          direc[iter] = 1;
          q = q + 1;
```

```
        }
        else {
          direc[iter] = 0;
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": Intensification not good!");
          continue; //for loop
        }
      }
      else {//outside the boundary
        direc[iter] = 0;
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": Intensification not feasible!");
        continue; //for loop
      }
    }
    else if (direc[iter-1] == 2) {
      if ((ynow + delta) <= YUL) {
        xtemp = xnow;
        ytemp = y[p+1][q];
        ztemp = z[p+1][q];
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
       if (etemp > enow) {//good move
          direc[iter] = 2;
          p = p + 1;
        }
        else {
          direc[iter] = 0;
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": Intensification not good!");
          continue; //for loop
        }
      }
      else {//outside the boundary
        direc[iter] = 0;
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": Intensification not feasible!");
        continue; //for loop
      }
    }
    else if (direc[iter-1] == 3) {
      if ((xnow - delta) >= XLL) {
        ytemp = ynow;
        xtemp = x[p][q-1];
        ztemp = z[p][q-1];
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp > enow) {//good move
```

```
            direc[iter] = 3;
            q = q - 1;
          }
          else {
            direc[iter] = 0;
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": Intensification not good!");
            continue; //for loop
          }
        }
        else {//outside the boundary
          direc[iter] = 0;
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": Intensification not feasible!");
          continue; //for loop
        }
      }
      else if (direc[iter-1] == 4) {
        if ((ynow - delta) >= YLL) {
          xtemp = xnow;
          ytemp = y[p-1][q];
          ztemp = z[p-1][q];
          etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
          if (etemp > enow) {//good move
            direc[iter] = 4;
            p = p - 1;
          }
          else {
            direc[iter] = 0;
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": Intensification not good!");
            continue; //for loop
          }
        }
        else {//outside the boundary
          direc[iter] = 0;
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": Intensification not feasible!");
          continue; //for loop
        }
      }
    }//end else, i.e. direc[iter-1] == direc[iter-2]

    tabux = xnow; tabuy = ynow;
    enow = etemp; xnow = xtemp;
    ynow = ytemp; znow = ztemp;
```

```
            if (p < pmin) pmin = p;
            if (p > pmax) pmax = p;
            if (q < qmin) qmin = q;
            if (q > qmax) qmax = q;
            if (enow > ebest) {
               ebest = enow; xopt = xnow;
               yopt = ynow; zopt = znow;
            }

         } //end for loop

         //Write in output file
            out1.printf("%35s", "Hence, the solution obtained after ");
            out1.printf("iteration %2i", iter);
            out1.printf("%1s\n", ":");
            out1.printf("x = %4.1f", xopt);
            out1.printf(", y = %4.1f", yopt);
            out1.printf(", z = %9.5f", zopt);
            out1.printf(" with emax(+) = %9.6f\n", ebest);
            out1.printf("%36s\n", "Plate is divided in 21 row x 31 col.");
            out1.printf("%15s", "Area searched: ");
            out1.printf(" row = %2i", pmin+1);
            out1.printf(" to row = %2i", pmax+1);
            out1.printf(" and col = %2i", qmin+1);
            out1.printf(" to col = %2i\n", qmax+1);

         System.out.println("Another starting point for emax(+)? Y/N");
         reply = in.readChar();
      } while (reply == 'Y'); //end do/while loop

         out1.close();//Close output file

   } //end main method
} //end TabuSearch class
```

B.5 Hybrid Search

```
/* This program finds solution points corresponding to emax(-ve direction) and
 * emax(+ve dir) using Hybrid search method.
 * Arrays to hold (x,y,z) data points of the population.
 * File access to read data from & write to files
*/
import chapman.io.*;
public class HybridSearch {
   public static void main(String[] args) {
```

```
final int MAXVAL = 31; //Array size
final double err = 1.0e-5;
final double XLL = 5. - err, XUL = 8. + err; //boundary in x-direction
final double YLL = 5. - err, YUL = 7. + err; //boundary in y-direction
final double l0 = -0.00187, m0 = 0.000193, z0 = -14.84665; //LS Plane

//Variables Declaration
double x[][] = new double[MAXVAL][MAXVAL];
double y[][] = new double[MAXVAL][MAXVAL];
double z[][] = new double[MAXVAL][MAXVAL];
String infile = "dataFlAlPlate4.txt"; //data file is same as in Tabu search.
String outfile = "outHS.txt"; //output file name
int p = 0, q = 0; //row & col index
int pmin, pmax, qmin, qmax;//to track search area
int iter, itermax = 35; /*iteration max includes intensification iter,
 which results no move bec' either bad or outside the boundary */
int N; //Number of data values = # of row * # of col
int badmove, badmovemax = 1;
int direc[] = new int[itermax+1];/* direction of move in each iteration
= 1 for (1,0), 2 (1,1), 3 (0,1), 4 (-1,1), 5 (-1,0), 6 (-1,-1),
7(0,-1), or 8 (1,-1). The elements are direc[0] to direc[itermax].
Iter starts w/ 1 in 'for' loop, so we need to define direc[0] to avoid
out of bound error. But direc[0] is not used in the program */
direc[0] = 0;
double tabux, tabuy; // x & y value of tabu point
double x1 = 0.0, y1 = 0.0, z1 = 0.0, x2 = 0.0, y2 = 0.0, z2 = 0.0;
double x3 = 0.0, y3 = 0.0, z3 = 0.0, x4 = 0.0, y4 = 0.0, z4 = 0.0;
double x5 = 0.0, y5 = 0.0, z5 = 0.0, x6 = 0.0, y6 = 0.0, z6 = 0.0;
double x7 = 0.0, y7 = 0.0, z7 = 0.0, x8 = 0.0, y8 = 0.0, z8 = 0.0;
double e1, e2, e3, e4, e5, e6, e7, e8, ebest;
double delta = 0.1; //step size
double e[] = new double[itermax+1];// enow of each iteration,
   //for starting point it's e[0] and then e[1] to e[itermax]
double xnow, ynow, znow, xopt = 0.0, yopt = 0.0, zopt = 0.0;
double xtemp = 0.0, ytemp = 0.0, ztemp = 0.0, etemp = 0.0;
char reply = 'N'; //Re-start for the search: Y/N

//Instantiate objects from the classes in chapman.io package
StdIn in = new StdIn(); //To read from keyboard
FileIn in1 = new FileIn(infile);//Opens data file
FileOut out1 = new FileOut(outfile);//Opens output file

//check for valid data file open
if (in1.readStatus != in1.FILE_NOT_FOUND) {
   //Read points into arrays
   while (in1.readStatus != in1.EOF) {
```

```
            p = in1.readInt();
            q = in1.readInt();
            x[p][q] = in1.readDouble();
            y[p][q] = in1.readDouble();
            z[p][q] = in1.readDouble();
          } //end while
          N = (p+1)*(q+1);
          in1.close();//Close data file
        } //end if
        else {
          System.out.println("File " + infile + " not found!");
        }

        //Write a note in the output file
        out1.printf("%5s\n", "Note: ");
        out1.printf("%46s", "1. From iteration, subtract the iteration for ");
        out1.printf("%41s\n", "which pattern move was not good/feasible.");
        out1.printf("%47s", "2. Iter starts w/ 1; if stopped bec' of itermax");
        out1.printf("%13s\n", ", subtract 1.");
        out1.printf("%34s", "3. If stopped bec' of badmovemax, ");
        out1.printf("%41s\n", "the last iteration point is not visited.");
        out1.printf("%51s\n", "Both case, points visited = (iter-1) - failed iter.");
        out1.printf("%55s\n", "Sampled points = visited in emax (- & +) + initial set.");

    //Finding a point with an optimum value of emax(-)
        System.out.println("Search for a solution for emax(-): ");
        //initial values
        ebest = 100.0; tabux = 100.0; tabuy = 100.0;
        pmin = 20; pmax = 0; qmin = 30; qmax = 0;

    do {
        badmove = 0;
        System.out.println("Enter row index [p] of the start point: ");
        p = in.readInt();
        System.out.println("Enter column index [q] of the start point: ");
        q = in.readInt();
        if((p <= pmax && p >= pmin) && (q <= qmax && q >= qmin)) {
         System.out.println("Enter a point outside p = " + pmin + " to " + pmax
           + " & q = " + qmin + " to " + qmax);
         continue;//do loop
        }
        System.out.println("Enter e value of the start point: ");
          e[0] = in.readDouble();
          xnow = x[p][q];
          ynow = y[p][q];
          znow = z[p][q];
```

```
    out1.printf("%2s\n", "  ");
    out1.printf("%35s\n", "Starting point to find emax(-) was ");
    out1.printf("x = %4.1f", xnow);
    out1.printf(", y = %4.1f", ynow);
    out1.printf(", z = %9.5f", znow);
    out1.printf(" with e = %9.6f\n", e[0]);
    if (p < pmin) pmin = p;
    if (p > pmax) pmax = p;
    if (q < qmin) qmin = q;
    if (q > qmax) qmax = q;
    if (e[0] < ebest) {
       ebest = e[0]; xopt = xnow;
       yopt = ynow; zopt = znow;
    }

for (iter = 1; iter <= itermax; iter++) {
   if (iter < 2 || (e[iter - 1] >= e[iter - 2])) {

     //Coordinate search exploratory move
     if ((xnow + delta) <= XUL) {//within boundary
        x1 = x[p][q+1]; y1 = ynow;
        if ((Math.abs(x1 - tabux) <= 0.1e-5) && (Math.abs(y1 - tabuy) <= 0.1e-5))
            e1 = 100.0; //tabu
        else {
            z1 = z[p][q+1];
            e1 = z1 - (z0 + l0*x1 + m0*y1);
        }//end else
     }//end if w/in boundary
     else e1 = 100.0;//outside the boundary

     if (e1 < e[iter-1]) {//good
        if ((y1 + delta) <= YUL) {//within boundary
          x2 = x1; y2 = y[p+1][q+1];
          if ((Math.abs(x2 - tabux) <= 0.1e-5) && (Math.abs(y2 - tabuy) <= 0.1e-5))
            e2 = 100.0; //tabu
          else {
            z2 = z[p+1][q+1];
            e2 = z2 - (z0 + l0*x2 + m0*y2);
          }//end else
        }//end if w/in boundary
        else e2 = 100.0;//outside the boundary

        if (e2 < e1) {
          etemp = e2; xtemp = x2; ytemp = y2; ztemp = z2;
          p = p + 1; q = q + 1; direc[iter] = 2;
        }
```

