INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
OPTIMIZATION OF CASTING RIGGING DESIGN
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark "✓".

1. Glossy photographs or pages ______
2. Colored illustrations, paper or print ______
3. Photographs with dark background ______
4. Illustrations are poor copy ______
5. Pages with black marks, not original copy ______
6. Print shows through as there is text on both sides of page ______
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements ______
9. Tightly bound copy with print lost in spine ______
10. Computer printout pages with indistinct print ______
11. Page(s) _________ lacking when material received, and not available from school or author.
12. Page(s) _________ seem to be missing in numbering only as text follows.
13. Two pages numbered _________. Text follows.
14. Curling and wrinkled pages ______
15. Other _______________________________________________________________________

University Microfilms International
OPTIMIZATION OF CASTING RIGGING DESIGN

DISSERTATION

Presented in Partial Fulfillment of the Requirements
for the Degree Doctor of Philosophy

by

Ronald Lee Lewis, B.S., M.S.

The Ohio State University

1983

Reading Committee:

Dr. Clarence H. Martin
Dr. Carroll E. Mobley, Jr.
Dr. Clark A. Mount-Campbell

Approved By:

[Signature]

Adviser

Department of Industrial and Systems Engineering
ACKNOWLEDGMENTS

I would like to dedicate this work to my wife, Jean, whose help in typing this dissertation and whose encouragement has been instrumental in helping complete this.

I would also like to thank my parents for giving me the standards and work ethic needed to complete this research. It was their guidance over the last twenty-nine years that gave me the perseverance to see it through.

I would also like to give special recognition to my adviser, Dr. Clark Mount-Campbell, for working closely with me throughout this research and also through the course of my entire graduate studies program at Ohio State. He has exhibited unlimited patience in smoothing out the "rough edges" of this work.

Finally, I would like to give my thanks to Drs. Carroll Mobley and Clarence Martin for being a part of both my general exam committee and dissertation reading committee.
VITA

February 26, 1954 . . . . Born - Columbus, Ohio

1976 . . . . . . . . . . B.S.I.S.E., The Ohio State University, Columbus, Ohio

1978 . . . . . . . . . . Graduate Teaching Associate, Department of Industrial and Systems Engineering, The Ohio State University, Columbus, Ohio

1980 . . . . . . . . . . M.S., The Ohio State University, Columbus, Ohio

FIELDS OF STUDY

Major Field: Industrial and Systems Engineering

Manufacturing Engineering

Operations Research applied to Manufacturing Related Problems
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>VITA</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTERS</td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. GENERAL FOUNDRY AND SOLIDIFICATION DISCUSSION</td>
<td>6</td>
</tr>
<tr>
<td>General Foundry Operation</td>
<td>6</td>
</tr>
<tr>
<td>Pattern Making Design And Testing</td>
<td>6</td>
</tr>
<tr>
<td>Core Making</td>
<td>8</td>
</tr>
<tr>
<td>Molding</td>
<td>8</td>
</tr>
<tr>
<td>Pouring</td>
<td>9</td>
</tr>
<tr>
<td>Cleaning, Inspection, And Repair</td>
<td>9</td>
</tr>
<tr>
<td>General Rigging Discussion</td>
<td>10</td>
</tr>
<tr>
<td>General Solidification Discussion</td>
<td>13</td>
</tr>
<tr>
<td>Constant Temperature: Pure Metals</td>
<td>18</td>
</tr>
<tr>
<td>Solid Solutions: Long and Short</td>
<td>20</td>
</tr>
<tr>
<td>Freezing Range Alloys</td>
<td>34</td>
</tr>
<tr>
<td>Shrinkage Defect Elimination</td>
<td>35</td>
</tr>
<tr>
<td>Riser Design</td>
<td>37</td>
</tr>
<tr>
<td>Chill Utilization</td>
<td>38</td>
</tr>
<tr>
<td>Padding Of Castings</td>
<td>40</td>
</tr>
<tr>
<td>Research Goals</td>
<td></td>
</tr>
<tr>
<td>III. CURRENT TECHNIQUES IN RIGGING DESIGN</td>
<td>45</td>
</tr>
<tr>
<td>Current Techniques Of Riser Design</td>
<td>45</td>
</tr>
<tr>
<td>Chvorinov's Method</td>
<td>46</td>
</tr>
<tr>
<td>Adams and Taylor Method</td>
<td>47</td>
</tr>
<tr>
<td>Caine's Method</td>
<td>48</td>
</tr>
<tr>
<td>Bishop's Shape Factor</td>
<td>50</td>
</tr>
<tr>
<td>Ruddle's Modulus Factor</td>
<td>50</td>
</tr>
<tr>
<td>Merchant's Method</td>
<td>54</td>
</tr>
<tr>
<td>Pipe Discussion</td>
<td>56</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>IV. SOLIDIFICATION MODELING</td>
<td>71</td>
</tr>
<tr>
<td>Methods Of Quantifying Solidification</td>
<td>71</td>
</tr>
<tr>
<td>Pour Out Method</td>
<td>72</td>
</tr>
<tr>
<td>Temperature Measurement Technique</td>
<td>73</td>
</tr>
<tr>
<td>Analogue Methods</td>
<td>74</td>
</tr>
<tr>
<td>Mathematical Analysis</td>
<td>74</td>
</tr>
<tr>
<td>Approximation Techniques</td>
<td>78</td>
</tr>
<tr>
<td>Computer Simulation Of Solidification</td>
<td>82</td>
</tr>
<tr>
<td>Derivation Of Nodal Temperatures</td>
<td>87</td>
</tr>
<tr>
<td>Solidification Model Results</td>
<td>92</td>
</tr>
<tr>
<td>Shrinkage Cavity Size</td>
<td>92</td>
</tr>
<tr>
<td>Latent Heat Of Fusion</td>
<td>94</td>
</tr>
<tr>
<td>Rigging Effects</td>
<td>97</td>
</tr>
<tr>
<td>Gradient Results</td>
<td>101</td>
</tr>
<tr>
<td>V. PROBLEM ATTACK</td>
<td>108</td>
</tr>
<tr>
<td>Problem Formulation</td>
<td>108</td>
</tr>
<tr>
<td>Shrinkage Constraint</td>
<td>123</td>
</tr>
<tr>
<td>Bounding Constraint</td>
<td>127</td>
</tr>
<tr>
<td>Existence Constraint</td>
<td>128</td>
</tr>
<tr>
<td>Non-Negativity Constraints</td>
<td>129</td>
</tr>
<tr>
<td>Solution Methodology</td>
<td>129</td>
</tr>
<tr>
<td>Optimal Selection Of Combinations</td>
<td>132</td>
</tr>
<tr>
<td>Subproblem Optimization</td>
<td>144</td>
</tr>
<tr>
<td>Review Of Chapter V</td>
<td>178</td>
</tr>
<tr>
<td>VI. RESULTS</td>
<td>182</td>
</tr>
<tr>
<td>General Discussion</td>
<td>184</td>
</tr>
<tr>
<td>Optimization Algorithm Efficiency</td>
<td></td>
</tr>
<tr>
<td>Discussion</td>
<td>188</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>203</td>
</tr>
<tr>
<td>Steepest Descent Vs. Davidon-Fletcher-Powell</td>
<td>206</td>
</tr>
<tr>
<td>Penalty Discussion</td>
<td>207</td>
</tr>
<tr>
<td>Convexity Problems Related To Golden Section</td>
<td>207</td>
</tr>
<tr>
<td>Dual Adequate Solution Procedure</td>
<td>212</td>
</tr>
</tbody>
</table>
VII. FUTURE RESEARCH

Formulation of Padding .................................. 213
Solidification Model-Gradient Update .................. 215
Gating Design .............................................. 216
Three-Dimensional Extension ............................. 217
Non-Simulation Solidification Models .................... 219
Exothermic Riser Aids .................................... 220
Alloy Considerations ...................................... 221
Isolated Shrinkage Cavities ................................ 222
Defect Analysis - Metal Penetration ...................... 222
Defect Analysis - Hot Tearing ............................. 223
Grain Size .................................................. 223

BIBLIOGRAPHY ............................................. 225

APPENDICES

A. EXAMPLE PROBLEM - BENDERS' DECOMPOSITION 229
B. EXAMPLE PROBLEM - PENALTY METHOD ................. 249
C. EXAMPLE PROBLEM - DAVIDON-FLETCHER-POWELL .... 252
D. EXAMPLE PROBLEM - GOLDEN SECTION LINE SEARCH .... 255
E. COMPUTER PROGRAM LISTINGS .......................... 259
<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Complete Problem Formulation</td>
</tr>
<tr>
<td>2</td>
<td>Kuhn-Tucker Conditions</td>
</tr>
<tr>
<td>3</td>
<td>Data Utilized In Efficiency Experimental Design</td>
</tr>
<tr>
<td>4</td>
<td>Penalty Results</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Foundry Process Flow</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Mold-Casting Diagram</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Solidification Of Alloys That Form A Thin Skin</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Gradient And Growth Direction</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>Columnar Grain Growth</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>Solidification Toward Riser</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>Equilibrium Cooling Of An Isomorphous Alloy</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>Temperature Gradients</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>Dendrite Tree</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>Gradient Vs. Interface Morphology</td>
<td>31</td>
</tr>
<tr>
<td>11</td>
<td>Solidification Waves</td>
<td>33</td>
</tr>
<tr>
<td>12</td>
<td>Riser Feeding</td>
<td>36</td>
</tr>
<tr>
<td>13</td>
<td>Padding Effect</td>
<td>39</td>
</tr>
<tr>
<td>14</td>
<td>Alternative Rigging Example</td>
<td>41</td>
</tr>
<tr>
<td>15</td>
<td>Caine's Relationship</td>
<td>49</td>
</tr>
<tr>
<td>16</td>
<td>Relation Of Casting Shape Factor To Minimum Effective Riser Volume To Casting Volume</td>
<td>51</td>
</tr>
<tr>
<td>17</td>
<td>Merchant's Linear Relationship</td>
<td>55</td>
</tr>
<tr>
<td>18</td>
<td>Cross Section Of Riser Pipe</td>
<td>57</td>
</tr>
<tr>
<td>19</td>
<td>End Effect</td>
<td>60</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>20 Creese' Optimization Design</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>21 Conductivity Vs. Solidification</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>22 Example Of Casting And Rigging Design Not Covered By Formulation</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>23 Shrinkage Size Flow Chart</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>24 Time-Temperature Graph For A 2&quot; x 8&quot; Low Carbon Steel Plate</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>25 Isochronal Solidification Profile Of A 2&quot; x 8&quot; Low Carbon Steel Plate</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>26 Riser-Chill Dimensions Effect Upon Shrinkage Cavities Via Solidification Simulation</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>27 Example Set Of Risers And Chills Constructed By Foundry Engineer For A Given Casting Geometry</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>28 Riser And Chill Height Effects On Cost</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>29 Cost Function With Respect To Position</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>30 Trade-Off Of Risers And Chills</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>31 Position Effect With Respect To Shrinkage Defects</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>32 General Optimization Procedure Flow Chart</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>33 Fathoming Of Infeasible Combinations</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>34 Benders' Flow Chart</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>35 Enumeration Tree</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>36 Effect Of Number Of Riser-Casting Nodes On Solidification Simulation Time</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>37 Penalty Method Flow Chart</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>38 Comparison Of Steepest Descent And Newton's Method Search Direction</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>39</td>
<td>Davidson-Fletcher-Powell Flow Chart</td>
<td>162</td>
</tr>
<tr>
<td>40</td>
<td>Golden Section</td>
<td>169</td>
</tr>
<tr>
<td>41</td>
<td>Kuhn-Tucker Algorithm</td>
<td>179</td>
</tr>
<tr>
<td>42</td>
<td>General Informational Outputs</td>
<td>186</td>
</tr>
<tr>
<td>43</td>
<td>Golden Section Sample Output</td>
<td>189</td>
</tr>
<tr>
<td>44</td>
<td>Benders' Decomposition Sample Output</td>
<td>190</td>
</tr>
<tr>
<td>45</td>
<td>Efficiency Results For Optimization Algorithm</td>
<td>200</td>
</tr>
<tr>
<td>46</td>
<td>Output From SAS - Optimization Efficiency</td>
<td>202</td>
</tr>
<tr>
<td>47</td>
<td>Sensitivity Analysis Of Dual Variables Corresponding To Upper Bound Of Riser Height</td>
<td>205</td>
</tr>
<tr>
<td>48</td>
<td>Output From SAS - Penalty Experiment</td>
<td>209</td>
</tr>
<tr>
<td>49</td>
<td>Convexity Problem</td>
<td>211</td>
</tr>
<tr>
<td>50</td>
<td>Padding Formulation</td>
<td>214</td>
</tr>
<tr>
<td>51</td>
<td>Three Dimension Model Extension</td>
<td>218</td>
</tr>
</tbody>
</table>
Chapter I
INTRODUCTION

Founding, as described by Webster, is "the art of melting and forming objects by pouring molten metal into sand molds." Archeological findings indicate that metal craftsmen around 3,000 B.C. cast bronze tools and weapons utilizing permanent stone molds. The earliest known operating foundry in America is believed to have been established in Lynn, Massachusetts in 1643 [1].

By 1995 shipments of U.S. produced metal castings are projected to reach nineteen million tons valued at over seventy-three billion dollars [2]. In comparison, more than fifteen million tons of castings valued at sixteen and one-half billion dollars were shipped in 1980. This projection represents an annual growth of more than ten percent in value and one and four-tenths in tonnage during the next fifteen years. If this projection is accurate, technology must be prepared to meet it.

One of the most noteworthy men involved in the history of the foundry industry was Charles Willers Briggs [3]. Described by his contemporaries as a genius, he authored a number of works involving a wide range of foundry technological topics. Among these were: Fundamentals of Steel
Foundry Sands, Fundamentals Of Risering Steel Castings, and Core Sands and Binders. As he approached retirement, Mr. Briggs looked to the future and forecast a number of avenues the industry should pursue if it were to advance and achieve maximum growth. Some of these were: rapid pattern formation, computerized steel melting and chemistry control, more advanced scheduling techniques, and rapid determination of casting rigging requirements (determination of size and position of risers, chills, gates, etc.). This dissertation will address a portion of the latter of these—determination of both size and location of risers and chills.

The research proposals of Charles Briggs have been stymied throughout the seventies by the impact of high inflation, labor demands, and environmental standards of both the Occupational Safety and Health Administration (OSHA) and the Environmental Protection Agency (EPA). Throughout this period, and continuing today, foundry closings have shown a sharp increase. Perhaps more importantly, the surviving foundries have foresaken capital improvements as inflation has driven up the cost of such expansion. Also, the meeting of OSHA and EPA standards has drawn off much of the capital which might have met the inflated costs. A Cast Metals Federation study of capital expenditures between 1972 and 1976 found expenditures
for compliance of environmental standards consumed thirty-five percent of available capital for large foundries (those exceeding five million dollars in annual sales) [3]. Because of the environmental expenditures, and due to the author's belief that the foundry industry has always been slow to change, technological improvements and especially the use of operations research techniques have been slow in being utilized. It is the author's hope that this dissertation will further advance the adoption of operations research techniques in the solving of foundry related problems, especially the problem of rigging design.

The research topic addressed by this dissertation is optimal rigging design. To this end, the problem was mathematically formulated. Decision variables represented the number of risers and chills to utilize along with their respective positions and sizes. The objective was to minimize cost and yet produce a shrinkage defect-free casting. To evaluate the shrinkage potential of a given casting, with a given riser-chill dimensioning, an explicit finite difference solidification simulation of the casting was performed.

Motivation for this research came from the author's experience as both a production supervisor and foundry engineer in a steel foundry and, consequently, from finding frustration in trying to produce shrinkage defect-free castings which had not been properly rigged.
A description of the general foundry operation and a description of the mode of casting solidification is offered in Chapter II. Methods of shrinkage defect elimination and a description of the problem can also be found in Chapter II. Chapter III contains a review of the current techniques of rigging design. As previously mentioned, an explicit finite difference technique was utilized to simulate the casting solidification, and in turn determine the extent of the shrinkage defect. This is described in Chapter IV along with a discussion of simulation results. Chapter V presents the mathematical formulation of the problem and a description of the technique utilized to solve it. Research results including discussions of sensitivity analysis, feasibility region, convexity, etc., are presented in Chapter VI. Chapter VII is a discussion of future research topics. Included are extensions of the formulation to a three-dimensional design, alternative formulations, model extensions and speedier casting solidification simulation. A number of appendices are also included. Appendix A presents a sample problem demonstrating Benders' Decomposition. Appendix B contains an example problem utilizing the use of a penalty to transform a constrained optimization problem to one which is unconstrained. Appendix C presents an example problem utilizing the Davidon-Fletcher-Powell Method for solving nonlinear unconstrained optimization problems. A demonstration of
the algorithm utilized to determine the dual variables satisfying the Kuhn-Tucker conditions is also included in Appendix C. Appendix D contains an example of the line search technique employed— that of the golden section. Finally, the FORTRAN programs used are described and a listing is given in Appendix E.
Chapter II

GENERAL FOUNDRY AND SOLIDIFICATION DISCUSSION

To lay the groundwork for future discussion, five foundry related topics will be discussed in this chapter. They are: (1) a description of the general foundry operation, (2) a description of general rigging design, (3) a description of the solidification process, (4) a discussion of currently utilized techniques to eliminate shrinkage defects, and (5) a description of the problem and its importance with respect to cost effectiveness.

General Foundry Operation

Figure 1 demonstrates the process flow of a foundry. One may note the cost percentage shown on the left and the scope of the process addressed as research in this dissertation. The following is a description of the major steps of the process [4].

Pattern Design And Testing

Patterns are required to produce molds. The mold is made by packing some readily-formed plastic material, such as molding sand, around the pattern. When the pattern is withdrawn, its imprint provides the mold cavity, which is later filled with liquid metal to become a casting. A
Figure 1. Foundry Process Flow
pattern is simply a near replica of the exterior of the casting. Included on most patterns is the casting geometry and rigging (risers, chills, prints, etc.). If the casting is hollow, additional patterns, referred to as core boxes, are used to form the sand that is used to create these cavities. Also included in pattern design is the rigging design and sampling requirements. The rigging design is the determination of the number, size, and location of chills, gates, risers, etc. (these will be defined later). This again, is the area of interest of this research. A sampling period is a trial run to determine if the proposed pattern produces an acceptable quality casting in an efficient and moldable manner.

Core Making

Cores are forms, usually made of sand, that are placed into a mold to form interior casting surfaces. Thus, the void between the core and the casting cavity surface is what forms the casting. Cores are generally made separately from molds in a different area of the foundry.

Molding

Molding is a set of operations necessary to prepare a mold for receiving molten metal. Molding usually involves placing a plastic material around the pattern, withdrawing the pattern to leave the mold cavity, finishing the mold,
setting cores, and then closing the mold. After pouring the casting is allowed to cool for a specified period of time, depending on its size, alloy type and desired physical characteristics. This time period is termed hold time.

Pouring

Preparation of molten metal for casting is referred to by the general term melting. Melting is usually performed in a separate area of the foundry. The molten metal is transferred to a molding area where the molds are then poured.

Cleaning, Inspection, And Repair

Cleaning refers to all operations necessary to remove sand scale and excess metal from the casting. The casting is separated from the molding sand and transported to the finishing department. This is termed shake-out. Removal of the risers and gating system then takes place. Surface defects such as fins, metal penetration (metal penetrating into the molding sand) and other surface blemishes are removed by the grinding operations. Inspection of the casting for defects and general quality then takes place. Defective castings that may be salvaged are welded, ground, etc. If all quality standards have been satisfied, the casting is ready to be shipped.
Finally, one should note the high cost associated with inspection and repair. Defective castings can usually be salvaged, but usually at high cost. In many foundries between fifty to seventy percent of the cost of producing a casting is spent in the inspection and repair phase. The author has witnessed 1,200 pounds of weld rod used to repair an 18,000 pound casting due to large internal shrinkage cavities. This should imply that prudent quality and process controls applied in the foundry, along with proper rigging design, are two of the best approaches to cost reduction.

General Rigging Discussion

The rigging of a casting is the determination of all characteristics of the riser-gating system. This includes determination of riser size and position relative to the casting, chill size and placement, gating dimensions and position, and all other directional solidification techniques utilized, such as padding, exothermic risering aids, etc. This riser-gating system should attempt to accomplish the following:

1. Fill the mold cavity with a minimum amount of turbulence so as to prevent erosion of the mold wall.

2. Provide directional solidification so as to prevent unacceptable shrinkage cavities (i.e. shrinkage defects).
3. Create a design that is compatible with molding, pouring, and cleaning operations so as to minimize effort and cost.

The elements of a basic riser-gating design are shown in Figure 2. Definitions of each element are:

1. Casting: that portion of the riser-gating system which is saleable to the customer.

2. Downspreu: channel through which liquid metal from the pouring ladle enters the mold cavity, and in turn into the runner system.

3. Runner: channels through which liquid metal passes from the downspreu to the ingates.

4. Ingates: contact between the runner and the casting.

5. Pouring Cup: usually a funnel-shaped cup used as a target for the ladle operator, so as to initiate the pouring.

6. Riser: a reservoir of liquid metal which stays liquid while the casting is freezing and provides feed metal to compensate for the shrinkage occurring while the casting is changing from a liquid to a solid state [4].

7. Top Riser: located on the top of the casting.

8. Side Riser: located on the side of the casting.

9. Open Riser: riser is open to the atmosphere.

10. Blind Riser: an internal riser which does not reach the exterior of the mold.

11. Chill: an object, usually made of metal, placed outside or inside of the mold cavity to induce more rapid cooling at the contact area [4].
Note: Mold Cavity includes both casting cavity and rigging.

Figure 2. Mold-Casting Diagram
12. Sleeves: hollow cylinders formed of gypsum, diatomaceous earth, pearlite, etc., placed around the riser cavities to decrease heat loss and the rate of solidification of the riser. This, in turn, enhances the effectiveness of the riser.

13. Cope: topmost section of a flask, mold, or pattern.

14. Drag: bottom section of a flask, mold, or pattern.

15. Parting Line: dividing line between the cope and drag.

16. Padding: metal added deliberately to the cross section of a casting wall to ensure adequate feeding of a localized area in which a shrinkage cavity may otherwise occur. The padding is removed from the casting prior to its use.

General Solidification Discussion

Solidification occurs through nucleation of minute grains or crystals, which then grow under the influence of crystallographic and thermal conditions. The size and character of the resulting grains is controlled by alloy composition and cooling rate. It is important to note that grain size affects many mechanical and physical properties such as hardness, yield, and ultimate stress, etc. Growth obviously ceases when all available liquid metal has solidified.

The mode of heat removal and the stages of cooling impact the solidification process. Clearly, as soon as alloy enters the mold, heat is extracted from the liquid
alloy. This heat is termed superheat as it is the temperature difference between the melting point of the alloy and the tap temperature at the furnace, which must be removed for solidification to begin. A certain degree of superheat is necessary to facilitate the following: transportation of liquid alloy from the furnace to the pouring area, transportation of the ladle containing the liquid metal from mold to mold, and the ability of liquid alloy to enter the casting cavity without freezing prior to the mold cavity being completely filled. As the superheat is removed and solidification begins, the latent heat of fusion is evolved. Latent heat of fusion is the energy required to melt a crystal (i.e. to randomize the molecules) [4]. When a liquid crystallizes, the substance releases the latent heat of fusion. For solidification to continue, this latent heat of fusion must be transferred to the already solidified portion of the casting, and then in turn to the mold and the mold environment. Obviously the more quickly heat can be removed from the casting, the faster solidification can proceed. Thus, a water cooled solid metal mold would provide a faster rate of solidification than would an insulating green sand mold.

It should be apparent that there are three stages of cooling. These are:
1. Liquid Cooling: dissipation of superheat.

2. Liquid-Solid Cooling: transformation of liquid to solid and, in turn, dissipation of latent heat of fusion.

3. Solid Cooling: cooling from melting point to room temperature.

Throughout all three stages of cooling shrinkage is occurring in most alloys. The alloy contracts as it looses superheat, as it transforms from liquid to solid, and as the solid cools to room temperature.

Obviously, for production of sound (i.e. shrinkage defect-free) and dimensionally correct castings, shrinkage compensation must be provided in all stages of cooling.

Liquid shrinkage is generally compensated for by the use of risers. Also, good pouring practice will include "touching-up" the mold. This entails refilling the mold cavity after it has been initially poured and superheat dissipation has begun. This touch-up of the mold should compensate for a portion of the liquid shrinkage while the risers should account for the remainder.

The liquid-solid shrinkage is compensated for by promoting directional solidification. Directional solidification is the process of controlling the cooling rate by mold design, and utilizing the intrinsic design of the casting, such that freezing begins in those parts of the mold furthest away from the risers and then proceeds toward
the risers. The risers in turn are designed to solidify last so as to supply liquid feed metal throughout solidification to the casting cavity. This process should force most shrinkage cavities into the risers. Various methods of promoting directional solidification are:

1. properly placed risers and gates,
2. strategically placed chills,
3. promotion of directional solidification through casting redesign.

All three of these methods will be discussed in more detail later.

Finally, the solid shrinkage stage is compensated for by an oversized mold cavity. This is what foundrymen term pattern maker's rule. In other words, the mold cavity is produced slightly oversize, with respect to the casting design size. This allows the casting to shrink to the design size.

Rigging design (i.e. determination of position and size of risers and chills) can promote directional solidification and in turn eliminate only those shrinkage defects occurring during the liquid-solid stage of cooling. This dissertation is concerned with rigging design, so only the liquid-solid stage of cooling is considered.

Naturally, to determine a method of promoting directional solidification, a complete understanding of the solidification process is imperative. Other variables
which influence the directionality of the solidification process include:

1. mold material
2. thickness of mold material
3. mold geometry
4. metal thickness
5. metal properties (i.e. thermal conductivity, solidification temperature range, etc.)
6. heat transfer from mold to atmosphere
7. control of grain size by inoculants, vibration, etc.

Because these variables affect the solidification process, the following discussion does not consider them. Only the process itself will be considered. Finally, promotion of directional solidification to eliminate liquid-solid shrinkage defects does not guarantee elimination of other defects, even those associated with rigging problems. Some defects might be:

3. Burn-on: sand imbedded into metal surface.

The chosen design of casting rigging could even promote some of the above defects.
Considering only binary alloy systems, solidification behavior may be divided into four classifications. These are:

1. Those freezing at a constant temperature (pure metals and eutectic alloys).
2. Those freezing over a range of temperature with marked skin formation, termed "solid solutions", such as low carbon steel, brass, aluminum and copper.
3. Those freezing over a range of temperature with an extensive mushy zone, also termed "solid solutions", including medium and high carbon steel, nickel-brass alloys, and magnesium alloys.
4. Those which show an expansion stage during freezing, such as gray cast iron.

A discussion of the first three behavioral classifications follows.

**Constant Temperature: Pure Metals**

When a pure metal is poured into a mold cavity the first metal to contact the mold wall generally freezes with an equiaxed grain structure. The chilling action of the mold wall results in the formation of a thin skin or shell of solid metal surrounding the liquid (see Figure 3). With increasing heat extraction through the solid and, in turn, through the mold, liquid metal freezes onto the already solidified wall and proceeds toward the center of the casting.

This process is retarded, however, after the initial skin formation due to the latent heat of fusion being
Figure 3. Solidification of Alloys That Form a Thin Skin
expelled into the liquid metal. This increase in the liquid metal temperature slows down solidification.

The rate at which the wall thickness grows is dependent upon:

1. The temperature gradient imposed upon it—growth is in a direction opposite the temperature gradient (see Figure 4).

2. Crystallographic orientation—only those grains favorably oriented will grow toward the center of the casting cavity. Those less favorably oriented will be pinched off. This results in a columnar growth structure (see Figure 5).

The ratio of equiaxed grains to columnar grains is:

1. inversely proportional to the effective superheat; and

2. inversely proportional to the critical degree of supercooling necessary for nucleation to occur at a fairly high rate [4].

The liquid-solid interface is smooth due to the metal freezing at a constant temperature. Figure 6 demonstrates both solidification of a pure metal and progressive solidification toward the riser.

Solid Solutions: Long And Short Freezing Range Alloys

The freezing process for solid solutions is more complex than for that of a pure metal. The following three phenomena are responsible for the increase in complexity between the freezing of pure metals and that of alloys:
Figure 4. Gradient And Growth Direction
Figure 5. Columnar Grain Growth
Figure 6. Solidification Toward Riser
1. more than one solid phase could form from a homogeneous melt;

2. solid formed is of a different composition than the liquid from which it came;

3. alloys freeze over a temperature range [4].

Let us examine solid solution freezing by considering equilibrium cooling of an isomorphous alloy (see Figure 7). In this example a binary system is considered with components A and B. The liquid melt is of uniform bulk composition $C_L$. As superheat is dissipated the melt attains temperature $T^0$. This is the initial temperature at which solidification begins. At temperature $T^0$, composition $C_S^0$ is rejected as solid from liquid which has composition $C_L^0$. Later, at temperature $T^1$, solid composition is now at $C_S^1$, with liquid composition now at $C_L^1$. Finally, at temperature $T^F$, solidification is complete with solid composition $C_L$, and the last bit of liquid is at composition $C_L^F$. Using the phase diagram and the "lever rule", both the composition of the solid and of the liquid can be determined at any temperature, along with percentages of both liquid and solid at any temperature.