```
    else {//e2 >= e1
     if ((y1 - delta) >= YLL) {//within boundary
      x8 = x1; y8 = y[p-1][q+1];
      if ((Math.abs(x8 - tabux) <= 0.1e-5) && (Math.abs(y8 - tabuy) <= 0.1e-5))
       e8 = 100.0; //tabu
      else {
       z8 = z[p-1][q+1];
       e8 = z8 - (z0 + l0*x8 + m0*y8);
      }//end else
     }//end if w/in boundary
     else e8 = 100.0;//outside the boundary

     if (e8 < e1) {
      etemp = e8; xtemp = x8; ytemp = y8; ztemp = z8;
      p = p - 1; q = q + 1; direc[iter] = 8;
     }
     else {//e8 >= e1 (already e2 >= e1)
      etemp = e1; xtemp = x1; ytemp = y1; ztemp = z1;
      q = q + 1; direc[iter] = 1; //p not changed
     }//end else e8 >= e1
    }//end else e2 >= e1
}//end if e1 < e[iter-1]
else {//e1 >= e[iter-1]
 if ((xnow - delta) >= XLL) {//within boundary
    x5 = x[p][q-1]; y5 = ynow;
    if ((Math.abs(x5 - tabux) <= 0.1e-5) && (Math.abs(y5 - tabuy) <= 0.1e-5))
       e5 = 100.0; //tabu
    else {
       z5 = z[p][q-1];
       e5 = z5 - (z0 + l0*x5 + m0*y5);
    }//end else
 }//end if w/in boundary
 else e5 = 100.0;//outside the boundary

 if (e5 < e[iter-1]) {//good
   if ((y5 + delta) <= YUL) {//within boundary
    x4 = x5; y4 = y[p+1][q-1];
    if ((Math.abs(x4 - tabux) <= 0.1e-5) && (Math.abs(y4 - tabuy) <= 0.1e-5))
       e4 = 100.0; //tabu
    else {
       z4 = z[p+1][q-1];
       e4 = z4 - (z0 + l0*x4 + m0*y4);
    }//end else
   }//end if within boundary
   else e4 = 100.0;//outside the boundary
```

```
    if (e4 < e5) {
     etemp = e4; xtemp = x4; ytemp = y4; ztemp = z4;
     p = p + 1; q = q - 1; direc[iter] = 4;
    }
    else {//e4 >= e5
     if ((y5 - delta) >= YLL) {//within boundary
      x6 = x5; y6 = y[p-1][q-1];
      if ((Math.abs(x6 - tabux) <= 0.1e-5) && (Math.abs(y6 - tabuy) <= 0.1e-5))
          e6 = 100.0; //tabu
      else {
       z6 = z[p-1][q-1];
       e6 = z6 - (z0 + l0*x6 + m0*y6);
      }//end else
     }//end if within boundary
     else e6 = 100.0;//outside the boundary

     if (e6 < e5) {
       etemp = e6; xtemp = x6; ytemp = y6; ztemp = z6;
       p = p - 1; q = q - 1; direc[iter] = 6;
     }
     else {//e6 >= e5, already e4 >= e5
       etemp = e5; xtemp = x5; ytemp = y5; ztemp = z5;
       q = q - 1; direc[iter] = 5;
     }//end else e6 >= e5
    }//end else e4 >= e5
  }//end if e5 < e[iter-1]
  else {//e5 >= e[iter-1]
   if ((ynow + delta) <= YUL) {//within boundary
    x3 = xnow;
    y3 = y[p+1][q];
    if ((Math.abs(x3 - tabux) <= 0.1e-5) && (Math.abs(y3 - tabuy) <= 0.1e-5))
       e3 = 100.0; //tabu
    else {
       z3 = z[p+1][q];
       e3 = z3 - (z0 + l0*x3 + m0*y3);
    }//end else
   }//end if within boundary
   else e3 = 100.0;//outside the boundary

   if (e3 < e[iter-1]) {//good
    etemp = e3; xtemp = x3; ytemp = y3; ztemp = z3;
    p = p + 1; direc[iter] = 3;
   }
   else {//e3 >= e[iter-1]
    if ((ynow - delta) >= YLL) {//within boundary
     x7 = xnow;
```

```
  y7 = y[p-1][q];
  if ((Math.abs(x7 - tabux) <= 0.1e-5) && (Math.abs(y7 - tabuy) <= 0.1e-5))
    e7 = 100.0; //tabu
  else {
    z7 = z[p-1][q];
    e7 = z7 - (z0 + l0*x7 + m0*y7);
  }//end else
 }//end if within boundary
 else e7 = 100.0;//outside the boundary

 if (e7 < e[iter-1]) {//good
    etemp = e7; xtemp = x7; ytemp = y7; ztemp = z7;
    p = p - 1; direc[iter] = 7;
 }
 else {// e7 >= e[iter-1], e1, e5, e3 are already bad. If e1 is
 // bad, e2 & e8 are not computed. If e5 is bad, e4 & e6 are not
 // computed. So, choose the least bad from e1, e3, e5, e7.

 if (e1 <= e3){
  if (e1 <= e5) {
    if (e1 <= e7) {
       etemp = e1; xtemp = x1;
       ytemp = y1; ztemp = z1;
       q = q + 1; direc[iter] = 1;
    }//end if e1 <= e7
    else {//e1 > e7
        etemp = e7; xtemp = x7;
        ytemp = y7; ztemp = z7;
        p = p - 1; direc[iter] = 7;
    }//end else e1 > e7
  }//end if e1 <= e5
  else {// e1 > e5
    if (e5 <= e7) {
       etemp = e5; xtemp = x5;
       ytemp = y5; ztemp = z5;
       q = q - 1; direc[iter] = 5;
    }//end if e5 <= e7
    else {//e5 > e7
        etemp = e7; xtemp = x7;
        ytemp = y7; ztemp = z7;
        p = p - 1; direc[iter] = 7;
    }//end else e5 > e7
  }//end else e1 > e5
 }//end if e1 <= e3
 else {//e1 > e3
  if (e3 <= e5) {
```

```
          if (e3 <= e7) {
            etemp = e3; xtemp = x3;
            ytemp = y3; ztemp = z3;
            p = p + 1; direc[iter] = 3;
          }//end if e3 <= e7
          else {//e3 > e7
             etemp = e7; xtemp = x7;
             ytemp = y7; ztemp = z7;
             p = p - 1; direc[iter] = 7;
          }//end else e3 > e7
        }//end if e3 <= e5
        else { // e3 > e5
          if (e5 <= e7) {
            etemp = e5; xtemp = x5;
            ytemp = y5; ztemp = z5;
            q = q - 1; direc[iter] = 5;
          }//end if e5 <= e7
          else {//e5 > e7
             etemp = e7; xtemp = x7;
             ytemp = y7; ztemp = z7;
             p = p - 1; direc[iter] = 7;
          }//end else e5 > e7
        }//end else e3 > e5
      }//end else e1 > e3
      }//end else e7 >= e[iter-1]
    }//end else e3 >= e[iter-1]
   }//end else e5 >= e[iter-1]
  }//end else e1 >= e[iter-1]

  if (etemp > e[iter-1]) { //bad move
     badmove = badmove + 1;
     if (badmove > badmovemax){
        out1.printf("%32s\n", "Bad moves exceeded the max limit");
        break;
     }//exit from the for loop
  }

}//end if iter < 2  or e[iter-1] >= e[iter-2]
else { // iter >=2 and e[iter-1] < e[iter-2]
 //Check for HJ pattern move in the direction of the prev move.
 //No need to check for tabu as it is not returning to prev point.
   if (direc[iter-1] == 1) {
      if ((xnow + delta) <= XUL) {//within boundary
         ytemp = ynow;
         xtemp = x[p][q+1];
         ztemp = z[p][q+1];
```

```
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp < e[iter-1]) {//good move
          direc[iter] = 1;
          q = q + 1;
        }
        else {
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not good!");
          continue; //for loop
        }
      }//end if w/in boundary
      else {//outside the boundary
        e[iter] = e[iter-1];
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": pattern move not feasible!");
        continue; //for loop
      }
    }
    else if (direc[iter-1] == 2) {
      if ((xnow + delta <= XUL) && (ynow + delta <= YUL)) {
        xtemp = x[p+1][q+1];
        ytemp = y[p+1][q+1];
        ztemp = z[p+1][q+1];
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp < e[iter-1]) {//good move
          direc[iter] = 2;
          p = p + 1; q = q + 1;
        }
        else {
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not good!");
          continue; //for loop
        }
      }//end if w/in boundary
      else {//outside the boundary
        e[iter] = e[iter-1];
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": pattern move not feasible!");
        continue; //for loop
      }
    }
    else if (direc[iter-1] == 3) {
      if ((ynow + delta) <= YUL) {//within boundary
        xtemp = xnow;
```

```
        ytemp = y[p+1][q];
        ztemp = z[p+1][q];
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp < e[iter-1]) {//good move
          direc[iter] = 3;
          p = p + 1;
        }
        else {
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not good!");
          continue; //for loop
        }
      }//end if w/in boundary
      else {//outside the boundary
        e[iter] = e[iter-1];
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": pattern move not feasible!");
        continue; //for loop
      }
    }
    else if (direc[iter-1] == 4) {
      if ((xnow - delta >= XLL) && (ynow + delta <= YUL)) {
        xtemp = x[p+1][q-1];
        ytemp = y[p+1][q-1];
        ztemp = z[p+1][q-1];
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp < e[iter-1]) {//good move
          direc[iter] = 4;
          p = p + 1; q = q - 1;
        }
        else {
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not good!");
          continue; //for loop
        }
      }//end if w/in boundary
      else {//outside the boundary
        e[iter] = e[iter-1];
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": pattern move not feasible!");
        continue; //for loop
      }
    }
    else if (direc[iter-1] == 5) {
```

```
    if ((xnow - delta) >= XLL) {//within boundary
      ytemp = ynow;
      xtemp = x[p][q-1];
      ztemp = z[p][q-1];
      etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
      if (etemp < e[iter-1]) {//good move
        direc[iter] = 5;
        q = q - 1;
      }
      else {
        e[iter] = e[iter-1];
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": pattern move not good!");
        continue; //for loop
      }
    }//end if w/in boundary
    else {//outside the boundary
      e[iter] = e[iter-1];
      out1.printf("Iteration %2i", iter);
      out1.printf("%30s\n", ": pattern move not feasible!");
      continue; //for loop
    }
  }
  else if (direc[iter-1] == 6) {
    if ((xnow - delta >= XLL) && (ynow - delta >= YLL)) {
      xtemp = x[p-1][q-1];
      ytemp = y[p-1][q-1];
      ztemp = z[p-1][q-1];
      etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
      if (etemp < e[iter-1]) {//good move
        direc[iter] = 6;
        p = p - 1; q = q - 1;
      }
      else {
        e[iter] = e[iter-1];
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": pattern move not good!");
        continue; //for loop
      }
    }//end if w/in boundary
    else {//outside the boundary
      e[iter] = e[iter-1];
      out1.printf("Iteration %2i", iter);
      out1.printf("%30s\n", ": pattern move not feasible!");
      continue; //for loop
    }
```

```
}
else if (direc[iter-1] == 7) {
   if ((ynow - delta) >= YLL) {//within boundary
      xtemp = xnow;
      ytemp = y[p-1][q];
      ztemp = z[p-1][q];
      etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
      if (etemp < e[iter-1]) {//good move
         direc[iter] = 7;
         p = p - 1;
      }
      else {
         e[iter] = e[iter-1];
         out1.printf("Iteration %2i", iter);
         out1.printf("%30s\n", ": pattern move not good!");
         continue; //for loop
      }
   }//end if w/in boundary
   else {//outside the boundary
      e[iter] = e[iter-1];
      out1.printf("Iteration %2i", iter);
      out1.printf("%30s\n", ": pattern move not feasible!");
      continue; //for loop
   }
}
else {// direc[iter-1] == 8
   if ((xnow + delta <= XUL) && (ynow - delta >= YLL)) {
      xtemp = x[p-1][q+1];
      ytemp = y[p-1][q+1];
      ztemp = z[p-1][q+1];
      etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
      if (etemp < e[iter-1]) {//good move
         direc[iter] = 8;
         p = p - 1; q = q + 1;
      }
      else {
         e[iter] = e[iter-1];
         out1.printf("Iteration %2i", iter);
         out1.printf("%30s\n", ": pattern move not good!");
         continue; //for loop
      }
   }//end if w/in boundary
   else {//outside the boundary
      e[iter] = e[iter-1];
      out1.printf("Iteration %2i", iter);
      out1.printf("%30s\n", ": pattern move not feasible!");
```

```
                continue; //for loop
            }
          }

        }//end else, iter >= 2 and e[iter-1] < e[iter-2]

        tabux = xnow; tabuy = ynow;
        e[iter] = etemp; xnow = xtemp;
        ynow = ytemp; znow = ztemp;
        if (p < pmin) pmin = p;
        if (p > pmax) pmax = p;
        if (q < qmin) qmin = q;
        if (q > qmax) qmax = q;
        if (e[iter] < ebest) {
          ebest = e[iter]; xopt = xnow;
          yopt = ynow; zopt = znow;
        }

    } //end for loop

     //Write in output file
        out1.printf("%35s", "Hence, the solution obtained after ");
        out1.printf("iteration %2i", iter);
        out1.printf("%1s\n", ":");
        out1.printf("x = %4.1f", xopt);
        out1.printf(", y = %4.1f", yopt);
        out1.printf(", z = %9.5f", zopt);
        out1.printf(" with emax(-) = %9.6f\n", ebest);
        out1.printf("%36s\n", "Plate is divided in 21 row x 31 col.");
        out1.printf("%15s", "Area searched: ");
        out1.printf(" row = %2i", pmin+1);
        out1.printf(" to row = %2i", pmax+1);
        out1.printf(" and col = %2i", qmin+1);
        out1.printf(" to col = %2i\n", qmax+1);

    System.out.println("Another starting point for emax(-)? Y/N");
    reply = in.readChar();
} while (reply == 'Y'); //end do/while loop

//Finding a point with an optimum value of emax(+)
    System.out.println("Search for a solution with emax(+)");
    //initial values
    ebest = -100.0; tabux = 100.0; tabuy = 100.0;
    pmin = 20; pmax = 0; qmin = 30; qmax = 0;

do {
```

```
badmove = 0;
System.out.println("Enter row index [p] of the start point: ");
p = in.readInt();
System.out.println("Enter column index [q] of the start point: ");
q = in.readInt();
if((p <= pmax && p >= pmin) && (q <= qmax && q >= qmin)) {
 System.out.println("Enter a point outside p = " + pmin + " to " + pmax
   + " & q = " + qmin + " to " + qmax);
 continue;//do loop
}
System.out.println("Enter e value of the start point: ");
   e[0] = in.readDouble();
   xnow = x[p][q];
   ynow = y[p][q];
   znow = z[p][q];
   out1.printf("%2s\n", "  ");
   out1.printf("%35s\n", "Starting point to find emax(+) was ");
   out1.printf("x = %4.1f", xnow);
   out1.printf(", y = %4.1f", ynow);
   out1.printf(", z = %9.5f", znow);
   out1.printf(" with e = %9.6f\n", e[0]);
   if (p < pmin) pmin = p;
   if (p > pmax) pmax = p;
   if (q < qmin) qmin = q;
   if (q > qmax) qmax = q;
   if (e[0] > ebest) {
      ebest = e[0]; xopt = xnow;
      yopt = ynow; zopt = znow;
   }

for (iter = 1; iter <= itermax; iter++) {
   if (iter < 2 || (e[iter - 1] <= e[iter - 2])) {

    //Coordinate search exploratory move
    if ((xnow + delta) <= XUL) {//within boundary
       x1 = x[p][q+1]; y1 = ynow;
       if ((Math.abs(x1 - tabux) <= 0.1e-5) && (Math.abs(y1 - tabuy) <= 0.1e-5))
           e1 = -100.0; //tabu
       else {
           z1 = z[p][q+1];
           e1 = z1 - (z0 + l0*x1 + m0*y1);
       }//end else
    }//end if w/in boundary
    else e1 = -100.0;//outside the boundary

    if (e1 > e[iter-1]) {//good
```

```
    if ((y1 + delta) <= YUL) {//within boundary
      x2 = x1; y2 = y[p+1][q+1];
      if ((Math.abs(x2 - tabux) <= 0.1e-5) && (Math.abs(y2 - tabuy) <= 0.1e-5))
        e2 = -100.0; //tabu
      else {
        z2 = z[p+1][q+1];
        e2 = z2 - (z0 + l0*x2 + m0*y2);
      }//end else
    }//end if w/in boundary
    else e2 = -100.0;//outside the boundary

    if (e2 > e1) {
      etemp = e2; xtemp = x2; ytemp = y2; ztemp = z2;
      p = p + 1; q = q + 1; direc[iter] = 2;
    }
    else {//e2 <= e1
     if ((y1 - delta) >= YLL) {//within boundary
      x8 = x1; y8 = y[p-1][q+1];
      if ((Math.abs(x8 - tabux) <= 0.1e-5) && (Math.abs(y8 - tabuy) <= 0.1e-5))
        e8 = -100.0; //tabu
      else {
        z8 = z[p-1][q+1];
        e8 = z8 - (z0 + l0*x8 + m0*y8);
      }//end else
     }//end if w/in boundary
     else e8 = -100.0;//outside the boundary

     if (e8 > e1) {
       etemp = e8; xtemp = x8; ytemp = y8; ztemp = z8;
       p = p - 1; q = q + 1; direc[iter] = 8;
     }
     else {//e8 <= e1 (already e2 <= e1)
       etemp = e1; xtemp = x1; ytemp = y1; ztemp = z1;
       q = q + 1; direc[iter] = 1; //p not changed
     }//end else e8 <= e1
    }//end else e2 <= e1
  }//end if e1 > e[iter-1]
  else {//e1 <= e[iter-1]
   if ((xnow - delta) >= XLL) {//within boundary
     x5 = x[p][q-1]; y5 = ynow;
     if ((Math.abs(x5 - tabux) <= 0.1e-5) && (Math.abs(y5 - tabuy) <= 0.1e-5))
        e5 = -100.0; //tabu
     else {
        z5 = z[p][q-1];
        e5 = z5 - (z0 + l0*x5 + m0*y5);
     }//end else
```

```
}//end if w/in boundary
else e5 = -100.0;//outside the boundary

if (e5 > e[iter-1]) {//good
  if ((y5 + delta) <= YUL) {//within boundary
    x4 = x5; y4 = y[p+1][q-1];
    if ((Math.abs(x4 - tabux) <= 0.1e-5) && (Math.abs(y4 - tabuy) <= 0.1e-5))
      e4 = -100.0; //tabu
    else {
      z4 = z[p+1][q-1];
      e4 = z4 - (z0 + l0*x4 + m0*y4);
    }//end else
  }//end if within boundary
  else e4 = -100.0;//outside the boundary

  if (e4 > e5) {
    etemp = e4; xtemp = x4; ytemp = y4; ztemp = z4;
    p = p + 1; q = q - 1; direc[iter] = 4;
  }
  else {//e4 <= e5
    if ((y5 - delta) >= YLL) {//within boundary
      x6 = x5; y6 = y[p-1][q-1];
      if ((Math.abs(x6 - tabux) <= 0.1e-5) && (Math.abs(y6 - tabuy) <= 0.1e-5))
          e6 = -100.0; //tabu
      else {
        z6 = z[p-1][q-1];
        e6 = z6 - (z0 + l0*x6 + m0*y6);
      }//end else
    }//end if within boundary
    else e6 = -100.0;//outside the boundary

    if (e6 > e5) {
      etemp = e6; xtemp = x6; ytemp = y6; ztemp = z6;
      p = p - 1; q = q - 1; direc[iter] = 6;
    }
    else {//e6 <= e5, already e4 <= e5
      etemp = e5; xtemp = x5; ytemp = y5; ztemp = z5;
      q = q - 1; direc[iter] = 5;
    }//end else e6 <= e5
  }//end else e4 <= e5
}//end if e5 > e[iter-1]
else {//e5 <= e[iter-1]
  if ((ynow + delta) <= YUL) {//within boundary
    x3 = xnow;
    y3 = y[p+1][q];
    if ((Math.abs(x3 - tabux) <= 0.1e-5) && (Math.abs(y3 - tabuy) <= 0.1e-5))
```