From the previous discussion, it is important to understand the following facts:

1. Solidification occurred over temperatures ranging from $T^0$ to $T^F$.

2. Composition of solid changed from an initial composition at $C_S^0$ to $C_L$. 

Figure 7. Equilibrium Cooling Of An Isomorphous Alloy
Figure 7 (continued)

Temperature gradient

Temperature

Solid

Liquid

Distance

Liquid-Solid Interface

TEMPERATURE GRADIENT
3. Composition of the liquid changed from an initial composition of $C_L$ to $C^F_L$.

It is assumed under equilibrium conditions that diffusion is sufficient in the solid to obtain a homogeneous solid phase of composition $C_L$, the initial composition, upon complete solidification.

Since solidification occurs over a range of temperatures, it should be recognized that there are three distinct zones occurring simultaneously. These are:

1. a completely solid zone adjacent to the mold wall
2. a mushy zone containing both liquid and solid
3. a completely liquid zone in the thermal center--i.e. the area of the casting which is last to solidify

An important point to consider in alloy solidification is that of grain growth. Again, referring to Figure 7, solute buildup may occur in the liquid phase in front of the interface. For the type of phase diagram considered the increase in solute forces a lower required freezing point (i.e. as the percentage of solute in the liquid state increases, the liquidus temperature decreases). Thus, in front of the interface the solute compositional gradient, in turn, causes a liquidus temperature gradient. This in no way implies that this is the temperature of the liquid in front of the liquid-solid interface. It only implies that this is the temperature required for solidification in front of the interface to continue. Figure 8 demonstrates
Gradients A and B represent planar front solidification. Gradients C and D represent non-planar front solidification, due to the constitutional supercooling exhibited at 1. Gradient D, due to the large amount of supercooling, could even exhibit independent nucleation in front of the interface.

Figure 8. Temperature Gradients
different possibilities of temperature gradients in front of the liquid-solid interface. Every point in front of the liquid-solid interface is at a temperature above that of the liquidus for gradients A and B. This condition is necessary for planar front solidification. That is, if a protuberance would form on the interface, it would be in a superheated environment and thus would be melted back. However, gradients C and D represent an unstable case. The actual liquid temperature in front of the interface is lower than that of the equilibrium liquidus temperature. The term for this is supercooling. Chalmers [7] termed this constitutional supercooling—the word constitutional indicating that the supercooling is a result of a change in composition. Clearly, any protuberance forming on the interface would find itself in a supercooled environment, especially those with a favorable crystallographic orientation. This environment, in turn, would be favorable to the protuberance not being remelted. These protuberances, particularly those associated with gradients C and D (see Figure 8), are created in a tree-like formation. They are termed "dendrites", and are shown in Figure 9.

Finally, if supercooling is sufficient as demonstrated by gradient D (see Figure 8), independent nucleation far from the interface could result. Figure 10 demonstrates the influence of undercooling on interface morphology [4].
Figure 9. Dendrite Tree
Figure 10. Gradient Vs. Interface Morphology
As shown in Figures 8 and 10, if supercooling is sufficient random nucleation can occur in advance of the growing interface. If supercooling continues, each of these nuclei can form as dendrites. This continues until constitutional supercooling ceases because of thermal and compositional adjustments. Many alloys exhibit columnar dendritic grains near the surface and the equiaxed dendrite grains near the thermal center [4]. The ratio of equiaxed to columnar grains is:

1. Inversely proportional to the effective superheat. It has been shown [8] that a low pouring temperature creates more nuclei which survive in the liquid and which drift away from the interface by convection. Assuming these nuclei form equiaxed grains, the lower the pouring temperature the greater the number of equiaxed grains.

2. Inversely proportional to the critical degree of supercooling necessary for nucleation to occur at a fairly high rate.

3. Proportional to the freezing range.

4. Inversely proportional to the slopes of the solidus and liquidus lines [4].

The previous discussion is the major distinction between those alloys of long and short freezing range. One could consider solidification as two waves: one representing the start of freezing and another representing the end of freezing. Within this band is what is termed the mushy zone of the casting (see Figure 11). Clearly,
Distance From Interface

Time

(1) .05-10% Carbon Steel

Distance From Interface

Time

(2) .25-30% Carbon Steel

Distance From Interface

Time

(3) .55-.60% Carbon Steel

Note: The increase in carbon content causes a wider band between the start and end of freezing. In (3) a large mushy zone is evident.

Figure 11. Solidification Waves
as the carbon and solute content of the initial melt rises so does the freezing range, and in turn the potential for undercooling. This increase in undercooling widens the band between the start and end of freezing waves. Some alloys can contain an almost complete mushy zone. Again, this is the distinction between long and short freezing range alloys.

The alloy solidification process, although interesting, is mathematically complex. Due to the author's hands-on experience and also to ease the burden of mathematical complexity, only pure metals and those with short freezing ranges are considered. Unfortunately, the issues of constitutional supercooling and interface morphology are not considered by either the mathematical formulation or the solidification simulation.

**Shrinkage Defect Elimination**

There are basically three different tools the foundry engineer can utilize separately or in conjunction with one another to produce shrinkage defect-free castings. They are risers, chills, and design alterations (usually termed padding). As previously discussed, without the application of any of these, nearly all castings would have shrinkage cavities. Each of these different tools will be discussed in the following paragraphs. The chapter which
follows will address currently utilized methods of riser, chill, and padding design.

Riser Design

The most common approach to attaining directional solidification is that of utilizing risers. As previously discussed, risers are needed to compensate for shrinkage in the liquid and liquid-solid (i.e. solidification) cooling stages of a casting. The amount of feed metal necessary is determined by the specific alloy type and the mechanism of freezing (see Figure 12). As an example, steel has a solidification volumetric shrinkage of approximately four percent. If there were no risers this would be exhibited as one or more shrinkage cavities within the casting. If these cavities were unacceptable to the customer they would be considered shrinkage defects.

A serious problem with respect to shrinkage defects is that visual inspection of a casting may not indicate a shrinkage cavity. Only through careful x-ray examination can one be positive that the defect is eliminated.

The following factors influence riser effectiveness:

1. riser shape
2. riser size
3. riser location on the casting
4. groupings of the castings
Figure 12. Riser Feeding
5. riser connection to the casting
6. utilization of insulators or exothermic compounds

Chill Utilization

The previous discussion considered promoting directional solidification by delaying the freezing process directionally toward the riser. The riser then compensated for the liquid-solid shrinkage. The same end could be achieved in an opposite way—that is, by chilling the area most remote from the liquid metal source.

There are two types of chills: internal and external. External chills are molded and then placed in the walls of the casting cavity. They are strategically located in the cavity so as to increase the freezing rate at that point. Chills can be made of any material, such as steel, copper, brass, or even different types of sand. However, to be considered a chill heat must be removed faster at the area of application than at the non-chilled areas.

Internal chills are placed within the mold cavity at strategic locations so as to promote a faster freezing rate. Clearly, the internal chill must have the same approximate composition as the casting being produced as they will be fused into the casting upon pouring. If internal chills are utilized care must be exercised to ensure the following:

1. the chill must fuse with the casting,
2. the chill must be clean so as to eliminate
gas defects, and

3. the chill must not alter the casting's
mechanical properties.

It should be apparent that chills are an excellent
tool in promoting directional solidification. However,
the drawbacks of chill usage are:

1. The chill must be purchased, or in-house
produced. The chill is yet another
item which must be available during mold-
ing (i.e. if produced in-house, chills
must be scheduled to be produced prior to
the molding run).

2. To lower costs chills must be retrieved
if they are external.

3. Chills must be positioned correctly during
molding.

4. Usage of chills slows down molding time.

5. If left in the mold a considerable amount
of time prior to pouring, condensation on
the chill promotes the susceptibility to
gas related defects later on.

Padding Of Castings

Padding, which is a method of promoting directional
solidification, actually changes the original casting
design slightly. Excess metal is added to the casting so
as to promote directional solidification (see Figure 13).
In Figure 13(a) the original casting is poured without
risers. Centerline shrinkage develops in the connecting
rod along with shrinkage in both hubs. In Figure 13(b),
13-A

13-B: SINGLE RISER

13-C: DOUBLE RISER

13-D: DOUBLE RISER & PADDING

Figure 13. Padding Effect
a riser is added to one of the hubs which eliminates shrinkage in the respective hub. Feeding into the other hub cannot occur due to the connecting rod freezing to the hub. In Figure 13(c) a second riser is added to the other hub. This eliminates all shrinkage except the centerline shrinkage in the connecting rod. Finally, as shown in Figure 13(d) padding is added to the casting. This allows promotion of directional solidification in the connecting rod and, in turn, allows a completely sound casting to be formed. Many foundries could go through all of these experimental stages until a sound casting is finally produced.

Clearly padding is a valuable tool in promoting directional solidification. The drawback is that it may have to be removed by expensive machining operations if the padding is unacceptable to the customer.

Research Goals

This research is intended to consider the cost trade-offs of utilizing different combinations of directional solidification tools. As stated previously, different combinations of directional solidification tools can lead to alternative feasible casting rigging designs. In Figure 14 a ring casting is shown with two alternative directional solidification approaches. Both of these approaches produce
DESIGN 1:
SIX RISERS

DESIGN 2:
THREE RISERS,
THREE CHILLS

Figure 14. Alternative Rigging Example
an equally sound casting. The algorithm resulting from this research effort will consider both alternatives, choosing the one with the minimum cost.

In general, the algorithm will accomplish the following:

1. It will set up an initial feasible solution.
2. It will set up a cost equation to evaluate alternative solutions.
3. It will determine a set of possible areas on the casting which are prime candidates for variable directional solidification tools to be applied.
4. It will vary these areas in a systematic way such that a lower cost is produced, while maintaining feasibility.

Proposed inputs into the algorithms are:

1. casting geometry
2. flask size
3. alloy type
4. cost values
5. appropriate thermal properties

Proposed output of the model is the optimum position and dimension of all risers and chills.

It should be apparent that the final product, the casting, is only a portion of the mold cavity. As previously discussed, the mold cavity is composed of both the casting and its rigging. The ratio of casting weight to pour weight (i.e. both casting and rigging weight) is
termed yield. It is a management measurement of the success of the foundry in metal conservation. The author's experience in steel foundries indicates a yield range of somewhere between thirty and seventy percent. Naturally, this depends upon casting geometry, alloy type, and the degree of usage of other directional solidification tools. This indicates that possibly more than fifty percent of the total weight is utilized in the rigging of the casting. Ruddle [9] estimated that for a steel foundry shipping four hundred tons of castings per month with annual sales of five and one-half million dollars an increase in profit of four hundred and eighty-thousand dollars would be attainable if the yield could be increased from its current level of forty-three percent to a new level of sixty-eight percent.

Foundry costs which have been considered in the evaluation of rigging economics are [9]:

1. Metal Costs
   a. metal cost, including both charge and alloy elements
   b. melting cost, such as electrodes, oxygen, electricity and labor
   c. delivery cost of supplying metal to the mold, including labor, ladle refractory, etc.

2. Removal and Dressing Costs
   a. contact area consideration to remove riser, gates, etc.
b. labor to remove risers

c. labor to remove gates

d. materials such as oxygen, gas, etc. to remove risers, gates, etc.

e. grinding costs to clear contact area, including electricity, grinding wheel costs, labor, etc.

It should be clear that conservation of metal (maximizing yield) can lead to savings within the foundry. However, one should be prudent in the design of rigging. That is, the virtue of maximizing yield should not exceed the requirement of producing castings of sound quality.
Chapter III
CURRENT TECHNIQUES IN RIGGING DESIGN

This chapter provides a review of currently utilized techniques of rigging castings. Included are discussions of riser size and a presentation of two approaches to optimal riser design. As discussed in Chapter II, there are three primary rigging tools utilized to promote adequate directional solidification: risers, chills, and padding. Each is discussed below.

Current Techniques Of Riser Design

Considerable research has been conducted in trying to determine correct riser size (i.e. those providing adequate feed metal). Many foundries, especially smaller ones, utilize an experienced person to determine riser size based upon past history. The following discussion will present six accepted methods for determining riser size. These are:

1. Chvorinov's Method
2. Adams and Taylor Method
3. Caine's Method
4. Bishop's Shape Factor
5. Ruddle's Modulus Method
6. Merchant's Method

7. Pipe Consideration Method

For all of the methods to be discussed one important relationship must hold. That is, the volume of the riser must be greater than the volumetric shrinkage of the casting. Thus:

\[ V_R \geq \beta V_C \]

where:

- \( V_R \) = volume riser
- \( V_C \) = volume casting
- \( \beta \) = solidification shrinkage = \( \frac{\rho_S - \rho_L}{\rho_L} \)
- \( \rho_S \) = density solid
- \( \rho_L \) = density liquid

Chvorinov's Method

Chvorinov was the pioneer in attempting to change riser design from an art to a science. He discovered the following relationship:

\[ \frac{\text{Solidification Time}}{\text{Volume}} = K \left( \frac{\text{Volume}}{\text{Surface Area}} \right)^2 \]

\[ t = K \left( \frac{V}{SA} \right)^2 \]

where: \( K = \) material constant
Chvorinov simply stated that the time for riser solidification must be greater than the solidification time of the casting. Then, the riser dimension can be determined.

\[ t_{\text{riser}} > t_{\text{casting}} \]

\[ \left( \frac{V}{SA} \right)^2_{\text{riser}} > \left( \frac{V}{SA} \right)^2_{\text{casting}} \]

where:

- \( V \) = volume
- \( SA \) = surface area
- \( t_R \) = time to solidify for riser
- \( t_C \) = time to solidify for the casting

**Adams And Taylor Method [6, 10, 11]**

Adams and Taylor incorporated Chvorinov's rule in design of risers by recognizing that the most efficient riser, from a freezing time point of view, is the one in which solidification of the riser ceases simultaneously with the solidification in the casting. Because the riser must provide feed metal throughout the solidification process, the final volume is different from the initial volume.

A material balance on the riser-casting system, assuming all shrinkages resulting in the riser would be:

\[ V_{RF} = V_R - \beta (V_R + V_C) \]

where \( V_{RF} \) = final volume of the riser
The quantity $\beta(V_R + V_C)$ is the total quantity of feed metal $V_R$ required by both the riser and the casting $V_C$.

Combining Chvorinov's relationship along with the material balance equation, the following relationship can be utilized for riser design:

$$\frac{SA_R}{SA_C} = (1-\beta) \frac{V_R}{V_C} - \beta$$

**Caine's Method**

Caine's approach utilized the relationship discovered by Chvorinov—solidification time is proportional to the ratio of volume to surface area. By utilizing empirical data, a relationship of sound versus unsound castings was developed for different casting geometries and riser sizes. The relationship found was as follows (see Figure 15):

$$X = \frac{a}{y-b} + c$$

where:

$$x = \frac{\text{casting surface area/casting volume}}{\text{riser surface area/riser volume}}$$

$$y = \frac{\text{riser volume}}{\text{casting volume}}$$

$a,b$ = constants related to the mode of solidification and contraction from pouring temperature

$c$ = measure of change in relative freezing times

As an example, for steel:

$$X = \frac{0.12}{y-0.05} + 1.0$$
\[ Y = \frac{\text{volume riser}}{\text{volume casting}} \]

\[ X = \frac{\text{surface area casting/volume casting}}{\text{surface area riser/volume riser}} \]

\[ X = \frac{12}{Y - 0.05} + 1.0, \text{ for plain carbon steel castings} \]

Figure 15. Caine's Relationship
Bishop's Shape Factor [12]

Bishop's approach attempts to simplify the calculations required by Caine. The ratio of volume to surface area was replaced with a shape factor where:

\[
\text{Shape Factor} = \frac{\text{Casting Length}}{\text{Casting Width}} + \frac{\text{Casting Thickness}}{\text{Casting Thickness}}
\]

\[
\text{SF} = \frac{L + W}{T}
\]

An investigation was conducted considering cubes, bars, and plates, similar to that of Caine's. The castings were produced with different size risers. The result was the determination of the minimum acceptable size riser. From this data, graphs were developed relating the ratio of riser volume to casting volume for determining shape factors (see Figure 16). This permits determination of the minimum effective riser size for a given casting geometry.

Ruddle's Modulus Factor [13]

This approach is similar to that of Chvorinov in that the ratio of volumetric to surface area is utilized. Ruddle simply defines a modulus where:

\[
M = \text{Modulus} = \left( \frac{\text{Casting Volume}}{\text{Casting Surface Area}} \right)^2
\]

Ruddle claimed empirically that if the modulus of the riser is twenty percent greater than the modulus of the casting sufficient feed metal would be provided.
Shape Factor = \frac{\text{casting length} + \text{casting width}}{\text{casting thickness}}

Figure 16. Relation Of Casting Shape Factor To Minimum Effective Riser Volume To Casting Volume
The modulus of different shaped castings was determined. They are:

1. Cube

   \[ M = \frac{L}{6} \]

2. Cylinder

   \[ M = \frac{DH}{2D + 4H} \]

3. Bar or Plate

   \[ M = \frac{TWL}{2(TW + WL + LT)} \]

4. Endless Cylinder (end terminated by part of casting)

   \[ M = \frac{D}{4} \]
5. **Endless Plate**

\[ M = \frac{T}{2} \]

6. **Endless Bar**

\[ M = \frac{TW}{2(W+T)} \]

7. **Hollow Cylinder**

\[ M = \frac{R_2 - R_1}{2} \]

8. **Flange**

\[ M = \frac{H(TC)}{2(H+(TC))} - TD \]
From the casting modulus considerations riser dimensions can be determined.

**Merchant's Method [14]**

Merchant approached the problem differently from either Chvorinov or Caine. He saw a relationship which existed between the ratio of riser to casting surface area and the ratio of riser to casting volume. This relationship was found by searching available literature for adequate casting riser dimension relationships (see Figure 17). A least fit was then applied. The following equation was determined:

\[
\frac{S_A R}{S_A C} = M \left( \frac{V_R}{V_C} \right) - C'
\]

where:

- \( S_A R \) = surface area riser
- \( S_A C \) = surface area casting
- \( V_R \) = volume riser
- \( V_C \) = volume casting
- \( C' \) = intercept
- \( M \) = slope

For plain carbon steel in a green sand mold:

\[
\frac{S_R}{S_C} = 0.081 \left( \frac{V_R}{V_C} \right) - 0.041
\]
For the graph shown:

\[
\frac{S_R}{S_C} = \frac{\text{Surface Area Riser}}{\text{Surface Area Casting}}
\]

\[
\frac{V_R}{V_C} = \frac{\text{Volume Riser}}{\text{Volume Casting}}
\]

For plain carbon steel in a green sand mold.

Figure 17. Merchant's Linear Relationship
Pipe Discussion [4]

As in the Adams and Taylor method, this method considers the volume of metal in the casting which must be replaced by feed metal in the riser. The volume lost in the riser during solidification is termed the pipe (see Figure 18). The procedure is as follows:

1. Obtain casting weight.

2. Calculate the volume of feed metal required by the casting, where:

\[ V_f = \frac{(\text{casting weight})(\% \text{ solidification shrinkage})}{(\text{weight/cubic inch})} \]

3. Calculate pipe diameter, \( D_p \), and pipe height, \( H_p \), where:

\[ V_f = \pi \left(\frac{D_p}{2}\right)^2 H_p \]

where:

\( H_p = 2.5 \) \( D_p = \) pipe height

\( D_p = \) pipe diameter

4. Determine the riser diameter, \( D_R \), where:

\[ D_R = 2W + D_p \]

where:

\( W = \) casting thickness

5. Determine the height of the riser pressure section, \( H_M \). This depends upon the highest point of the casting to be fed. The pressure height should usually be a minimum of one inch above the highest point to be fed in the casting.
\[ D_p = \text{Diameter of Pipe} \]
\[ H_p = \text{Height of Pipe} \]
\[ H_R = \text{Height of Riser} \]
\[ H_M = \text{Height of Pressure Section of the Riser (usually one inch above highest portion of the casting to be fed)} \]
\[ H_B = \text{Height of Bottom Section of Riser (provides a channel for feed metal to reach the casting)} \]
\[ W = \text{Casting Thickness} \]

Figure 18. Cross Section Of Riser Pipe
6. Design the bottom section of the riser where:

Empirically $2W < H_B < 2W + D_p$

Usually $H_B = 3W$

This section, $H_B$, functions as a channel for feed metal to reach the casting.

7. Determine final riser height, $H_R$, where:

$$H_R = H_P + H_M + H_B$$

The advantage of this system is that it considers the riser requirements regardless of the casting shape. It is simply based upon the feed metal requirements.

As mentioned previously, there are several different methods of calculating riser size (i.e. dimensions). Only a few of the more common ones have been discussed. Notice that all but the pipe method were in some way based upon the basic theory of Chvorinov—solidification time is proportional to the ratio of volume to surface area. Finally, it is important to note that nearly all of the methods were based upon empirical testing.

Location Of Risers

The previous discussion considered the appropriate riser size for a given casting shape, surface area to volume ratio, etc. No consideration as yet has been given to where the riser should be located with respect to the casting. Obviously, one of the most important considerations
is the feeding range of the riser. Different sources such as Woodlower [15], Ruddle [13], and others give empirical data on feeding range.

Another consideration with respect to riser placement is what is termed the "end effect" [4]. This refers to the fact that a certain length from each end of a casting, especially a bar or plate, is sound even without a riser. This results from the directional solidification that develops at the ends because of the greater heat extraction from those points as compared with others. The same sources, [4, 13, 15], give data on the end effect. The end effect is best summarized in Figure 19 [16]. Note the riser contribution is only twice the thickness. With the end effect included, a total of four and one-half times the plate thickness is sound.

It should be clear that the current method of determining riser characteristics is through experience and trial and error. All of the listed methods are based upon empirical relationships proven to be successful through years of testing. Also note that none of the latest methods have considered cost in riser determination.

A large, complex casting may need a series of risers. The general method of determining multiple risering is to divide the casting into sections, then to determine the riser requirements of each section independently.
Figure 19. End Effect
Chill Design

Woodlower [15] discusses the calculation of chill size. Chills should reduce the time taken for a casting or portion of a casting to solidify. This, in turn, should allow for the application of smaller risers. Thus, chill utilization could increase yield by lowering required riser sizes.

Chills, as discussed in Chapter II, extract heat faster from the casting than the surrounding mold environment. As in the mold wall, chills accept a portion of the superheat, latent heat of fusion, and heat from cooling of the already solidified casting.

In modeling chill effectiveness, the idea is that a casting of volume \( V \), utilizing a chill should show a diminished solidification time equal to that of a casting of volume \( V' \). Thus, application of a chill produces an apparent reduction in casting volume. Similarly from a heat extraction point of view the amount of heat corresponding to the weight, differences of the actual casting volume (i.e. without the chill), and the apparent casting volume (i.e. with the chill) should be accepted by the application of the chill.

Thus an alteration is applied either to the casting modulus to reduce it or to the casting volume. Given the reduction a new, smaller riser can be utilized. One should
note, however, that this method does not determine the appropriate chill size, only the riser size for the given chill chosen.

Ruddle [13] formulates the problem in a similar manner where the modulus of a casting is reduced by the use of a chill as follows:

\[ M'_C = M_C \left( 1 - \frac{W_{\text{chill}}}{0.27 W_{\text{casting}}} \right) \]

where:

- \( M'_C \) = revised casting modulus
- \( M_C \) = original casting modulus
- \( W_{\text{chill}} \) = chill weight
- \( W_{\text{casting}} \) = casting weight

This is an empirically developed formula. Obviously, the size of the chill can be determined by solving the above equation for \( W_{\text{chill}} \), if the desired reduction in the casting modulus is known. This approach also requires the contact area of the casting and chill to satisfy the following:

\[ A_{\text{chill}} = \frac{V_C}{2} \left( \frac{M_C - M'_C}{M_C M'_C} \right) \]

Ruddle further recommends that if it is desired to reduce the modulus of a heavy section of a casting to equalize it or make it less than the modulus of an adjacent lighter
section, then chills must only be used to effect a forty percent reduction. If more than this is required, chills must not be used and the heavier section will have to be fed with its own riser.

Padding Design

Padding can be defined as a change in the casting geometry with the goal of promoting directional solidification. As discussed in Chapter II, padding basically extends the effective feeding distance of a riser. Woodlower [15] gives standard padding allowances one could employ. They are developed empirically and have as their major variable casting wall thickness where padding is to be applied. The idea is to determine the extended riser feeding distances, compute a new casting modulus, and then use a small number of risers each of a smaller size.

Optimal Riser Design

Creese [17, 18, 19] formulated the problem of riser design as one that minimized the riser volume subject to Chvorinov's relationship—riser solidification time is greater than or equal to the casting solidification time. That is:

\[
\text{Minimize Volume Riser} \\
\text{Subject To:} \quad \frac{\text{Riser Solidification Time}}{} > \frac{\text{Casting Solidification Time}}{}
\]
The optimization technique applied was geometric programming.

The basic theory of what Creese set up is as follows:

General Formulation:

\[ \text{MIN } V_R = f (D_R, H_R) \]

Subject to \( t_R > t_C \)

where:

- \( V_R \) = riser volume
- \( D_R \) = riser diameter
- \( H_R \) = riser height
- \( t_R \) = time for riser to solidify
- \( t_C \) = time for casting to solidify

Assuming Chvorinov's rule,

\[ t = K \left( \frac{V}{SA} \right)^n \]

\( t \) = time

\( K \) = constant

\( n \) = constant

\( V \) = volume

\( SA \) = surface area

then:

\[ t_R = K_R \left( \frac{V_R}{SA_R} \right)^{n_R} \]

\[ t_C = K_C \left( \frac{V_C}{SA_C} \right)^{n_C} \]
The constraint can then be represented:

\[ t_R \geq t_C \]

\[ K_R \left( \frac{V_R}{S_{AR}} \right)^{n_R} \geq K_C \left( \frac{V_C}{S_{AC}} \right)^{n_C} \]

\[ \left( \frac{V_R}{S_{AR}} \right)^{n_R} \geq \frac{K_C}{K_R} \left( \frac{V_C}{S_{AC}} \right)^{n_C} \]

\[ \left( \frac{V_R}{S_{AR}} \right)^{n_R} \geq \left( \frac{K_C}{K_R} \right)^{1/n_R} \left( \frac{V_C}{S_{AC}} \right)^{n_C/n_R} \]

Note: \( K_R, K_C, n_R, n_C, V_C, S_{AC} \) are constants for a given problem.

\[ \left( \frac{V}{SA} \right)_R \geq y = \text{casting system modulus} \]

General relationships of volume to surface area:

\[ V_R = AD_R^2 H_R + B D_R^3 \]

\[ S_{AR} = CD_R^2 + KD_R H_R \]

where: \( A, B, C \) and \( K \) are constants to encompass different shaped risers.

\[ \left( \frac{V}{SA} \right)_R = \frac{AD_R^2 H_R + B D_R^3}{CD_R^2 + KD_R H_R} \geq y \]

The problem can then be formulated as:
\[
\begin{align*}
\min \quad V_R &= AD_R^2 H_R + BD_R^3 \\
\text{subject to:} \quad \frac{AD_R^2 H_R + BD_R^3}{CD_R^2 + KD_R H_R} &\geq y
\end{align*}
\]

Using geometric programming, the results are:

\[
H_R = \frac{3y}{2A} \left(\frac{2CA-3BK}{A}\right)
\]

\[
D_R = \frac{3y}{2A} K
\]

\[
V_R = \left(\frac{3y}{2A}\right)^3 2K^2 (CA-BK)
\]

where: \[ y = \frac{K_C}{K_R}^{1/n_R} \frac{V_C^{n_C/n_R}}{S_{AC}} \]

Creese [17] compared results of the geometric programming solution to that of three accepted risering equations:

1. Caine's Riser Equation
2. Shape Factor Equation
3. Adams and Taylor Equation

Each of the above techniques have been discussed previously in this chapter. A two-inch by eight-inch by six-inch steel plate was considered with a cylindrical side riser (see Figure 20). Comparison of all three equation recommendations to that of the geometric programing recommendation are also exhibited in Figure 20. Note that the geometric programing recommendation compares favorably with the other approaches.
The rectangular casting below, and the riser attached, is the design that Creese worked with. He formulated the problem to minimize riser volume, while at the same time, satisfying Chvorinov's rule. A Geometric Programming Technique was utilized.

Figure 20. Creese's Optimization Design
Creese further extended the problem to consider cylindrical riser design with insulating materials [11].