```
    e3 = -100.0; //tabu
  else {
    z3 = z[p+1][q];
    e3 = z3 - (z0 + l0*x3 + m0*y3);
  }//end else
}//end if within boundary
else e3 = -100.0;//outside the boundary

if (e3 > e[iter-1]) {//good
  etemp = e3; xtemp = x3; ytemp = y3; ztemp = z3;
  p = p + 1; direc[iter] = 3;
}
else {//e3 <= e[iter-1]
  if ((ynow - delta) >= YLL) {//within boundary
    x7 = xnow;
    y7 = y[p-1][q];
    if ((Math.abs(x7 - tabux) <= 0.1e-5) && (Math.abs(y7 - tabuy) <= 0.1e-5))
      e7 = -100.0; //tabu
    else {
      z7 = z[p-1][q];
      e7 = z7 - (z0 + l0*x7 + m0*y7);
    }//end else
  }//end if within boundary
  else e7 = -100.0;//outside the boundary

  if (e7 > e[iter-1]) {//good
     etemp = e7; xtemp = x7; ytemp = y7; ztemp = z7;
     p = p - 1; direc[iter] = 7;
  }
  else {// e7 <= e[iter-1], e1, e5, e3 are already bad. If e1 is
  //bad, e2 & e8 are not computed. If e5 is bad, e4 & e6 are not
  //computed. So, choose the least bad from e1, e3, e5, e7.

  if (e1 >= e3){
    if (e1 >= e5) {
      if (e1 >= e7) {
        etemp = e1; xtemp = x1;
        ytemp = y1; ztemp = z1;
        q = q + 1; direc[iter] = 1;
      }//end if e1 >= e7
      else {//e1 < e7
         etemp = e7; xtemp = x7;
         ytemp = y7; ztemp = z7;
         p = p - 1; direc[iter] = 7;
      }//end else e1 < e7
    }//end if e1 >= e5
```

```
        else {// e1 < e5
          if (e5 >= e7) {
            etemp = e5; xtemp = x5;
            ytemp = y5; ztemp = z5;
            q = q - 1; direc[iter] = 5;
          }//end if e5 >= e7
          else {//e5 < e7
             etemp = e7; xtemp = x7;
             ytemp = y7; ztemp = z7;
             p = p - 1; direc[iter] = 7;
          }//end else e5 < e7
        }//end else e1 < e5
      }//end if e1 >= e3
      else {//e1 < e3
        if (e3 >= e5) {
          if (e3 >= e7) {
            etemp = e3; xtemp = x3;
            ytemp = y3; ztemp = z3;
            p = p + 1; direc[iter] = 3;
          }//end if e3 >= e7
          else {//e3 < e7
             etemp = e7; xtemp = x7;
             ytemp = y7; ztemp = z7;
             p = p - 1; direc[iter] = 7;
          }//end else e3 < e7
        }//end if e3 >= e5
        else { // e3 < e5
          if (e5 >= e7) {
            etemp = e5; xtemp = x5;
            ytemp = y5; ztemp = z5;
            q = q - 1; direc[iter] = 5;
          }//end if e5 >= e7
          else {//e5 < e7
             etemp = e7; xtemp = x7;
             ytemp = y7; ztemp = z7;
             p = p - 1; direc[iter] = 7;
          }//end else e5 < e7
        }//end else e3 < e5
      }//end else e1 < e3
      }//end else e7 <= e[iter-1]
    }//end else e3 <= e[iter-1]
  }//end else e5 <= e[iter-1]
}//end else e1 <= e[iter-1]

if (etemp < e[iter-1]) { //bad move
   badmove = badmove + 1;
```

```
    if (badmove > badmovemax){
      out1.printf("%32s\n", "Bad moves exceeded the max limit");
      break;
    }//exit from the for loop
  }

}//end if iter < 2  or e[iter-1] <= e[iter-2]
else { // iter >=2 and e[iter-1] > e[iter-2]
 //Check for HJ pattern move in the direction of the prev move.
 //No need to check for tabu as it is not returning to prev point.
   if (direc[iter-1] == 1) {
     if ((xnow + delta) <= XUL) {//within boundary
       ytemp = ynow;
       xtemp = x[p][q+1];
       ztemp = z[p][q+1];
       etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
       if (etemp > e[iter-1]) {//good move
         direc[iter] = 1;
         q = q + 1;
       }
       else {
         e[iter] = e[iter-1];
         out1.printf("Iteration %2i", iter);
         out1.printf("%30s\n", ": pattern move not good!");
         continue; //for loop
       }
     }//end if w/in boundary
     else {//outside the boundary
       e[iter] = e[iter-1];
       out1.printf("Iteration %2i", iter);
       out1.printf("%30s\n", ": pattern move not feasible!");
       continue; //for loop
     }
   }
   else if (direc[iter-1] == 2) {
     if ((xnow + delta <= XUL) && (ynow + delta <= YUL)) {
       xtemp = x[p+1][q+1];
       ytemp = y[p+1][q+1];
       ztemp = z[p+1][q+1];
       etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
       if (etemp > e[iter-1]) {//good move
         direc[iter] = 2;
         p = p + 1; q = q + 1;
       }
       else {
         e[iter] = e[iter-1];
```

```
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": pattern move not good!");
            continue; //for loop
          }
        }//end if w/in boundary
        else {//outside the boundary
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not feasible!");
          continue; //for loop
        }
      }
      else if (direc[iter-1] == 3) {
        if ((ynow + delta) <= YUL) {//within boundary
          xtemp = xnow;
          ytemp = y[p+1][q];
          ztemp = z[p+1][q];
          etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
          if (etemp > e[iter-1]) {//good move
            direc[iter] = 3;
            p = p + 1;
          }
          else {
            e[iter] = e[iter-1];
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": pattern move not good!");
            continue; //for loop
          }
        }//end if w/in boundary
        else {//outside the boundary
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not feasible!");
          continue; //for loop
        }
      }
      else if (direc[iter-1] == 4) {
        if ((xnow - delta >= XLL) && (ynow + delta <= YUL)) {
          xtemp = x[p+1][q-1];
          ytemp = y[p+1][q-1];
          ztemp = z[p+1][q-1];
          etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
          if (etemp > e[iter-1]) {//good move
            direc[iter] = 4;
            p = p + 1; q = q - 1;
          }
```

```
        else {
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not good!");
          continue; //for loop
        }
      }//end if w/in boundary
      else {//outside the boundary
        e[iter] = e[iter-1];
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": pattern move not feasible!");
        continue; //for loop
      }
    }
    else if (direc[iter-1] == 5) {
      if ((xnow - delta) >= XLL) {//within boundary
        ytemp = ynow;
        xtemp = x[p][q-1];
        ztemp = z[p][q-1];
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp > e[iter-1]) {//good move
          direc[iter] = 5;
          q = q - 1;
        }
        else {
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not good!");
          continue; //for loop
        }
      }//end if w/in boundary
      else {//outside the boundary
        e[iter] = e[iter-1];
        out1.printf("Iteration %2i", iter);
        out1.printf("%30s\n", ": pattern move not feasible!");
        continue; //for loop
      }
    }
    else if (direc[iter-1] == 6) {
      if ((xnow - delta >= XLL) && (ynow - delta >= YLL)) {
        xtemp = x[p-1][q-1];
        ytemp = y[p-1][q-1];
        ztemp = z[p-1][q-1];
        etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
        if (etemp > e[iter-1]) {//good move
          direc[iter] = 6;
```

```
            p = p - 1; q = q - 1;
          }
          else {
            e[iter] = e[iter-1];
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": pattern move not good!");
            continue; //for loop
          }
        }//end if w/in boundary
        else {//outside the boundary
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not feasible!");
          continue; //for loop
        }
      }
      else if (direc[iter-1] == 7) {
        if ((ynow - delta) >= YLL) {//within boundary
          xtemp = xnow;
          ytemp = y[p-1][q];
          ztemp = z[p-1][q];
          etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
          if (etemp > e[iter-1]) {//good move
            direc[iter] = 7;
            p = p - 1;
          }
          else {
            e[iter] = e[iter-1];
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": pattern move not good!");
            continue; //for loop
          }
        }//end if w/in boundary
        else {//outside the boundary
          e[iter] = e[iter-1];
          out1.printf("Iteration %2i", iter);
          out1.printf("%30s\n", ": pattern move not feasible!");
          continue; //for loop
        }
      }
      else {// direc[iter-1] == 8
        if ((xnow + delta <= XUL) && (ynow - delta >= YLL)) {
          xtemp = x[p-1][q+1];
          ytemp = y[p-1][q+1];
          ztemp = z[p-1][q+1];
          etemp = ztemp - (z0 + l0*xtemp + m0*ytemp);
```

```
            if (etemp > e[iter-1]) {//good move
               direc[iter] = 8;
               p = p - 1; q = q + 1;
            }
            else {
               e[iter] = e[iter-1];
               out1.printf("Iteration %2i", iter);
               out1.printf("%30s\n", ": pattern move not good!");
               continue; //for loop
            }
         }//end if w/in boundary
         else {//outside the boundary
            e[iter] = e[iter-1];
            out1.printf("Iteration %2i", iter);
            out1.printf("%30s\n", ": pattern move not feasible!");
            continue; //for loop
         }
      }

   }//end else, iter >= 2 and e[iter-1] > e[iter-2]

   tabux = xnow; tabuy = ynow;
   e[iter] = etemp; xnow = xtemp;
   ynow = ytemp; znow = ztemp;
   if (p < pmin) pmin = p;
   if (p > pmax) pmax = p;
   if (q < qmin) qmin = q;
   if (q > qmax) qmax = q;
   if (e[iter] > ebest) {
      ebest = e[iter]; xopt = xnow;
      yopt = ynow; zopt = znow;
   }

} //end for loop

 //Write in output file
   out1.printf("%35s", "Hence, the solution obtained after ");
   out1.printf("iteration %2i", iter);
   out1.printf("%1s\n", ":");
   out1.printf("x = %4.1f", xopt);
   out1.printf(", y = %4.1f", yopt);
   out1.printf(", z = %9.5f", zopt);
   out1.printf(" with emax(+) = %9.6f\n", ebest);
   out1.printf("%36s\n", "Plate is divided in 21 row x 31 col.");
   out1.printf("%15s", "Area searched: ");
   out1.printf(" row = %2i", pmin+1);
```

```
        out1.printf(" to row = %2i", pmax+1);
        out1.printf(" and col = %2i", qmin+1);
        out1.printf(" to col = %2i\n", qmax+1);

      System.out.println("Another starting point for emax(+)? Y/N");
      reply = in.readChar();
    } while (reply == 'Y'); //end do/while loop

      out1.close();//Close output file

  } //end main method
} //end HybridSearch class
```

APPENDIX C

SAS FILES

C.1 Straightness

C.1.1 Code