Finally, Poirier and Gandhi [10] considered optimization of riser design with insulating sleeves. Unlike Creese's work, costs were considered. An equation was derived for the cost of a riser. The equation contained the following:

1. Riser size as the variable.
2. The constraint that riser solidification time is greater than the casting solidification time, based upon Chvorinov's relationship. Costs of the riser include:
   a. sleeve cost, if utilized
   b. riser removal cost
   c. metal melting cost
   d. metal loss cost

First order necessary conditions were applied to find the optimal solution. The equation and solution follow:

The cost equation is:

\[
C = \frac{K_2 + K_1 D^2 (Z_1 + Z_2)}{S_1 D^2 + S_2 D^2 - S_3}
\]

where:

\[
C = \text{cost}
\]

\[
K_1 = \frac{\text{volume casting/surface area casting}}{(1-\text{fraction feed metal})}
\]

\[
K_2 = \frac{4(\text{fraction feed metal})(\text{volume casting})}{\pi (1-\text{fraction feed metal})}
\]
riser diameter

$Z_1, Z_2 =$ apparent surface alteration factor for top and side of riser

$$e = \frac{B_S (\alpha_S)}{B_C (\alpha_C)}$$

where: $B_S, B_C$ dimensionless group for sand casting; $\alpha_S, \alpha_C$ thermal diffusivity for sand casting.

$S_1, S_2, S_3 =$ cost coefficients

$$\frac{dC}{dD} = 0 \Rightarrow D_0^3 - B_2 D_0^2 + B_1 D_0 - B = 0$$

where:

\[
B_2 = \frac{8K_1e + 6K_1^2S_1}{s_2} \frac{ze}{1 + K_1 S_1} \frac{z}{s_2}
\]

\[
B_1 = \frac{16K_1^2e^2}{K_1 S_1} \frac{z}{1 + \frac{K_1 S_1}{s_2}}
\]

\[
B = \frac{2K_1K_2S_1e}{s_2} \left( 1 + \frac{K_1 S_1}{s_2} \right) z
\]
The solution for $\frac{dC}{dD} = 0$ is the roots of the equation. Substituting $D_0$, the roots into the cost equation yield the optimal cost.

It should be noted that none of the optimization models presented consider:

1. riser feeding distance
2. solidification shrinkage
3. inherent casting thermal gradient
4. riser neck design
5. possible chill application
6. feeding paths within the casting
7. riser placement
Chapter IV

SOLIDIFICATION MODELING

The purpose of this chapter is threefold: (1) to present a brief outline of currently existing methods to quantify solidification, (2) to outline the solidification simulation procedure utilized, and (3) to discuss some results of the simulation. The reader should be reminded that it is not the purpose of this research to develop a solidification simulation, but only to utilize the simulation to develop approximations to the shrinkage gradient and in turn use these approximations in the optimization package. As will be discussed later, this was necessary because the CPU time for even a simple casting of small size was enormous.

Methods Of Quantifying Solidification

The purpose of this discussion is to consider classical methods of quantifying solidification. By quantifying, the author implies predicting the solidification pattern (i.e. determining a time-dependent temperature distribution throughout the casting). If this were possible one would know when each section of the casting became completely
solid. Also, all of the passageways through which liquid feed metal could pass would be apparent, along with those sections which would solidify last, thus requiring liquid feed metal. Clearly, this knowledge would assist in determining proper rigging design.

The problem of quantifying solidification is not a trivial one. From a theoretical approach it quickly becomes mathematically burdensome. Extensive research has been conducted with respect to empirical methods of determining solidification. Five major categories of research with respect to quantifying solidification are:

1. Pour Out
2. Temperature Measurement Technique
3. Analogue Methods
4. Mathematical Analysis
5. Approximation Techniques

Each of these research categories are discussed separately in the paragraphs which follow.

**Pour Out Method**

This method was used in early investigations by Briggs and Gezelius [20]. According to Ruddle [21], it is still applied today. The technique is simple in that a number of identical molds are produced and then poured using the same alloy. At different time intervals after pouring the
molds are overturned and the liquid metal runs out leaving behind a partially solidified casting. From this experiment a relationship of wall thickness versus time can be determined. The drawbacks to this method are that it gives no information with respect to temperature gradients within the solidifying casting or in the mold, it is restricted to the early stages of solidification, and it is not applicable to any alloy with a large freezing range. Another drawback is that this method wastes a number of molds in the process.

Temperature Measurement Technique

This is a method devised as an alternative to the pour out method. In this approach thermocouples are situated at strategic positions within the mold cavity. A time-temperature graph is recorded for each thermocouple. One advantage of this method is that it yields clear information on events of solidification. It also allows both monitoring of the beginning and the end of solidification. Because temperatures are being taken at the same time at different positions in the mold cavity, thermal gradients can be determined. Finally, another advantage over the pour out method is that its application does not require production and waste of a number of molds and castings. This method is usually used for verification of solidification simulation results.
Analogue Methods

Analogue methods are those which simulate the flow of heat or distribution of temperature by the flow or distribution of some other medium whose properties are more easily measurable. Electrical analogues have been applied mostly to the study of solidifying castings.

Mathematical Analysis

Ruddle [21] explains that a major disadvantage of mathematical analysis is the complexity of the subject. Exact analytical solutions have been worked out for only a few simple cases, such as that of a semi-infinite mass of metal brought instantaneously into contact with a plane mold wall, also extending into infinity. Even for solution of this simple case, many assumptions are necessary. The assumptions are:

1. the temperature at the mold-metal interface remains constant;

2. the superheat of the metal is included in the latent heat of fusion allowance; and

3. it necessitates knowledge of thermal properties of all materials concerned.

However, mathematical treatment can prove valuable toward quantifying solidification.

Approximations as to the total time until solidification is complete for various shaped castings have been
proposed. Heine, Loper and Rosenthal [4] claim that the thickness of a casting solidified at any time \( t \) is:

\[
D = K(t)^{1/2} - C
\]

where:

- \( D \) = thickness solidified
- \( t \) = time
- \( K, C \) = constants

Chvorinov claimed that the time for complete solidification of a plate and sphere are [22]:

1. Plate:

\[
T_P = \left[ \frac{\pi \rho_1^2 L + C_1 (\theta_1^0 - \theta_{SM})}{4 \rho_2 C_2 \lambda_2 (\theta_1 - \theta_2^0)^2} \right] \left( \frac{V_C}{SA_C} \right)^2
\]

2. Sphere:

\[
T_S = \left[ \frac{9 \rho_2 C_2}{\pi \lambda_2^2} \right] \left[ \left( 1 + \frac{\pi \rho_1 [L + C_1 (\theta_1^0 - \theta_{SM})]}{3 \rho_2 \lambda_2 (\theta_1 - \theta_2^0)} \right)^{1/2} - 1 \right]^2 \left( \frac{V_C}{SA_C} \right)^2
\]

Adams and Taylor extended this to cylinders. It is given as (6):

\[
T_C = \left[ \frac{16 \rho_2 C_2}{\pi \lambda_2^2} \right] \left[ \left( 1 + \frac{\pi \rho_1 [L + C_1 (\theta_1^0 - \theta_{SM})]}{4 \rho_2 C_2 \lambda_2 (\theta_1 - \theta_2^0)} \right)^{1/2} - 1 \right] \left( \frac{V_C}{SA_C} \right)^2
\]
Variables for all the approximation equations are:

\[ T_p = \text{solidification time of infinite plate (length and width infinite)} \]

\[ T_s = \text{solidification time sphere} \]

\[ T_c = \text{solidification time cylinder (length infinite)} \]

\[ \lambda_1, \lambda_2 = \text{thermal conductivity metal, mold} \]

\[ \rho_1, \rho_2 = \text{density metal, mold} \]

\[ C_1, C_2 = \text{specific heat metal, mold} \]

\[ L = \text{latent heat of fusion} \]

\[ \theta_1^0, \theta_2^0 = \text{initial temperature of metal, mold} \]

\[ \theta_i = \text{interface temperature} \]

\[ \theta_{SM} = \text{average temperature of casting at end of solidification} \]

\[ V_C = \text{volume casting} \]

\[ S_{AC} = \text{surface area of casting in contact with the mold} \]

All of the above equations were derived by utilizing a heat balance between the metal and the mold. This basic heat balance is utilized in the finite difference simulation. All of the basic equations suffer from the fact that they cannot express the temperature at any time and position within the mold cavity. It is necessary to determine these data by simulation riser-chill design.

One method of obtaining a time-dependent temperature distribution would be to solve the basic heat flow equation.
It is:

\[ \frac{\partial \Theta}{\partial t} = \frac{k}{\rho c} \left( \frac{\partial^2 \Theta}{\partial x^2} + \frac{\partial^2 \Theta}{\partial y^2} + \frac{\partial^2 \Theta}{\partial z^2} \right) \]

where:

\( \Theta \) = temperature of any element
\( k \) = thermal conductivity, solid
\( t \) = time
\( \rho \) = density
\( c \) = specific heat (assumed constant with respect to temperature)

\[ \alpha = K = \frac{k}{\rho c} = \text{thermal diffusivity (assumed constant with respect to temperature)} \]

Considering only unidirectional solidification a solution could be \([21, 23]\):

\[ \Theta = t^{-1/2} e^{-x^2/4kt} \]

proof:

\[ \frac{\partial \Theta}{\partial t} = -1/2t^{-3/2}e^{-x^2/4kt} + \frac{x^2}{4kt^{5/2}} e^{-x^2/4kt} \]

\[ \frac{\partial \Theta}{\partial x} = -\frac{2x}{4kt^{3/2}} e^{-x^2/4kt} \]

\[ \frac{\partial^2 \Theta}{\partial x^2} = \left( -\frac{1}{2kt^{3/2}} \right) e^{-x^2/4kt^{3/2}} + \frac{x^2}{4kt^{5/2}} e^{-x^2/4kt} \]
remembering:
\[ \frac{\partial \theta}{\partial t} = k \frac{\partial^2 \theta}{\partial x^2} \]

substituting:

\[
\frac{-1}{2t^{3/2}} e^{-x^2/4kt} + \frac{x^2}{4kt^{5/2}} e^{-x^2/4kt} = \\
k \left[ \frac{-1}{2kt^{3/2}} e^{-x^2/4kt^{3/2}} + \frac{x^2}{4k^2t^{5/2}} e^{-x^2/4kt} \right]
\]

\[
e^{-x^2/4kt} \left[ \frac{-1}{2t^{3/2}} + \frac{x^2}{4kt^{5/2}} \right] = e^{-x^2/4kt} \left[ \frac{-1}{2t^{3/2}} + \frac{x^2}{4kt^{5/2}} \right]
\]

thus:

\[ \theta = t^{-1/2} e^{-x^2/4kt} \]

is a solution.

Other solutions incorporate an error function and can be found in Carslaw's *Conduction Of Heat In Solids*.

The theory can be extended to address more complicated examples, however, the mathematical complexity associated with it also increases. One should also be aware that it is a heat conduction solution. It does not give solutions to the solidification problem since the latent heat of fusion is not included.

**Approximation Techniques**

Clearly, as demonstrated in the previous discussion analytical techniques are both mathematically burdensom
and are only applicable to the most simple solidification problems. Approximation techniques, although not exactly analytical, yield solutions to more practical problems. One technique is Schmidt's graphical approach. This technique lays the groundwork for the simulation discussion which follows.

Schmidt's technique is similar to a moving average forecasting technique. The method described is applicable to unidimensional heat flow problems of an infinite slab. The method is as follows [21]:

For an infinite slab, the conduction equation is:

\[
\frac{\partial \theta}{\partial t} = \alpha \frac{\partial^2 \theta}{\partial x^2}
\]

where:

\( \theta \) = temperature at any perpendicular distance \( x \) from a face at time \( t \)

\( t \) = time

\( \alpha \) = temperature diffusivity

\( \Delta x \) = small increment in \( x \)

\( \Delta t \) = small increment in time \( t \)
The conduction equation can be rewritten as:

where:

\[ \Delta_x \theta = \text{change in } \theta \text{ due to } \Delta x, \text{ with time, } t, \text{ constant} \]

\[ \Delta_t \theta = \text{change in } \theta \text{ due to } \Delta t, \text{ with distance, } x, \text{ constant} \]

thus,

\[ \frac{\partial \theta}{\partial t} = \frac{\Delta_t \theta}{\Delta t} \]

\[ \frac{\partial \theta}{\partial x} = \frac{\Delta_x \theta}{\Delta x} \]

\[ \frac{\partial^2 \theta}{\partial x^2} = \frac{\Delta (\Delta_x \theta)}{(\Delta x)^2} \]

\[ \frac{\Delta_t \theta}{\Delta t} = \alpha \frac{\Delta_x (\Delta_x \theta)}{(\Delta x)^2} \]

\[ \Delta_t \theta = \alpha \frac{\Delta t}{(\Delta x)^2} \Delta_x (\Delta_x \theta) \]

Note: \( \alpha \) is a material constant and \( \Delta t \) and \( \Delta x \) are user-determined step sizes. If one chose:

\[ \alpha \frac{\Delta t}{(\Delta x)^2} = \frac{1}{2} \]

Then the heat conduction equation would be:

\[ \Delta_t \theta = \frac{1}{2} \left( \Delta_x (\Delta_x \theta) \right) \]
A graphical representation would be:

\[ \theta_{n,m} \] = temperature at a point \( n \Delta x \) units from surface after \( m \Delta t \) units of time

\[ \Delta_t \theta = \theta_{n,m+1} - \theta_{n,m} \]

\[ \Delta_x \theta = \theta_{n+1,m} - \theta_{n,m} \]

\[ \Delta_x^2 \theta = \Delta_x (\Delta_x \theta) \]

\[ = (\theta_{n+1,m} - \theta_{n,m}) - (\theta_{n,m} - \theta_{n-1,m}) \]

Substituting into the heat balance equation:

\[ \theta_{n,m+1} - \theta_{n,m} = \frac{k}{\Delta x^2} \left[ (\theta_{n+1,m} - \theta_{n,m}) - (\theta_{n,m} - \theta_{n-1,m}) \right] \]

\[ \theta_{n,m+1} = \frac{k}{\Delta x^2} (\theta_{n+1,m} + \theta_{n-1,m}) \]
Thus, the temperature of the next time unit is equal to the average of the previous time unit on either side of the element.

One can clearly see that more complex problems can be dealt with when utilizing approximation techniques. Casting geometry can be considered along with other options not available when using a mathematical approach. The following section discusses computer simulations of solidification which are based on or utilize the previously discussed approximation techniques.

**Computer Simulation Of Solidification**

The purpose of this section is to discuss computer simulation of solidification. Previously discussed were approximation techniques, with respect to quantifying solidification. The computer techniques are basically an extension of them. The basic approach in all computer simulations discussed below is an explicit finite difference analysis (i.e. numerical integration).

The procedure for explicit finite differencing is as follows:

1. set up differential equations describing the heat flow;
2. determine initial conditions;
3. determine boundary conditions;
4. place a grid around the casting design in question such that the grid size is $dx$, $dy$, $dz$, where:

![Diagram of grid around casting design]

5. at time = 0, the initial temperature at each grid point $(i,j)$ is known by initial conditions; and

6. the derivatives set up in step (1) should be evaluable by finite difference expressions (the expressions should be a function of constants $dx$, $dy$, $dz$, or $dt$, the time step) over different time steps.

This procedure should yield a transient temperature profile of the casting under consideration.

Pehlke [24, 25, 26, 27, 28, and 29] has been a leader in this field. The following simulations have been conducted successfully (i.e. simulation results correlate well with actual results):

1. square bar [24]
2. "T" section [25]
3. "L" section [26]
4. steel wheel casting [27]
5. casting with a chill [28]
In all of the simulations listed above the general procedure which was previously outlined was utilized. In all of the simulations the simulation results were compared to experimental results. The experimental results were produced by time-temperature thermocouple measurements. Agreement between computed and experimental results was excellent. Very little variation could be found at key positions within the casting. The order of solidification was always predicted correctly.

Even for application of simulation techniques assumptions are necessary. Some are:

1. The mold is instantaneously filled with liquid at a known pouring temperature.

2. Once the metal is poured it is stagnant (i.e. no mass convection or energy transport occurs in the liquid phase).

3. Liquidous and solidous lines are well-defined—this allows plotting of the beginning and the end of freeze waves.

4. The problem could be modeled using a two-dimensional heat flow simulation.

Pehlke further claimed [29] that both the time-temperature curves and the order of freezing could be accurately predicted by numerical simulation of solidification. This is regardless of both the metal and the casting complexity. A casting containing a series of connective discs and cylinders was used as an example, wherein the computer simulation predicted the occurrence of a shrinkage
cavity. This was based upon the order of solidification at various points within the casting. When the casting was experimentally made, the shrinkage cavity occurred in the predicted position.

The simulations can be quite detailed. In Pehlke's [28] simulation of solidification of a casting against a chill even the air gap created between the solidified metal and the chill was incorporated into the model. Again, the correlation between experimental results from thermocouple measurements taken within the casting and the solidification simulation results were excellent.

The simulation technique can also be utilized for more complex geometries. Pehkle [27] applied the computer simulation to a complex railroad wheel casting. It was shown that the shape of the riser shrinkage cavity could be accurately calculated using the computer simulation. The shrinkage cavity shape could then be used to determine the sufficiency of the risering system. Alternative riser designs involving exothermic padding were evaluated. The improved riser design eliminated the shrinkage defect in the casting.

Weatherwax and Riegger [30] simulated the solidification of a die-cast aluminum piston. The simulation method utilized was that of explicit finite difference. The simulation accomplished the following:
1. By realizing the solidification pattern, the location of shrinkage porosity can be predicted and the rigging system can be designed to move the shrinkage cavity into a riser and out of the casting.

2. By predicting solidification rates, mechanical properties can also be predicted.

In all of the examples the following requirements must be fulfilled for success of simulation. To be specific, one must:

1. Formulate an accurate physical description of the casting and solidification process in mathematical form.

2. Obtain accurate values for thermal properties of both the casting and the mold materials.

3. Perform a suitable numerical analysis to obtain an algorithm which can predict temperature-time relationships at specified space coordinates in the mold.

The advantages of numerical simulation are:

1. Accuracy in predicting the solidification pattern of a casting without burdensome mathematical analysis.

2. Reduction in trial and error on the shop floor.

3. Evaluation of more alternative designs since the simulation is quicker than actual trial and error.

4. Complex casting geometries can be considered as compared to only simple geometries with the use of analytical methods.

However, a major problem with numerical simulation is that of accurately predicting thermal properties, such as temperature-dependent thermal conductivities, specific heats, etc.
Finally, there are general purpose heat transfer codes available to the public. Some are:

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Developer</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADINAT</td>
<td>Massachusetts Institute Of Technology</td>
<td>31</td>
</tr>
<tr>
<td>AVER</td>
<td>Los Alamos Scientific Lab</td>
<td>32</td>
</tr>
<tr>
<td>SINDA</td>
<td>NASA</td>
<td>34</td>
</tr>
<tr>
<td>CINDA-3G</td>
<td>NASA</td>
<td>33</td>
</tr>
<tr>
<td>HEATING-5</td>
<td>Oak Ridge Numerical Lab</td>
<td>35</td>
</tr>
<tr>
<td>MITA-5</td>
<td>Control Data Corporation</td>
<td>36</td>
</tr>
<tr>
<td>TRUMP</td>
<td>Lawrence Radiation Lab</td>
<td>37</td>
</tr>
</tbody>
</table>

The approach utilized in this research was that of an explicit finite difference simulation. The simulation included approximations for both initial sand-metal interface temperatures and latent heat of fusion. The approach is to set a grid network over the two-dimensional casting surface and mold cavity (i.e. nodal network), determine heat flow equations, and then simulate. The following is a derivation for the method of temperature updates at individual nodes.

**Derivation Of Nodal Temperatures**

The following diagram depicts an example of the nodal grid:
The heat balance for the two-dimensional nodal network is:

\[
\begin{bmatrix}
\text{Heat from node } i,j \text{ to nodes } \\
(i-1,j-1), (i-1,j), (i-1,j+1), \\
(i,j-1), (i,j+1), (i+1,j-1), \\
(i+1,j), (i+1,j+1)
\end{bmatrix}
= \begin{bmatrix}
\text{Heat dissipated} \\
\text{from element } i,j
\end{bmatrix}
\]

Heat dissipated from element \(i,j = C_{i,j}\left(\theta'_i,j - \theta_i,j\right)\)

where:

\(C_{i,j} = \text{specific heat of node } i,j (\text{BTU/lb} \cdot \text{°F})\)

Heat conducted from node \(i,j\) to node \(i-1, j-1 = \)

\[K_{i-1,j-1 \text{ to } i,j}\left(\theta_{i-1,j-1} - \theta_{i,j}\right) \frac{\Delta t \Delta A}{\Delta x}\]

where:

\[K_{i-1,j-1 \text{ to } i,j} = \text{thermal conductivity of heat from node } i-1, j-1 \text{ to node } i,j. \text{ Units are:}\]

\[\frac{\text{BTU}}{\text{HR} \cdot \text{FT}^2 \cdot \text{°F}}\]
T = Temperature (°F)
Δt = incremental time step (hours)
A = contact area (square feet)
Δx = nodal length (in feet)

The following temperature-dependent specific heats and thermal conductivities were utilized in the solidification simulation and can be found in Pehlke [25].

Specific heat $C_{i,j}$ for low carbon steel =

\[
\begin{align*}
0.118 + 0.00005 (T-200) & \quad \text{if } 0 < T < 200 \\
0.198 + 0.000148 (T-800) & \quad \text{if } 200 < T < 800 \\
0.222 - 0.000108 (T-1300) & \quad \text{if } 800 < T < 1800 \\
0.168 + 0.000381 (T-1800) & \quad \text{if } 1800 < T < 2640 \\
0.20 + 0.0579 (T-2640) & \quad \text{if } 2640 < T < 2685 \\
2.805 + 0.0576 (T-2685) & \quad \text{if } 2685 < T < 2730 \\
0.21 & \quad \text{if } 2730 < T \\
\end{align*}
\]

Specific heat $C_{i,j}$ for green molding sand =

\[
\begin{align*}
0.17 + 0.000125 T & \quad \text{if } 0 < T < 800 \\
0.26 + 0.000015 (T-800) & \quad \text{if } 800 < T < 2800 \\
\end{align*}
\]

Thermal conductivity $K_{i,j}$ for low carbon steel =

\[
\begin{align*}
28 - 0.006875 (1600-T) & \quad \text{if } 0 < T < 1600 \\
17 - 0.009615 (T-1600) & \quad \text{if } 1600 < T < 2640 \\
1.18 - 0.033 (T-2640) & \quad \text{if } 2640 < T < 2730 \\
1.15 & \quad \text{if } 2730 < T \\
\end{align*}
\]

Thermal conductivity $K_{i,j}$ for green molding sand =

\[
0.38171 - 6.6889 \cdot 10^{-5} T + 1.3805 \cdot 10^{-7} T^2
\]

The basic equation then is, allowing SS to represent all eight surrounding nodes:

\[
\sum_{\text{SS}} K_{\text{SS to } i,j} (\Theta_{\text{SS}} - \Theta_{i,j}) \frac{\Delta t \cdot A_{\text{contact SS to } i,j}}{\Delta x_{\text{SS to } i,j}} = C_{i,j}(\Theta_{i,j}' - \Theta_{i,j})\rho_{i,j} V_{i,j}
\]
Assume

\[ A_{SS} \text{ to } i,j = \text{the contact area for all nodes including diagonals and is constant} \]

The unit balance then is:

\[
\begin{bmatrix}
\frac{\text{BTU}}{K} \\
\frac{\text{BTU}}{F^2 T} \\
\frac{\text{BTU}}{H^3 F^2 T}
\end{bmatrix}
\begin{bmatrix}
(\cdot F) (H K) (F T^2) \\
(\cdot F) (L H - F^3 T) (F T^3)
\end{bmatrix}
= \text{BTU}
\]

\[ \text{BTU} = \text{BTU} \]

Allowing \( \Delta X_{SS} \text{ to } i,j \) = nodal distance where for pure contact
\( \Delta X_{SS} \text{ to } i,j = \Delta X \) and for corner contact, assuming heat flow
at 45°, \( \Delta X_{SS} \text{ to } i,j = 1.414\Delta X \). Also allow \( V_{i,j} = (\Delta x)^2 \cdot 1 \),
assuming unit depth (two-dimensional heat flow).

Substitute into basic heat flow equation:

\[
\sum_{SS} K_{SS \text{ to } i,j} (\Theta_{SS} - \Theta_{i,j}) \frac{\Delta t \cdot A_{\text{contact } SS \text{ to } i,j}}{\Delta X_{SS \text{ to } i,j}} = C_{i,j} (\Theta_{i,j} - \Theta_{i,j}) \rho_{i,j} (\Delta x)^2
\]

\[
= C_{i,j} \rho_{i,j} (\Delta x)^2 (\Theta_{i,j} - \Theta_{i,j})
\]

Assuming \( A_{\text{contact SS to } i,j} = \Delta X \cdot 1 \) since for a two-dimensional
heat flow model it is only a line. This assumption is consistent with the literature [24, 25, 26]. Substitute:

\[
\sum_{SS} K_{SS \text{ to } i,j} (\Theta_{SS} - \Theta_{i,j}) \Delta t \frac{\Delta X_{SS \text{ to } i,j}}{(2)^{1/2} \Delta X_{SS \text{ to } i,j}} = C_{i,j} \rho_{i,j} (\Delta x)^2 (\Theta_{i,j} - \Theta_{i,j})
\]

cancel on all but 45° which has \( 2^{1/2} \) dependency.
Dividing both sides by $C_{i,j}$ $\rho_{i,j}(\Delta x)^2$

$$\sum_{SS} \frac{K_{SS \rightarrow i,j} \Delta t}{2} \frac{1/2 C_{i,j} \rho_{i,j}(\Delta x)^2}{(2)^{1/2}} (\theta_{SS} - \theta_{i,j}) = \theta'_{i,j} - \theta_{i,j}$$

Simplifying:

$$\theta'_{i,j} = \theta_{i,j} + \left[ \frac{\Delta t}{C_{i,j} \rho_{i,j}(\Delta x)^2} \right] \left[ \sum_{SS} \frac{K_{SS \rightarrow i,j}}{2^{1/2}} \right] x$$

$$= \theta_{i,j} + \left[ \frac{\Delta t}{C_{i,j} \rho_{i,j}(\Delta x)^2} \right] \left[ \sum_{SS} \frac{K_{SS \rightarrow i,j}}{\text{FAC}(SS)} \right]$$

where:

$$\text{FAC}(SS) = \begin{cases} 
2^{1/2} & \text{if node SS is a corner contact} \\
1 & \text{if node SS is not a corner contact}
\end{cases}$$

Finally,

$$\theta'_{i,j} = \theta_{i,j} - \left( \frac{\Delta t}{\rho_{i,j} C_{i,j}(\Delta x)^2} \right) \theta_{i,j} \left( \sum_{SS} \frac{K_{SS \rightarrow i,j}}{\text{FAC}(SS)} \right)$$

$$+ \left( \frac{\Delta t}{C_{i,j} \rho_{i,j}(\Delta x)^2} \right) \left( \sum_{SS} \frac{K_{SS \rightarrow i,j}}{\text{FAC}(SS)} \right) \theta_{SS}$$
The derivation previously described does not consider the phenomena of the latent heat of fusion dissipation. Latent heat of fusion is the amount of energy necessary for the phase change from liquid to solid to occur. This is what separates the pure heat conduction model from a solidification model. As Figure 21 demonstrates, the latent heat of fusion effectively delays the solidification process. To approximate this, Pehlke, et al. [30, 38, 39] has utilized an effective specific heat for steel which is considerably higher through its mushy stage than in other stages. This allows the solidification to be delayed and thus the latent heat of fusion affect is approximated. Figure 21 demonstrates the effective specific heat with respect to latent heat of fusion dissipation.

**Solidification Model Results**

There are four areas of discussion presented with respect to model verification: (1) the method of determining shrinkage cavity size, (2) results of a latent heat of fusion effect included in the simulation, (3) riser size effects, and (4) gradient results.

**Shrinkage Cavity Size**

First, it should be realized that this research is concerned only with those castings exhibiting internal shrinkage cavities which are removable via riser and chill
Figure 21. Conductivity Vs. Solidification
combinations. Thus, a casting using only chills to drive the shrinkage cavity to the top is not considered (see Figure 22). Those castings utilizing only risers are considered.

To determine the size of the shrinkage cavity for a casting with a given riser-chill design, two variables must be known. The first, riser-casting contact nodes, must be determined and then the volumetric solidification shrinkage for the alloy under consideration must be known. The riser-casting contact nodes within the simulation nodal gridwork are those which are in contact with both the riser and the casting. The hypothesis then is that if all riser-casting contact nodes are solid there is no future possibility of any riser feeding the casting cavity. Thus, the nodes still liquid in the casting will exhibit a shrinkage cavity equal to:

\[
\text{Shrinkage Cavity} = \frac{\text{Volumetric Summation of All Nodes} \times \text{Percent Still Liquid After All Riser-Casting Contact Nodes Are Solid}}{\text{Shrinkage}}
\]

Figure 23 exhibits a flow chart of the determination of shrinkage cavity size.

**Latent Heat Of Fusion**

As discussed previously, for a solidification simulation to be truly a solidification simulation as opposed to
Figure 22. Example of Casting & Rigging Design
Not Covered By Formulation
ARE ALL RISER-CASTING NODES SOLID?