```
Title '3-Factor Nested-Factorial Design for Straightness';
data Straightness;
input mfg$ plate$ step @@;
do i=1 to 4;
   input point p_error @@;
        output;
end;
cards;
end al4 0.05 8 6.1 7 0.0 7 0.0 7 6.9
end al12 0.05 7 6.75 7 11.4 6 3.25 4 0.61
end al9 0.05 5 4.7 6 2.0 7 2.04 3 9.6
end al2 0.05 6 9.58 5 9.84 6 15.17 8 3.76
end al4 0.1 7 12.8 6 0.14 6 7.2 6 8.2
end al12 0.1 7 6.75 5 6.08 5 2.69 4 0.61
end al9 0.1 4 4.7 7 2.0 7 4.2 4 3.8
end al2 0.1 4 21.86 5 9.84 5 14.53 8 2.43
end al4 0.2 7 14.7 6 10.2 4 20.9 4 19.7
end al12 0.2 5 14.23 4 6.97 5 5.82 4 0.61
end al9 0.2 3 13.3 6 0.0 5 2.6 4 8.26
end al2 0.2 4 19.05 4 3.54 5 24.9 5 30.62
face ci3 0.05 11 7.35 9 8.6 11 6.74 10 1.24
face ci1 0.05 9 6.3 11 3.9 9 18.4 9 0.85
face ci11 0.05 11 9.4 10 12.0 13 4.2 9 2.9
face ci7 0.05 10 4.65 8 4.17 6 7.93 10 2.7
face ci3 0.1 8 11.0 7 8.6 9 9.86 12 0.31
face ci1 0.1 8 2.0 8 13.9 10 13.56 7 13.75
face ci11 0.1 10 4.0 9 12.0 7 4.4 9 0.5
face ci7 0.1 6 0.03 7 4.17 7 7.93 7 0.65
face ci3 0.2 6 2.7 6 6.06 6 2.33 9 0.48
face ci1 0.2 6 2.0 7 13.9 7 25.2 6 16.0
face ci11 0.2 7 6.2 6 3.9 6 5.7 8 2.3
face ci7 0.2 6 0.03 7 4.17 6 7.93 5 11.14
;
proc glm;
class mfg plate step;
model point p_error=mfg plate(mfg) step mfg*step step*plate(mfg);
test h=mfg e=plate(mfg);
test h=step mfg*step e=step*plate(mfg);
means mfg plate(mfg) step mfg*step/tukey;
output out=Strout p=pred r=resid;
```

```
proc print data=Strout;
proc univariate plot normal;
  var resid;
run;
```

C.1.2 Output

3-Factor Nested-Factorial Design for Straightness
13:11 Thursday, May 30, 2002

Class Level Information

Class	Levels	Values
mfg	2	end face
plate	8	al12 al2 al4 al9 ci1 ci11 ci3 ci7
step	3	0.05 0.1 0.2

Number of observations 96

The GLM Procedure

Dependent Variable: point

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	23	308.2395833	13.4017210	8.06	<.0001
Error	72	119.7500000	1.6631944		
Corrected Total	95	427.9895833			

R-Square	Coeff Var	Root MSE	point Mean
0.720203	18.90173	1.289649	6.822917

Source	DF	Type I SS	Mean Square	F Value	Pr > F
mfg	1	168.0104167	168.0104167	101.02	<.0001
plate(mfg)	6	30.8958333	5.1493056	3.10	0.0094
step	2	90.5833333	45.2916667	27.23	<.0001
mfg*step	2	12.3333333	6.1666667	3.71	0.0293
plate*step(mfg)	12	6.4166667	0.5347222	0.32	0.9832

Source	DF	Type III SS	Mean Square	F Value	Pr > F
mfg	1	168.0104167	168.0104167	101.02	<.0001
plate(mfg)	6	30.8958333	5.1493056	3.10	0.0094
step	2	90.5833333	45.2916667	27.23	<.0001
mfg*step	2	12.3333333	6.1666667	3.71	0.0293
plate*step(mfg)	12	6.4166667	0.5347222	0.32	0.9832

Tests of Hypotheses Using the Type III MS for plate(mfg) as an Error Term

Source	DF	Type III SS	Mean Square	F Value	Pr > F
mfg	1	168.0104167	168.0104167	32.63	0.0012

Tests of Hypotheses Using the Type III MS for plate*step(mfg) as an Error Term

step	2	90.5833333	45.2916667	84.70	<.0001
mfg*step	2	12.33333333	6.16666667	11.53	0.0016

Dependent Variable: p_error

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	23	1728.497233	75.152054	2.53	0.0015
Error	72	2134.538500	29.646368		
Corrected Total	95	3863.035733			

R-Square	Coeff Var	Root MSE	p_error Mean
0.447445	73.31483	5.444848	7.426667

Source	DF	Type I SS	Mean Square	F Value	Pr > F
mfg	1	61.6001042	61.6001042	2.08	0.1538
plate(mfg)	6	894.4222458	149.0703743	5.03	0.0002
step	2	222.5879396	111.2939698	3.75	0.0281
mfg*step	2	169.0385021	84.5192510	2.85	0.0643
plate*step(mfg)	12	380.8484417	31.7373701	1.07	0.3974

Source	DF	Type III SS	Mean Square	F Value	Pr > F
mfg	1	61.6001042	61.6001042	2.08	0.1538
plate(mfg)	6	894.4222458	149.0703743	5.03	0.0002
step	2	222.5879396	111.2939698	3.75	0.0281
mfg*step	2	169.0385021	84.5192510	2.85	0.0643
plate*step(mfg)	12	380.8484417	31.7373701	1.07	0.3974

Tests of Hypotheses Using the Type III MS for plate(mfg) as an Error Term

Source	DF	Type III SS	Mean Square	F Value	Pr > F
mfg	1	61.60010417	61.60010417	0.41	0.5441

Tests of Hypotheses Using the Type III MS for plate*step(mfg) as an Error Term

step	2	222.5879396	111.2939698	3.51	0.0632
mfg*step	2	169.0385021	84.5192510	2.66	0.1104

Tukey's Studentized Range (HSD) Test for point

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05

Error Degrees of Freedom	72
Error Mean Square	1.663194
Critical Value of Studentized Range	2.81929
Minimum Significant Difference	0.5248

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	mfg
A	8.1458	48	face
B	5.5000	48	end

The GLM Procedure

Level of plate	Level of mfg	N	point Mean	point Std Dev	p_error Mean	p_error Std Dev
al12	end	12	5.250000	1.215431	5.480833	4.27931698
al2	end	12	5.41666667	1.37895437	13.7600000	8.97246698
al4	end	12	6.25000000	1.21543109	8.9033333	7.14497577
al9	end	12	5.08333333	1.50504203	4.7666667	3.81153592
ci1	face	12	8.08333333	1.56427929	10.8133333	7.68375185
ci11	face	12	8.75000000	2.09436473	5.6250000	3.68982015
ci3	face	12	8.66666667	2.10338832	5.4391667	3.83621371
ci7	face	12	7.08333333	1.56427929	4.6250000	3.53643760

Tukey's Studentized Range (HSD) Test for point

Alpha	0.05
Error Degrees of Freedom	72
Error Mean Square	1.663194
Critical Value of Studentized Range	3.38440
Minimum Significant Difference	0.7716

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	step
A	7.9688	32	0.05
B	6.9063	32	0.1
C	5.5938	32	0.2

The GLM Procedure

Level of mfg	Level of step	N	point Mean	point Std Dev	p_error Mean	p_error Std Dev
end	0.05	16	6.18750000	1.37689264	5.7312500	4.47959503
end	0.1	16	5.62500000	1.31021627	6.7393750	5.75181649
end	0.2	16	4.68750000	1.01447852	12.2125000	9.05145255
face	0.05	16	9.75000000	1.57056253	6.3331250	4.43464349

face	0.1	16	8.18750000	1.55857841	6.6662500	5.23899975
face	0.2	16	6.50000000	0.96609178	6.8775000	6.71777294

Dependent Variable: point

Obs	Residual	Obs	Residual	Obs	Residual	Obs	Residual
1	0.75	2	-0.25	3	-0.25	4	-0.25
5	1.00	6	1.00	7	0.00	8	-2.00
9	-0.25	10	0.75	11	1.75	12	-2.25
13	-0.25	14	-1.25	15	-0.25	16	1.75
17	0.75	18	-0.25	19	-0.25	20	-0.25
21	1.75	22	-0.25	23	-0.25	24	-1.25
25	-1.50	26	1.50	27	1.50	28	-1.50
29	-1.50	30	-0.50	31	-0.50	32	2.50
33	1.75	34	0.75	35	-1.25	36	-1.25
37	0.50	38	-0.50	39	0.50	40	-0.50
41	-1.50	42	1.50	43	0.50	44	-0.50
45	-0.50	46	-0.50	47	0.50	48	0.50
49	0.75	50	-1.25	51	0.75	52	-0.25
53	-0.50	54	1.50	55	-0.50	56	-0.50
57	0.25	58	-0.75	59	2.25	60	-1.75
61	1.50	62	-0.50	63	-2.50	64	1.50
65	-1.00	66	-2.00	67	0.00	68	3.00
69	-0.25	70	-0.25	71	1.75	72	-1.25
73	1.25	74	0.25	75	-1.75	76	0.25
77	-0.75	78	0.25	79	0.25	80	0.25
81	-0.75	82	-0.75	83	-0.75	84	2.25
85	-0.50	86	0.50	87	0.50	88	-0.50
89	0.25	90	-0.75	91	-0.75	92	1.25
93	0.00	94	1.00	95	0.00	96	-1.00

The UNIVARIATE Procedure
Variable: resid

Moments

N	96	Sum Weights	96
Mean	0	Sum Observations	0
Std Deviation	1.12273163	Variance	1.26052632
Skewness	0.28200601	Kurtosis	-0.1737485
Uncorrected SS	119.75	Corrected SS	119.75
Coeff Variation	.	Std Error Mean	0.11458832

Basic Statistical Measures

Location		Variability	
Mean	0.00000	Std Deviation	1.12273
Median	-0.25000	Variance	1.26053

Mode	-0.25000	Range	5.50000
		Interquartile Range	1.50000

Tests for Location: Mu0=0

Test	-Statistic-		-----p Value------	
Student's t	t	0	Pr > |t|	1.0000
Sign	M	-4.5	Pr >= |M|	0.4119
Signed Rank	S	-89	Pr >= |S|	0.7429

Tests for Normality

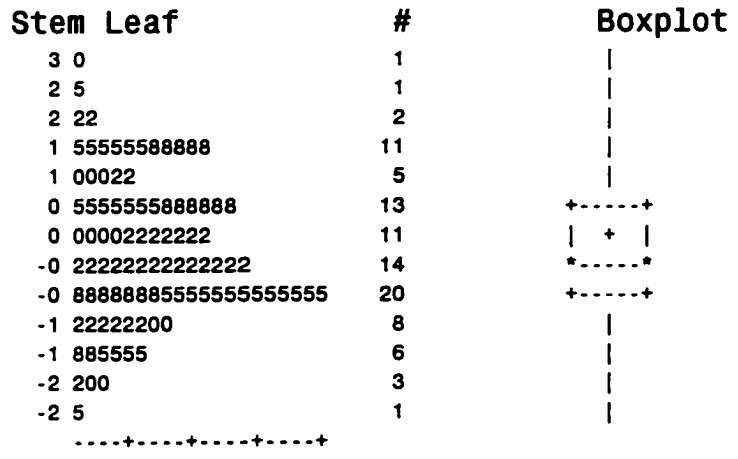
Test	--Statistic---		-----p Value------	
Shapiro-Wilk	W	0.981755	Pr < W	0.2028
Kolmogorov-Smirnov	D	0.129771	Pr > D	<0.0100
Cramer-von Mises	W-Sq	0.150828	Pr > W-Sq	0.0232
Anderson-Darling	A-Sq	0.775651	Pr > A-Sq	0.0438

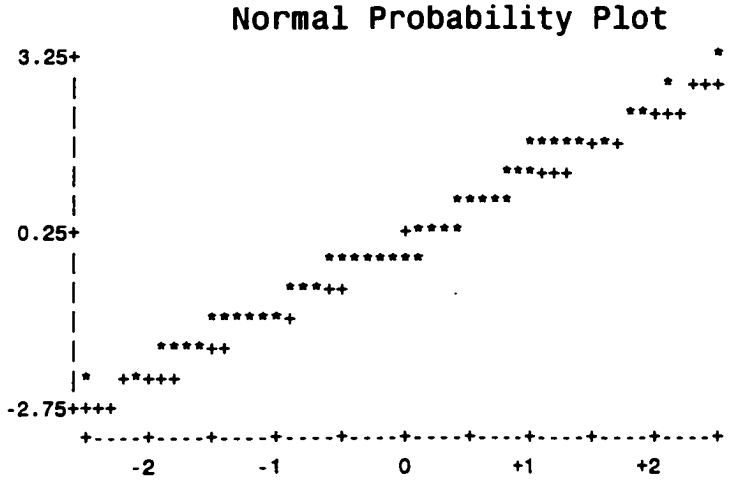
Quantiles (Definition 5)

Quantile	Estimate
100% Max	3.00
99%	3.00
95%	1.75
90%	1.50
75% Q3	0.75
50% Median	-0.25
25% Q1	-0.75
10%	-1.50
5%	-1.75
1%	-2.50
0% Min	-2.50

Extreme Observations

-----Lowest----		----Highest---	
Value	Obs	Value	Obs
-2.50	63	1.75	33
-2.25	12	2.25	84
-2.00	66	2.25	59
-2.00	8	2.50	32
-1.75	75	3.00	68





C.2 Flatness

C.2.1 Code

```
Title '4-Factor Nested-Factorial Design for Flatness';
data Flatness;
input mfg$ plate$ algorithm$ strategy$ @@;
do i=1 to 4;
   input point p_error @@;
        output;
end;
cards;
end all0 ts rs0bm1 10 34.8 8 21.8 11 46.1 8 35.2
end all0 ts rs0bm3 16 33.7 15 26.7 18 41.9 15 32.8
end all0 ts rs0bm5 26 33.7 22 26.7 24 41.9 25 32.8
end all0 ts rs1bm1 15 34.8 14 21.8 17 21.8 17 16.6
end all0 hs rs0bm1 9 33.7 8 21.8 10 46.1 8 35.2
```

end al10 hs rs0bm3 14 33.7 31 21.8 19 46.1 15 35.2
end al10 hs rs0bm5 23 33.7 19 21.8 31 46.1 29 35.2
end al10 hs rs1bm1 14 33.7 16 17.3 16 16.3 12 24.7
end al5 ts rs0bm1 10 24.3 15 10.4 12 46.9 10 44.1
end al5 ts rs0bm3 20 26.9 24 10.4 20 48.1 18 44.1
end al5 ts rs0bm5 28 26.9 35 10.4 30 4.8 26 44.1
end al5 ts rs1bm1 15 24.2 21 10.4 17 30.0 18 37.7
end al5 hs rs0bm1 13 30.0 11 10.4 12 44.1 10 44.1
end al5 hs rs0bm3 21 30.0 16 10.4 22 44.1 17 44.1
end al5 hs rs0bm5 28 30.0 26 10.4 32 44.1 25 44.1
end al5 hs rs1bm1 18 30.0 19 10.4 16 31.2 15 37.7
end al7 ts rs0bm1 12 40.8 17 35.3 19 33.3 12 49.6
end al7 ts rs0bm3 20 41.2 21 35.3 29 33.3 19 49.6
end al7 ts rs0bm5 32 41.2 30 36.4 41 33.3 29 38.6
end al7 ts rs1bm1 15 40.8 31 30.7 22 33.3 16 47.4
end al7 hs rs0bm1 10 40.0 16 35.3 16 42.9 12 49.6
end al7 hs rs0bm3 17 40.8 21 35.3 23 42.9 19 49.6
end al7 hs rs0bm5 26 41.2 30 36.4 31 42.9 27 49.8
end al7 hs rs1bm1 14 40.0 28 30.7 19 42.9 17 47.4
end al4 ts rs0bm1 13 57.1 14 51.6 13 41.8 10 21.8
end al4 ts rs0bm3 19 57.1 22 51.6 22 24.1 17 21.8
end al4 ts rs0bm5 27 57.1 32 41.1 27 24.1 23 21.8
end al4 ts rs1bm1 18 48.3 22 38.2 18 29.5 21 8.0
end al4 hs rs0bm1 15 50.9 14 51.6 12 58.7 10 21.8
end al4 hs rs0bm3 21 50.9 21 51.6 18 58.7 16 21.8
end al4 hs rs0bm5 27 50.9 29 46.9 26 58.7 21 21.8
end al4 hs rs1bm1 20 48.3 21 38.2 16 29.5 18 21.8
face ci2 ts rs0bm1 11 29.2 13 17.6 13 13.8 12 28.2
face ci2 ts rs0bm3 20 30.9 21 7.1 19 10.5 21 28.2
face ci2 ts rs0bm5 27 24.2 27 7.1 25 10.5 29 23.8
face ci2 ts rs1bm1 19 11.5 19 17.6 17 13.8 21 28.2
face ci2 hs rs0bm1 9 31.6 11 17.6 13 14.2 13 27.1
face ci2 hs rs0bm3 17 31.3 20 6.1 18 10.8 23 22.6
face ci2 hs rs0bm5 26 31.9 24 6.1 27 10.8 29 21.5
face ci2 hs rs1bm1 16 11.5 17 17.6 17 14.2 22 12.2
face ci9 ts rs0bm1 12 7.4 11 9.4 9 17.4 13 7.9
face ci9 ts rs0bm3 19 7.3 17 9.4 17 17.4 19 8.1
face ci9 ts rs0bm5 24 7.3 24 9.4 25 17.4 29 16.3
face ci9 ts rs1bm1 21 7.4 16 9.4 15 8.3 23 7.9
face ci9 hs rs0bm1 10 7.4 11 9.4 10 15.7 9 8.9
face ci9 hs rs0bm3 15 7.4 17 9.4 18 17.4 16 8.9
face ci9 hs rs0bm5 22 7.4 23 9.4 26 17.4 20 8.9
face ci9 hs rs1bm1 18 7.4 16 9.4 14 5.9 18 4.6
face ci11 ts rs0bm1 13 2.8 12 9.9 16 11.6 16 0.26
face ci11 ts rs0bm3 24 2.8 21 9.3 29 11.6 25 0.26
face ci11 ts rs0bm5 34 2.8 28 9.3 41 11.6 33 0.26
face ci11 ts rs1bm1 25 11.8 22 9.9 27 11.6 26 0.26
face ci11 hs rs0bm1 11 3.4 10 10.2 14 10.5 14 0.65
face ci11 hs rs0bm3 20 2.6 17 9.0 21 10.5 20 0.65
face ci11 hs rs0bm5 25 2.6 24 9.0 27 10.5 26 0.65
face ci11 hs rs1bm1 21 8.5 18 10.2 26 7.0 21 2.8
face ci5 ts rs0bm1 13 6.0 15 4.8 13 9.3 8 15.3
face ci5 ts rs0bm3 21 6.0 22 4.8 25 9.3 16 15.3
face ci5 ts rs0bm5 29 6.0 29 4.8 37 9.3 24 15.3
face ci5 ts rs1bm1 21 6.0 26 4.8 20 8.6 15 14.4
face ci5 hs rs0bm1 13 6.0 13 4.8 14 9.3 8 15.3