SIMULATE SOLIDIFICATION SIMULATION ITERATION I

ARE ALL RISER-CASTING NODES SOLID?

I=I+1

DETERMINE ALL RISER-CASTING CONTACT NODES

I=1

SIMULATE SOLIDIFICATION SIMULATION ITERATION I

DETERMINE SHRINKAGE CAVITY SIZE

SHRINKAGE CAVITY = VOLUMETRIC PERCENT SHRINKAGE [SUMMATION OF ALL NODES IN CASTING STILL LIQUID]

STOP

Figure 23. Shrinkage Size Flow Chart
a heat conductance simulation the latent heat of fusion must be incorporated. This is accomplished by utilizing an effective specific heat which is greatly increased during the mushy stage of solidification (the effective specific heat utilized for this research was presented in the earlier derivation of nodal temperatures). A two-inch by eight-inch steel plate with a two and six-tenths inch by one and six-tenths inch riser was simulated. A time-temperature graph for two nodes is supplied in Figure 24. As can be seen, the latent heat of fusion dissipation is apparently simulated successfully. Also supplied in Figure 25 are isochronal solidification profiles for end of freeze contours. The time-temperature graph and the isochronal solidification profiles both indicate that the solidification simulation is reasonable.

Rigging Effects

It should be evident that the larger the number of riser-casting contact nodes, the longer the simulation run-time. The previous example utilized a riser two and six-tenths inches in width. Utilizing three nodes per inch, this then requires eight riser-casting contact nodes. The simulation for this design required approximately 1,400 iterations, and four minutes of CPU time. At that time all riser-casting contact nodes were solidified
Figure 24. Time-Temperature Graph For A 2" x 8" Low Carbon Steel Plate
Figure 25. Isochronal Solidification Profile Of A 2" x 8" Low Carbon Steel Plate
(i.e. all riser-casting contact nodes were at or below 2,640°F Fahrenheit. If the simulation were continued until the riser-casting contact nodes had reached a temperature of 2,000°F Fahrenheit, it would have taken 4,200 iterations and approximately ten minutes of CPU time. As previously stated, the width of the riser, along with the number of nodes per inch, dictate the number of riser-casting contact nodes. To demonstrate the impact of an increase in riser width with respect to computing time, the width of the riser was increased by one inch. The increased width added three additional riser-casting contact nodes for a new total of eleven. This in turn adds to the length of the solidification simulation as it will require an increase in the number of iterations for the complete riser-casting contact nodal area to solidify. The shrinkage cavity size should then be reduced since the mold wall, due to the increased number of iterations, will be able to extract more heat from the casting hence solidifying more nodes. Thus, increased riser size via the solidification simulation results in a decrease in shrinkage cavity size. One should note that the simulation is representing the riser-casting contact area remaining liquid longer, and thus the riser is able to supply liquid feed metal for an extended period of time.
The simulation of the increased width riser required approximately 2,000 iterations and five minutes of CPU time for the riser-casting contact nodes to solidify. This represents a forty-two percent increase in the number of iterations, with respect to the smaller riser, and a twenty-five percent increase in CPU time. The simulation was not performed to allow the riser-casting contact nodes to reach a temperature of 2,000 degrees Fahrenheit. However, assuming that a proportional relationship exists between the two designs, it would have taken 6,000 iterations and fifteen minutes of CPU time. As one can see, an increase in riser width has a large impact upon CPU time. Also, it should be evident that solidification simulation in general is expensive with respect to CPU time.

**Gradient Results**

For the two riser designs discussed the number of nodes still liquid after the riser-casting contact nodes had solidified decreased from sixteen to eight. This would be the shrinkage gradient for riser width for this particular riser and casting. The gradient, allowing an eight percent volumetric shrinkage, would be -.071 cubic inches per inch width of the riser. Depending upon the casting's overall width, height, and geometry, experience has shown this number can range from -.071 cubic inches per
inch to \(-.25\) cubic inches per inch. One can quickly see how gradients can be determined for each variable of width, height, and position, for each riser and/or chill. The best way to accomplish this would be to calculate the gradient at each iteration of the optimization routine. However, this would be impractical because of the excess CPU time. For a simple one riser-one chill design this was attempted. The freezing point was raised to 2,760 degrees Fahrenheit to quicken the solidification simulation. Figure 26 shows the reduction in the shrinkage cavity size as the optimization continues. One should remember that not only is the shrinkage cavity being reduced, but it is being reduced in a cost optimal fashion. The CPU time for this partial run was forty minutes. If the nodes were allowed to become completely solid and the optimization reaches its end, two-hundred minutes of CPU time could easily have been used. This would represent a cost of one thousand six-hundred dollars. Restrictions at OSU's Instruction and Research Computer Center make this impossible. A single run of this magnitude could require a one-week turn around time. Because this is unacceptable, shrinkage gradients were estimated based upon experience gained from earlier trial runs of simple geometries. The fact that shrinkage gradients were estimated rather than
Figure 26. Riser-Chill Dimension's Effect Upon Shrinkage Cavities Via Solidification Simulation

O = air boundary surrounding the mold
B = molding sand
R = riser
C = chill
Z = metal remaining liquid after the riser-casting contact has solidified

SHRINKAGE CAVITY SIZE = 0.800 cubic inches
Figure 26. (continued)
SHRINKAGE CAVITY SIZE = 0.516 cubic inches

O = air boundary surrounding mold
B = molding sand
R = riser
C = chill
Z = metal remaining liquid after riser casting contact
    nodes had solidified

Figure 26. (continued)
SHRINKAGE CAVITY SIZE = 0.3535 cubic inches

O = air boundary surrounding mold
B = molding sand
R = riser
C = chill
Z = metal remaining liquid after the riser-casting contact has solidified

Figure 26. (continued)
calculated was not felt to be detrimental to testing the optimization model, since the purpose of this research was not to determine the shrinkage gradients but to use them once they were obtained. The calculation of shrinkage gradients is an area of possible future research for this author.
Chapter V
PROBLEM ATTACK

The objective of this chapter is threefold: (1) to discuss the problem formulation, (2) to discuss the methodology which is utilized in solving the problem, and (3) to present the justification for this proposed methodology. Due to the length of this chapter, a review is given at the end.

Problem Formulation

As discussed in Chapter II, there are basically three major tools which are utilized to attain proper directional solidification of castings and thus eliminate shrinkage defects. These tools are risers, chills, padding, and combinations of each. Two of these, risers and chills, will be modeled in this research. Discussed in Chapter VII is a possible formulation involving the variable padding.

Some assumptions which were utilized in the formulation are as follows:

1. Two-dimensional solidification simulation adequately captures the casting solidification process. It was felt that a three-dimensional solidification simulation, although conceptually simple, would be too time consuming for the computer, as discussed
2. The objective function coefficients are determinable and the objective function is convex. One should note, as discussed later in this section, a quadratic function is utilized in the formulation.

3. A reasonable starting set of risers and chills can be selected by the foundry engineer.

4. Curved casting geometries can be approximated adequately by a series of rectangles.

5. Sand density is uniform throughout the mold, thus heat flux on similar cross sections should be nearly identical.

6. Chills are all external.

The first step in the solution methodology is for a foundry engineer to input the desired casting geometry. He then constructs a set of possible risers and chills. From this set, the algorithm will select which combination of risers and chills to utilize along with their dimensions and positioning. The following discussion of the formulation assumes that the possible number of risers and chills has already been determined.

Figure 27(a) exhibits a possible casting geometry input into the program. Figure 27(b) exhibits a possible set of risers and chills constructed by the foundry engineer.

It is important to model the width, height, and position of each riser and/or chill due to both their cost
Figure 27. Example Set Of Risers And Chills Constructed By Foundry Engineer For A Given Casting Geometry
impact and their solidification impact. Increasing the width of a riser or chill impacts foundry costs in the following areas:

1. Overall riser or chill size is increased, thus all costs associated with metal preparation for the casting rigging is increased.

2. In the case of a riser when the casting riser contact is increased both riser removal time and subsequent grinding time is increased nonlinearly (this cost is assumed quadratic).

In Figure 28 it can be seen that increased height of a chill or riser has a number of results. Some of these results are:

1. An increase in the sand required to produce the mold, possibly not a linear increase, as there is only a finite number of flask sizes. If a chemically bonded molding sand is utilized, a large increase in sand requirements, due to its high cost, could seriously impact the profitability of the casting.

2. A different squeezing pattern or pressure might be necessary to maintain proper mold hardness if a green sand process is utilized.

3. An increase in riser-chill height affects riser-chill size and all melting and handling requirements.

4. An increased mold size affects the handling ease of the mold.

5. An increase in difficulty in drawing the mold due to the increase in height. This may require larger draft angles.
Note increased flask size and sand requirement for increased riser size.

Figure 28. Riser And Chill Height Effects On Cost
Position of chills and risers are generally not considered in the literature to be large contributors to cost, however, the positioning of risers and chills could have an effect on cost, given certain casting geometries. For example:

1. Due to the casting geometry a riser placed close to other parts of the casting can increase the difficulty of riser removal. Figure 29 demonstrates the type of cost function necessary to keep the riser located in an area where it is more easily removed.

2. Figure 29 also demonstrates the differences in moldability depending upon the position of the riser and/or chill. The pocket created by close positioning of the riser to the casting would be difficult to draw.

3. A riser located near a large bulky area of a casting tends to keep the casting/sand interface at elevated temperatures and, in turn, increases the possibility of burn-on sand defects. (A burn-on sand defect is where metal actually penetrates into the mold wall forming a crusty layer of sand and metal that must be removed by expensive grinding operations.)

4. A chill located in some mold areas is more difficult to position by the molder.

Finally, the width and height interaction affects cost due to the increased size of either the chill or riser.

Highlighted above were some of the costs associated with the height, width, and position of risers and chills. Just as important is the impact of riser-chill width, height, and position on shrinkage cavity size. As the
Poor riser location due to small pocket; tough to remove and a hot spot.

Better design.

Figure 29. Cost Function With Respect To Position.
width and/or height increase, so does the bulk of the riser or chill. Naturally, the riser stays liquid longer and in turn has a higher probability of being able to provide feed metal to the casting during solidification. The chill on the other hand, due to its larger size, will remove heat faster from the casting. This allows less area that an opposing riser would need to service. This is demonstrated in Figure 30. In essence, if positioned properly a chill can increase a riser's effectiveness.

The advantage of position of a riser can be seen in Figure 31 for a particular casting geometry. Also shown is the advantage of proper chill placement, especially when a chill opposes a riser. The position affect is dictated by the casting geometry and the allowable feeding range of the riser servicing the particular area of the casting.

Discussed later in this chapter in the mathematical formulation is a variable representing the inclusion of a particular riser or chill in the solution. The inclusion of the riser or chill by itself has no effect on the shrink-age cavity size—only when it has mass could it have an impact. However, there are fixed costs associated with the introduction of the riser or chill. Some sources of these fixed costs for risers are:
Figure 30. Trade-Off Of Risers And Chills
Example 1

![Diagram of Example 1]

Example 2

![Diagram of Example 2]

Figure 31. Position Effect With Respect To Shrinkage Defects
1. The pattern shop must add the riser to the pattern.

2. In the case of a blind riser (i.e. one that is totally enclosed by the mold) and those metals forming a thin skin upon solidification, a core is used to break the top skin of the riser. This allows atmospheric pressure to assist the riser in feeding the casting. It actually allows the riser to feed areas of the casting greater in height than the riser itself. Obviously, there is a cost associated with both producing and positioning this core in the mold.

3. In the case of open risers (i.e. those risers that have their top surface open to the atmosphere) "hot-topping" is usually applied to the top of each open riser upon pouring. Hot-topping is a substance that helps keep the top of the riser liquid for a prolonged period of time. This then allows the riser to be more effective for its given size. There is a cost associated with both the purchase and application of the hot-topping.

Some sources for fixed costs associated with chills are:

1. In the case of a chill a pattern must be produced to make chills if they are manufactured in-house. If not manufactured in-house, they must be purchased.

2. Each chill must be manufactured, cleaned, and placed in the mold and, if possible, recovered during shake-out for later reuse. Obviously, there are costs associated with this.

The following is a list of variables utilized in the formulation:

\[ Z_i = \begin{cases} 0 & \text{if riser } i \text{ or chill } i \text{ is not included} \\ 1 & \text{if riser } i \text{ or chill } i \text{ is included} \end{cases} \]

\[ Z = \text{vector of } Z_i \text{ values} \]
$W_i$ = width or riser or chill $i$

$H_i$ = height of riser or chill $i$

$P_i$ = position on a given surface, as measured from a given reference point of riser or chill $i$

$X$ = vector of continuous variable values for width, height, and position for all risers

$= (W_1, H_1, P_1, W_2, H_2, P_2 \ldots W_n, H_n, P_n)$

$n$ = number of risers and chills

The following is a list of cost parameters utilized in the model:

$CZ_i$ = fixed cost of variable $Z_i$ (riser or chill $i$) being included in a proposed solution

$CWID_i$ = linear cost coefficient associated with width $i$

$CWID_{2i}$ = quadratic cost coefficient associated with width $i$

$CHGT_i$ = linear cost coefficient associated with height $i$

$CHGT_{2i}$ = quadratic cost coefficient associated with height $i$

$CPOS_i$ = linear cost coefficient associated with position $i$

$CPOS_{2i}$ = quadratic cost coefficient associated with width and height for variable $i$

$CWDPS_i$ = interaction cost coefficient associated with width and position for variable $i$

$CHTPS_i$ = interaction cost coefficient associated with height and position for variable $i$