```
face ci5 hs rs0bm3 20 6.0 18 4.8 23 9.3 15 15.3
face ci5 hs rs0bm5 27 6.0 24 4.8 32 9.3 21 15.3
face ci5 hs rs1bm1 19 6.0 19 4.8 24 8.6 16 19.7
;
proc glm;
class mfg plate strategy algorithm;
model point p_error = mfg plate(mfg) strategy algorithm mfg*strategy
        mfg*algorithm strategy*plate(mfg) strategy*algorithm algorithm*plate(mfg)
        mfg*strategy*algorithm strategy*algorithm*plate(mfg);
test h=mfg e=plate(mfg);
test h=strategy mfg*strategy e=strategy*plate(mfg);
test h=algorithm mfg*algorithm e=algorithm*plate(mfg);
test h=strategy*algorithm mfg*strategy*algorithm e=strategy*algorithm*plate(mfg);
means mfg plate(mfg) strategy algorithm mfg*strategy
        mfg*algorithm strategy*algorithm mfg*strategy*algorithm/tukey;
output out=Flatout p=pred r=resid;
proc print data=Flatout;
proc univariate plot normal;
  var resid;
run;
```

C.2.2 Output

4-Factor Nested-Factorial Design for Flatness

13:11 Thursday, May 30, 2002

Class Level Information

Class	Levels	Values
mfg	2	end face
plate	8	al10 al4 al5 al7 ci11 ci2 ci5 ci9
strategy	4	rs0bm1 rs0bm3 rs0bm5 rs1bm1
algorithm	2	hs ts

Number of observations 256

The GLM Procedure

Dependent Variable: point

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	63	9044.21484	143.55897	14.14	<.0001
Error	192	1949.75000	10.15495		
Corrected Total	255	10993.96484			

R-Square	Coeff Var	Root MSE	point Mean
0.822653	16.35179	3.186683	19.48828

Source	DF	Type I SS	Mean Square	F Value	Pr > F
mfg	1	10.160156	10.160156	1.00	0.3184
plate(mfg)	6	743.710938	123.951823	12.21	<.0001
strategy	3	7715.386719	2571.795573	253.26	<.0001
algorithm	1	164.160156	164.160156	16.17	<.0001
mfg*strategy	3	51.011719	17.003906	1.67	0.1739
mfg*algorithm	1	32.347656	32.347656	3.19	0.0759
plate*strategy(mfg)	18	99.070313	5.503906	0.54	0.9349
strategy*algorithm	3	34.449219	11.483073	1.13	0.3378
plate*algorithm(mfg)	6	116.023438	19.337240	1.90	0.0820
mfg*strateg*algorith	3	8.761719	2.920573	0.29	0.8343
plat*stra*algor(mfg)	18	69.132813	3.840712	0.38	0.9904

Source	DF	Type III SS	Mean Square	F Value	Pr > F
mfg	1	10.160156	10.160156	1.00	0.3184
plate(mfg)	6	743.710938	123.951823	12.21	<.0001
strategy	3	7715.386719	2571.795573	253.26	<.0001
algorithm	1	164.160156	164.160156	16.17	<.0001
mfg*strategy	3	51.011719	17.003906	1.67	0.1739
mfg*algorithm	1	32.347656	32.347656	3.19	0.0759
plate*strategy(mfg)	18	99.070313	5.503906	0.54	0.9349
strategy*algorithm	3	34.449219	11.483073	1.13	0.3378
plate*algorithm(mfg)	6	116.023438	19.337240	1.90	0.0820
mfg*strateg*algorith	3	8.761719	2.920573	0.29	0.8343
plat*stra*algor(mfg)	18	69.132813	3.840712	0.38	0.9904

Tests of Hypotheses Using the Type III MS for plate(mfg) as an Error Term

Source	DF	Type III SS	Mean Square	F Value	Pr > F
mfg	1	10.160156	10.160156	0.08	0.7843

Tests of Hypotheses Using the Type III MS for plate*strategy(mfg) as an Error Term

strategy	3	7715.386719	2571.795573	467.27	<.0001
mfg*strategy	3	51.011719	17.003906	3.09	0.0533

Tests of Hypotheses Using the Type III MS for plate*algorithm(mfg) as an Error Term

algorithm	1	164.1601563	164.1601563	8.49	0.0269
mfg*algorithm	1	32.3476563	32.3476563	1.67	0.2434

Tests of Hypotheses Using the Type III MS for plat*stra*algor(mfg) as an Error Term

strategy*algorithm	3	34.44921875	11.48307292	2.99	0.0584
mfg*strateg*algorith	3	8.76171875	2.92057292	0.76	0.5308

Dependent Variable: p_error

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	63	45910.43629	728.73708	7.61	<.0001
Error	192	18386.56092	95.76334		

Corrected Total 255 64296.99721

R-Square	Coeff Var	Root MSE	p_error Mean
0.714037	42.38036	9.785874	23.09059

Source	DF	Type I SS	Mean Square	F Value	Pr > F
mfg	1	37713.88275	37713.88275	393.82	<.0001
plate(mfg)	6	5682.93981	947.15664	9.89	<.0001
strategy	3	803.05581	267.68527	2.80	0.0415
algorithm	1	94.46625	94.46625	0.99	0.3219
mfg*strategy	3	359.35550	119.78517	1.25	0.2926
mfg*algorithm	1	181.05384	181.05384	1.89	0.1707
plate*strategy(mfg)	18	692.18393	38.45466	0.40	0.9864
strategy*algorithm	3	71.98487	23.99496	0.25	0.8609
plate*algorithm(mfg)	6	129.72066	21.62011	0.23	0.9680
mfg*strateg*algorith	3	57.32753	19.10918	0.20	0.8966
plat*stra*algor(mfg)	18	124.46533	6.91474	0.07	1.0000

Source	DF	Type III SS	Mean Square	F Value	Pr > F
mfg	1	37713.88275	37713.88275	393.82	<.0001
plate(mfg)	6	5682.93981	947.15664	9.89	<.0001
strategy	3	803.05581	267.68527	2.80	0.0415
algorithm	1	94.46625	94.46625	0.99	0.3219
mfg*strategy	3	359.35550	119.78517	1.25	0.2926
mfg*algorithm	1	181.05384	181.05384	1.89	0.1707
plate*strategy(mfg)	18	692.18393	38.45466	0.40	0.9864
strategy*algorithm	3	71.98487	23.99496	0.25	0.8609
plate*algorithm(mfg)	6	129.72066	21.62011	0.23	0.9680
mfg*strateg*algorith	3	57.32753	19.10918	0.20	0.8966
plat*stra*algor(mfg)	18	124.46533	6.91474	0.07	1.0000

Tests of Hypotheses Using the Type III MS for plate(mfg) as an Error Term

Source	DF	Type III SS	Mean Square	F Value	Pr > F
mfg	1	37713.88275	37713.88275	39.82	0.0007

Tests of Hypotheses Using the Type III MS for plate*strategy(mfg) as an Error Term

strategy	3	803.0558105	267.68527	6.96	0.0026
mfg*strategy	3	359.3554980	119.785166	3.11	0.0521

Tests of Hypotheses Using the Type III MS for plate*algorithm(mfg) as an Error Term

algorithm	1	94.4662504	94.4662504	4.37	0.0816
mfg*algorithm	1	181.0538441	181.0538441	8.37	0.0276

Tests of Hypotheses Using the Type III MS for plat*stra*algor(mfg) as an Error Term

strategy*algorithm	3	71.98487305	23.99495768	3.47	0.0379
mfg*strateg*algorith	3	57.32752930	19.10917643	2.76	0.0720

Tukey's Studentized Range (HSD) Test for p_error

Alpha	0.05
Error Degrees of Freedom	192
Error Mean Square	95.76334
Critical Value of Studentized Range	2.78939
Minimum Significant Difference	2.4127

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	mfg
A	35.228	128	end
B	10.953	128	face

The GLM Procedure

Level of Plate	Level of mfg	N	point Mean	point Std Dev	p_error Mean	p_error Std Dev
al10	end	32	16.7187500	6.69730025	31.4218750	8.9609812
al4	end	32	19.4687500	5.51235782	39.9093750	15.1186994
al5	end	32	19.3750000	6.71901490	29.3375000	14.2581669
al7	end	32	21.5937500	7.41341145	40.2437500	5.9050518
ci11	face	32	22.0937500	7.05443981	6.3996875	4.4574906
ci2	face	32	19.2500000	5.63972088	18.4156250	8.5416667
ci5	face	32	20.0000000	6.72501349	8.9156250	4.3830398
ci9	face	32	17.4062500	5.30890533	10.0812500	3.8883978

Tukey's Studentized Range (HSD) Test for point

Alpha	0.05
Error Degrees of Freedom	192
Error Mean Square	10.15495
Critical Value of Studentized Range	3.66520
Minimum Significant Difference	1.46

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	strategy
A	27.4219	64	rs0bm5
B	19.6875	64	rs0bm3
B	18.9219	64	rs1bm1
C	11.9219	64	rs0bm1

Tukey's Studentized Range (HSD) Test for p_error

Alpha	0.05
Error Degrees of Freedom	192
Error Mean Square	95.76334

Critical Value of Studentized Range 3.66520
Minimum Significant Difference 4.4834
Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	strategy
A	24.906	64	rs0bm1
B A	24.155	64	rs0bm3
B A	23.060	64	rs0bm5
B	20.242	64	rs1bm1

Tukey's Studentized Range (HSD) Test for point

Alpha	0.05
Error Degrees of Freedom	192
Error Mean Square	10.15495
Critical Value of Studentized Range	2.78939
Minimum Significant Difference	0.7857

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	algorithm
A	20.2891	128	ts
B	18.6875	128	hs

The GLM procedure

Level of	Level of		point		p_error	
mfg	strategy	N	Mean	Std Dev	Mean	Std Dev
end	rs0bm1	32	11.8750000	2.79111494	37.8468750	12.5420579
end	rs0bm3	32	19.5625000	3.76689741	37.3625000	12.5077718
end	rs0bm5	32	27.7187500	4.28978062	35.2781250	12.9617983
end	rs1bm1	32	18.0000000	3.92674863	30.4250000	11.3599182
face	rs0bm1	32	11.9687500	2.11727724	11.9659375	8.0596025
face	rs0bm3	32	19.8125000	3.18704535	10.9471875	7.9145453
face	rs0bm5	32	27.1250000	4.47754253	10.8409375	7.2008971
face	rs1bm1	32	19.8437500	3.64655715	10.0581250	5.4560989

Level of	Level of		point		p_error	
mfg	algorithm	N	Mean	Std Dev	Mean	Std Dev
end	hs	64	18.8437500	6.49351019	36.6765625	12.4528793
end	ts	64	19.7343750	7.05588168	33.7796875	12.5989692
face	hs	64	18.5312500	5.57764458	10.7195312	7.1214443
face	ts	64	20.8437500	6.94014947	11.1865625	7.2840692

Level of	Level of		point		p_error	
strategy	algorithm	N	Mean	Std Dev	Mean	Std Dev
rs0bm1	hs	32	11.5312500	2.28578179	25.2578125	17.1317245

rs0bm1	ts	32	12.3125000	2.59574714	24.5550000	16.5597265
rs0bm3	hs	32	19.0312500	3.35515372	24.6578125	17.5515421
rs0bm3	ts	32	20.3437500	3.49755099	23.6518750	16.4512667
rs0bm5	hs	32	26.0312500	3.39339372	24.5484375	17.3948337
rs0bm5	ts	32	28.8125000	4.80884602	21.5706250	14.8635167
rs1bm1	hs	32	18.1562500	3.47441687	20.3281250	13.9159744
rs1bm1	ts	32	19.6875000	4.14602414	20.1550000	13.3881373

Level of	Level of	Level of		----------point----------		----------p_error----------	
mfg	strategy	algorithm	N	Mean	Std Dev	Mean	Std Dev
end	rs0bm1	hs	16	11.6250000	2.60448331	38.5125000	12.8127996
end	rs0bm1	ts	16	12.1250000	3.03040151	37.1812500	12.6483316
end	rs0bm3	hs	16	19.4375000	4.08197256	38.5625000	12.8205499
end	rs0bm3	ts	16	19.6875000	3.55375388	36.1625000	12.4851846
end	rs0bm5	hs	16	26.8750000	3.63088603	38.3750000	12.5562999
end	rs0bm5	ts	16	28.5625000	4.83002761	32.1812500	13.0037030
end	rs1bm1	hs	16	17.4375000	3.68725282	31.2562500	11.1045918
end	rs1bm1	ts	16	18.5625000	4.19473877	29.5937500	11.9127086
face	rs0bm1	hs	16	11.4375000	1.99895806	12.0031250	8.2214449
face	rs0bm1	ts	16	12.5000000	2.16024690	11.9287500	8.1639303
face	rs0bm3	hs	16	18.6250000	2.50000000	10.7531250	7.7346292
face	rs0bm3	ts	16	21.0000000	3.42539535	11.1412500	8.3397377
face	rs0bm5	hs	16	25.1875000	3.01592993	10.7218750	7.7352649
face	rs0bm5	ts	16	29.0625000	4.93246051	10.9600000	6.8773638
face	rs1bm1	hs	16	18.8750000	3.20156212	9.4000000	4.7048911
face	rs1bm1	ts	16	20.8125000	3.90245649	10.7162500	6.2018243

Dependent Variable: point

Obs	Residual	Obs	Residual	Obs	Residual	Obs	Residual
1	0.75	2	-1.25	3	1.75	4	-1.25
5	0.00	6	-1.00	7	2.00	8	-1.00
9	1.75	10	-2.25	11	-0.25	12	0.75
13	-0.75	14	-1.75	15	1.25	16	1.25
17	0.25	18	-0.75	19	1.25	20	-0.75
21	-5.75	22	11.25	23	-0.75	24	-4.75
25	-2.50	26	-6.50	27	5.50	28	3.50
29	-0.50	30	1.50	31	1.50	32	-2.50
33	-1.75	34	3.25	35	0.25	36	-1.75
37	-0.50	38	3.50	39	-0.50	40	-2.50
41	-1.75	42	5.25	43	0.25	44	-3.75
45	-2.75	46	3.25	47	-0.75	48	0.25
49	1.50	50	-0.50	51	0.50	52	-1.50
53	2.00	54	-3.00	55	3.00	56	-2.00
57	0.25	58	-1.75	59	4.25	60	-2.75
61	1.00	62	2.00	63	-1.00	64	-2.00

65	-3.00	66	2.00	67	4.00	68	-3.00
69	-2.25	70	-1.25	71	6.75	72	-3.25
73	-1.00	74	-3.00	75	8.00	76	-4.00
77	-6.00	78	10.00	79	1.00	80	-5.00
81	-3.50	82	2.50	83	2.50	84	-1.50
85	-3.00	86	1.00	87	3.00	88	-1.00
89	-2.50	90	1.50	91	2.50	92	-1.50
93	-5.50	94	8.50	95	-0.50	96	-2.50
97	0.50	98	1.50	99	0.50	100	-2.50
101	-1.00	102	2.00	103	2.00	104	-3.00
105	-0.25	106	4.75	107	-0.25	108	-4.25
109	-1.75	110	2.25	111	-1.75	112	1.25
113	2.25	114	1.25	115	-0.75	116	-2.75
117	2.00	118	2.00	119	-1.00	120	-3.00
121	1.25	122	3.25	123	0.25	124	-4.75
125	1.25	126	2.25	127	-2.75	128	-0.75
129	-1.25	130	0.75	131	0.75	132	-0.25
133	-0.25	134	0.75	135	-1.25	136	0.75
137	-0.00	138	-0.00	139	-2.00	140	2.00
141	0.00	142	0.00	143	-2.00	144	2.00
145	-2.50	146	-0.50	147	1.50	148	1.50
149	-2.50	150	0.50	151	-1.50	152	3.50
153	-0.50	154	-2.50	155	0.50	156	2.50
157	-2.00	158	-1.00	159	-1.00	160	4.00
161	0.75	162	-0.25	163	-2.25	164	1.75
165	1.00	166	-1.00	167	-1.00	168	1.00
169	-1.50	170	-1.50	171	-0.50	172	3.50
173	2.25	174	-2.75	175	-3.75	176	4.25
177	0.00	178	1.00	179	0.00	180	-1.00
181	-1.50	182	0.50	183	1.50	184	-0.50
185	-0.75	186	0.25	187	3.25	188	-2.75
189	1.50	190	-0.50	191	-2.50	192	1.50
193	-1.25	194	-2.25	195	1.75	196	1.75
197	-0.75	198	-3.75	199	4.25	200	0.25
201	0.00	202	-6.00	203	7.00	204	-1.00
205	0.00	206	-3.00	207	2.00	208	1.00
209	-1.25	210	-2.25	211	1.75	212	1.75
213	0.50	214	-2.50	215	1.50	216	0.50
217	-0.50	218	-1.50	219	1.50	220	0.50
221	-0.50	222	-3.50	223	4.50	224	-0.50
225	0.75	226	2.75	227	0.75	228	-4.25
229	0.00	230	1.00	231	4.00	232	-5.00
233	-0.75	234	-0.75	235	7.25	236	-5.75
237	0.50	238	5.50	239	-0.50	240	-5.50
241	1.00	242	1.00	243	2.00	244	-4.00
245	1.00	246	-1.00	247	4.00	248	-4.00

249	1.00	250	-2.00	251	6.00	252	-5.00
253	-0.50	254	-0.50	255	4.50	256	-3.50

The UNIVARIATE Procedure
Variable: resid

Moments

N	256	Sum Weights	256
Mean	0	Sum Observations	0
Std Deviation	2.76515432	Variance	7.64607843
Skewness	0.62721531	Kurtosis	1.58732994
Uncorrected SS	1949.75	Corrected SS	1949.75
Coeff Variation	.	Std Error Mean	0.17282215

Basic Statistical Measures

Location		Variability	
Mean	0.00000	Std Deviation	2.76515
Median	-0.25000	Variance	7.64608
Mode	-1.00000	Range	17.75000
		Interquartile Range	3.25000

Tests for Location: Mu0=0

Test	-Statistic-		-----p Value------
Student's t	t	0	Pr > |t| 1.0000
Sign	M	-7	Pr >= |M| 0.4091
Signed Rank	S	-655.5	Pr >= |S| 0.5631

Tests for Normality

Test	--Statistic---		-----p Value------	
Shapiro-Wilk	W	0.971725	Pr < W	<0.0001
Kolmogorov-Smirnov	D	0.078502	Pr > D	<0.0100
Cramer-von Mises	W-Sq	0.20386	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	1.367397	Pr > A-Sq	<0.0050

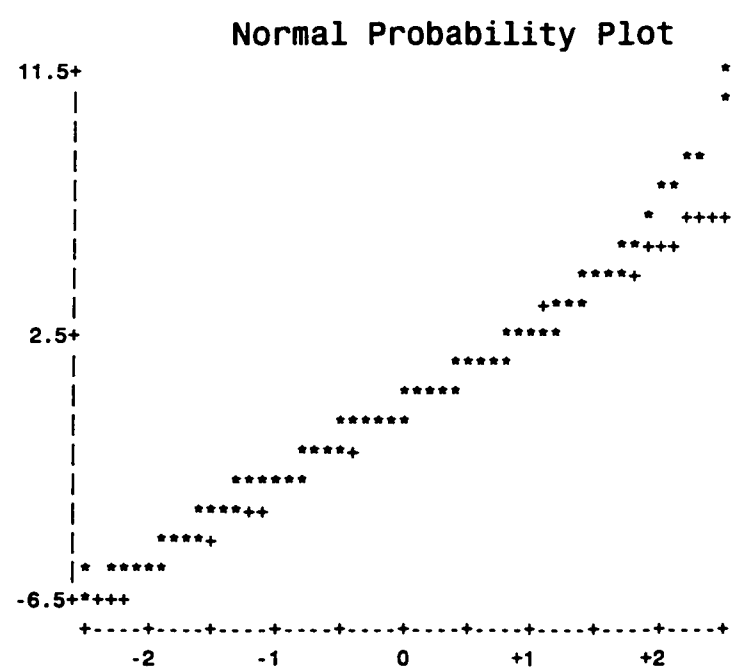
Quantiles (Definition 5)

Quantile	Estimate
100% Max	11.25
99%	8.50
95%	4.50
90%	3.25
75% Q3	1.50
50% Median	-0.25
25% Q1	-1.75
10%	-3.00
5%	-4.25
1%	-6.00
0% Min	-6.50

Extreme Observations

----Lowest----		----Highest----	
Value	Obs	Value	Obs
-6.50	26	7.25	235
-6.00	202	8.00	75
-6.00	77	8.50	94
-5.75	236	10.00	78
-5.75	21	11.25	22

```
Stem Leaf                                          #          Boxplot
  11 2                                             1             0
  10 0                                             1             0
   9
   8 05                                            2             0
   7 02                                            2             0
   6 08                                            2             0
   5 255                                           3             |
   4 0000222558                                   10             |
   3 0022225555                                   10             |
   2 000000000000222255558                        21             |
   1 00000000000022222225555555555558888888      38          +-----+
   0 00000000002222222255555555555888888888      37          |  +  |
  -0 88888888888555555555555555222222           33          *-----*
  -1 88888885555555522222220000000000000        36          +-----+
  -2 88888855555555555222222000000                28             |
  -3 888555200000000                              15             |
  -4 8822000                                       7             |
  -5 8855000                                       7             |
  -6 500                                           3             |
     ----+----+----+----+----+----+----+---
```



Normal Probability Plot

```
11.5+                                                    *
    |                                                    *
    |
    |                                                 **
    |                                                **
    |                                              *  ++++
    |                                            **+++
    |                                         ****+
    |                                      +***
 2.5+                                   *****
    |                              *****
    |                          *****
    |                      ******
    |                   ****+
    |              ******
    |          ****++
    |       ****+
    |* *****
-6.5+*+++
     +----+----+----+----+----+----+----+----+----+----+
         -2        -1         0        +1        +2
```