$CWDHT$ = interaction cost coefficient associated with width and height for variable $i$
\[ C = \text{vector consisting of the linear cost components associated with } W_i, H_i, P_i \text{ for all } n \]
\[ = (CWID_1, CHGT_1, CPOS_1, CWID_2, CHGT_2, CPOS_2, \ldots, CWID_n, CHGT_n, CPOS_n) \]
\[ Q = (3n \times 3n) \text{ matrix consisting of the quadratic cost components associated with } W_i, H_i, \text{ and } P_i \text{ for all } n = \]

\[
\begin{array}{ccccccccc}
CWID_{1} & CWDHT_{1} & CWDPS_{1} & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
CWDHT_{1} & CHGT_{1} & CHTPS_{1} & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
CWDPS_{1} & CHTPS_{1} & CPOS_{1} & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & CWID_{2} & CWDHT_{2} & CWDPS_{2} & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & CWDHT_{2} & CHGT_{2} & CHTPS_{2} & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & CWDPS_{2} & CHTPS_{2} & CPOS_{2} & 0 & \ldots & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}
\]
The objective function then is:

\[
\text{Cost} = \sum_{i=1}^{n} \left[ \text{Cost Of Including Riser Or Chill } i \right] \left[ \text{Decision Variable For Including Or Not Including Riser Or Chill } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch Width Variable } i \right] \left[ \text{Width (Inches) Variable } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch }^2 \text{ Width Variable } i \right] \left[ \text{Width (Inches)}^2 \text{ Variable } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch Height Variable } i \right] \left[ \text{Height (Inches) Variable } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch }^2 \text{ Height Variable } i \right] \left[ \text{Height (Inches)}^2 \text{ Variable } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch Position Variable } i \right] \left[ \text{Position (Inches) Variable } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch }^2 \text{ Position Variable } i \right] \left[ \text{Position (Inches)}^2 \text{ Variable } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch }^2 \text{ Interaction Of Width and Position Variable } i \right] \left[ \text{Width (Inches) Variable } i \right] \left[ \text{Position (Inches) Variable } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch }^2 \text{ Interaction Of Width and Height Variable } i \right] \left[ \text{Width (Inches) Variable } i \right] \left[ \text{Height (Inches) Variable } i \right]
+ \sum_{i=1}^{n} \left[ \text{Cost/Inch }^2 \text{ Interaction Of Position And Height Variables } \right] \left[ \text{Position (Inches) Variable } i \right] \left[ \text{Height (Inches) Variable } i \right]
\]
Using previously defined notation:

Total Cost =

\[ \sum_{i=1}^{n} (CZ_i)Z_i \]

\[ + \sum_{i=1}^{n} (CWID_i)W_i \]

\[ + \sum_{i=1}^{m} (CHGT_i)H_i \]

\[ + \sum_{i=1}^{n} (CPOS_i)P_i \]

\[ + \sum_{i=1}^{n} (CWID2_i)W_i^2 \]

\[ + \sum_{i=1}^{n} (CHGT2_i)H_i^2 \]

\[ + \sum_{i=1}^{n} (CPOS2_i)P_i^2 \]

\[ + \sum_{i=1}^{n} (CWDHT_i)W_i H_i \]

\[ + \sum_{i=1}^{n} (CWDPS_i)W_i P_i \]

\[ + \sum_{i=1}^{n} (CHTPS_i)H_i P_i \]

Unless otherwise stated, all vectors are row vectors. The objective function in matrix format is then:

\[ \text{COST} = CZ' + BX' + XQX' \]

As in many mathematical formulations, there are limitations on available resources and performance standards that the solution to the mathematical formulation must satisfy. In particular, there are four types of constraints utilized in this formulation. They are: shrinkage, bounding, existence, and non-negativity. Each of these will be discussed in the sections that follow.
Shrinkage Constraint

The shrinkage constraint is one which makes this problem unique. Unfortunately, it is mathematically cumbersome to derive equations quantifying solidification and, in turn, shrinkage. Simulating solidification is mathematically simpler. However, as discussed in Chapter IV the simulation model has two major drawbacks. The first is that it is very time consuming on the computer. Even a simple solidification simulation can take over ten minutes of CPU time on very fast computers. A recent study on comparison of different simulation packages [40] claimed that a foundry must be prepared to spend anywhere from one hundred dollars to one thousand dollars per simulation. The second drawback is that it has a discrete size node it can simulate. As an example, if node lengths are one-fourth of an inch, a change of five-thousandths of an inch would not be detectable without altering the node length and in turn escalating solidification time even more. Because of this a more prudent method must be utilized to both reduce CPU time and approximate the discreteness in a continuous manner. To this end, a shrinkage gradient concept was introduced.

At the beginning of any run all chills and risers are allowed to be in the solution. The continuous variables
associated with each one are valued at an averaged value. Each variable, width, height, and position for each riser and chill is varied slightly. The difference in the shrinkage values divided by the difference in the variable values yields a first order approximation to the shrinkage gradient. It is:

\[
\text{SHRINKAGE GRADIENT} = \frac{\partial \delta}{\partial w} = \Delta \left( \frac{\text{ORIGINAL SHRINKAGE VALUE}}{\text{ALTERED SHRINKAGE VALUE}} \right) \left( \frac{w_i - (w_i + \Delta w_i)}{w_i} \right)
\]

Once this is done the maximum shrinkage is determined. This is accomplished by evaluating the potential shrinkage in the casting.

The potential shrinkage, or maximum shrinkage, is the shrinkage cavity size occurring without the aid of any risers or chills. This can be evaluated by running the solidification simulation without any risers or chills. Another less costly method of evaluating the potential shrinkage with respect to CPU time would be to multiply the total casting area by the solidification shrinkage factor. Thus, the shrinkage at any value for any riser-chill combination can be approximated by the following equation:
where: \( W_i' \) = value of width \( i \) 
\( H_i' \) = value of height \( i \) 
\( P_i' \) = value of position \( i \)

By performing further initial evaluations of the solidification simulation, a second-order approximation could be determined. This can be accomplished by again varying each dimension for each riser and chill and constructing an approximation to the Hessian of the shrinkage equation. The second-order approximation would be as follows:

\[
g(X) \equiv \text{MAXIMUM SHRINKAGE} + \sum_{i=1}^{n} \left[ \int_{0}^{W_i'} \frac{\partial g}{\partial W_i} \, dW_i + \int_{0}^{H_i'} \frac{\partial g}{\partial H_i} \, dH_i + \int_{0}^{P_i'} \frac{\partial g}{\partial P_i} \, dP_i \right] 
\]

Similarly, for height and position the second partials for width, height, and position can be estimated as follows:
ACCELERATION OF SHRINKAGE CHANGE FOR WIDTH \( i \) = \( \frac{\partial^2 g_i}{\partial W_i^2} \Delta = \left( \frac{\text{SHRINKAGE GRADIENT 1 \ - \ GRADIENT 2}}{\text{FOR WIDTH} \ i} \right) \frac{1}{W_i - (W_i + 2\Delta_i)} \)

\[
\frac{\partial g_1}{\partial W_i} - \frac{\partial g_2}{\partial W_i} = \frac{-2\Delta_i}{-2\Delta_i}
\]

Finally, the shrinkage equation could be represented by:

\[
g(X) = \text{MAXIMUM SHRINKAGE} + \sum_{i=1}^{n} \left[ \int_0^{W_i'} \frac{\partial g_i}{\partial W_i} dW_i + \int_0^{H_i'} \frac{\partial g_i}{\partial H_i} dH_i + \int_0^{P_i'} \frac{\partial g_i}{\partial P_i} dP_i \right] \\
+ \sum_{i=1}^{n} \left[ \int_0^{W_i'} \int_0^{W_i'} \frac{\partial^2 g_i}{\partial W_i^2} dW_i dW_i + \int_0^{H_i'} \int_0^{H_i'} \frac{\partial^2 g_i}{\partial H_i^2} dH_i dH_i + \int_0^{P_i'} \int_0^{P_i'} \frac{\partial^2 g_i}{\partial P_i^2} dP_i dP_i \right]
\]
One may note that the approximation shown is not a complete approximation to the Hessian. Further varying of the solidification model could be carried out to determine interaction accelerations. If this were done a Taylor Series approximation could be utilized.

Finally, the shrinkage constraint is represented as:
\[ g(\text{shrinkage}) \leq \text{shrinkage tolerance}. \]
The shrinkage tolerance is the amount of porosity allowed in the casting and is user-specified. This will vary from zero to the maximum allowable shrinkage value, depending upon the casting's purpose.

**Bounding Constraint**

Bounding constraints are simple upper bounds on variable values due to inherent casting geometry or flask limitations. Using the following parameters:

- \( \text{WIDTHBND}_i \) = maximum allowable width for variable \( i \) (inches)
- \( \text{HGTBND}_i \) = maximum allowable height for variable \( i \) (inches)
- \( \text{WIDPSBND}_i \) = maximum allowable distance over which riser of width \( W_i \) can be positioned
- \( \text{SHRINKTOL} \) = allowable shrinkage in casting
- \( M \) = very large number, greater than the sum of the largest \( \text{WIDTHBND}_i \), \( \text{HGTBND}_i \) and \( \text{WIDPSBND}_i \)
The constraints then are:

\[
\begin{align*}
\text{WIDTH VARIABLE}_i & \leq \text{MAXIMUM ALLOWABLE WIDTH FOR } i \\
\text{HEIGHT VARIABLE}_i & \leq \text{MAXIMUM ALLOWABLE HEIGHT FOR } i \\
\text{WIDTH VARIABLE}_i + \text{POSITION VARIABLE}_i & \leq \text{MAX ALLOWABLE DISTANCE RISER OR CHILL } i \text{ CAN BE LOCATED OVER} \\
W_i & \leq \text{WIDTHBND}_i \quad i=1,2\ldots n \\
H_i & \leq \text{HEIGHTBND}_i \quad i=1,2\ldots n \\
W_i + P_i & \leq \text{WIDPSBND}_i \quad i=1,2\ldots n
\end{align*}
\]

Existence Constraint

As will be discussed, the overall problem will be decomposed. Because of this there must be constraints which prevent a chill or riser from having mass if it is not in the solution set. Thus, the following constraints are included:

\[
\begin{align*}
\text{WIDTH}_i + \text{POSITION}_i + \text{HEIGHT}_i & \leq \text{VERY LARGE NUMBER IF RISER OR CHILL } i \text{ IS INCLUDED, } i=1,2\ldots n \\
\text{WIDTH}_i + \text{POSITION}_i + \text{HEIGHT}_i & \leq \text{IF RISER OR CHILL } i \text{ IS NOT INCLUDED, } i=1,2\ldots n
\end{align*}
\]

Since \( Z_i \) is a zero-one variable, having the value zero if riser or chill \( i \) is not included and having the value 1 if it is, then the constraints can be formulated as follows:

\[
W_i + P_i + H_i \leq MZ_i \quad i=1,2\ldots n
\]
Non-Negativity Constraints

These constraints are simply that all continuous variables must have a positive value and that discrete values must have the value zero or one. Thus:

\[ W_i > 0 \quad i=1,2,...,n \]
\[ H_i > 0 \quad i=1,2,...,n \]
\[ P_i > 0 \quad i=1,2,...,n \]
\[ Z_i = 0 \quad i=1,2,...,n \]

Table 1 indicates the summary of the formulations.

Solution Methodology

As discussed previously, the foundry engineer must perform two tasks: (1) he must input the casting geometry, and (2) he must formulate a set of possible risers and chills. Input of the casting geometry is described in Appendix E. As shown in the flow chart in Figure 32, the algorithm selects different combinations of risers and chills from the subset chosen by the foundry engineer, and for each combination chosen the algorithm determines the optimal width, height, and position so as to produce a shrinkage defect-free casting.

Thus, there are basically two optimization problems to solve. First, selection of the combinations of risers and chills, and second how to best optimize the continuous
Table 1. Complete Problem Formulation

This Table gives a complete variable description of the mathematical formulation proposed. All variable, cost coefficient and parameter definitions and descriptions can be found in the previous discussion.

\[
\text{minimize Cost} = \sum_{i=1}^{n} (CZ_i)Z_i + \sum_{i=1}^{n} (CWI_1)W_i + \sum_{i=1}^{n} (CWI_2)W_i^2 \\
+ \sum_{i=1}^{n} (CHG_1)H_i + \sum_{i=1}^{n} (CHG_2)H_i^2 \\
+ \sum_{i=1}^{n} (CP_1)P_i + \sum_{i=1}^{n} (CP_2)P_i^2 \\
+ \sum_{i=1}^{n} (CW_1)W_iH_i \\
+ \sum_{i=1}^{n} (CW_2)W_iP_i \\
+ \sum_{i=1}^{n} (CHP_1)H_iP_i
\]

subject to:

\[g(X) \leq \text{SHRINKTOL}\]

\[W_i \leq \text{WIDTHBND}_i \quad i=1,2,...,n\]

\[H_i \leq \text{HEIGHTBND}_i \quad i=1,2,...,n\]

\[W_i + P_i \leq \text{WIDPSBND}_i \quad i=1,2,...,n\]

\[W_i + H_i + P_i \leq (M)Z_i \quad i=1,2,...,n\]

\[W_i \geq 0 \quad H_i \geq 0 \quad P_i \geq 0 \quad i=1,2,...,n\]

\[Z_i = 0,1 \quad i=1,2,...,n\]
Figure 32. General Optimization Procedure Flow Chart

1. Input casting geometry
2. Input risers and chill possibilities
3. Select best alternative
4. Determine minimum cost for alternative along with the respective widths, heights, and positions

- Optimal
  - No
  - Yes: Stop
variables of the chosen combination. Each of these subjects are addressed below.

**Optimal Selection Of Combinations**

A decomposition of the problem is the approach taken. It is clear that once a set of discrete values is obtained, the problem simplifies to one containing only continuous variables. The problem then is to be able to move systematically through the various combinations so as to arrive at the optimal one without complete enumeration. Complete enumeration of all solutions would require $2^n$ subproblems to be solved. Due to the potential of a large number of subproblems, Generalized Benders' Decomposition (GBD) was utilized with a slight modification. The development presented in this section follows closely that found in Geoffrion [41, 42].

The problem to be decomposed is of the following form:

$$\begin{align*}
\min_{x,y} & \quad f(x,y) \\
\text{subject to:} & \quad G(x,y) \leq 0 \\
& \quad x \in X \\
& \quad y \in Y
\end{align*}$$

where $y$ is a vector of complicating variables, in this case all 0-1 integer variables.

The key concept is that of projection. Projection [43], sometimes known as partitioning, is a device which
takes advantage in certain problems of the relative simplicity resulting when certain variables are temporarily fixed in value. The projection of $P_1$ onto $y$ then is:

$$\min_{y} \ v(y) \ \text{subject to } y \in Y \cap V \quad (5-1)$$

where $v(y) = \infimum f(x,y) \ \text{subject to } G(x,y) \leq 0 \ \forall x \in X \quad (5-2)$

and $V = \{ y : G(x,y) \leq 0 \ \text{for some } x \in X \} \quad (5-3)$

Note, $v(y) = \text{optimal value of original problem, } P_1, \text{ for a fixed } y$.

Again, the advantage is that evaluating $v(y)$ is much easier than evaluating $P_1$. In this riser-chill problem it is equivalent to evaluating only the widths, heights, and positions of each riser and/or chill for a chosen combination. The set $V$ consists of those combinations of risers and chills for which feasible sizes and locations exist. If a particular combination of risers and chills are at their maximum sizes and are strategically placed and the casting still has a shrinkage defect, then that combination would not be in $V$. Thus, the set $Y \cap V$ is the set of $y$ yielding a feasible solution $P_1$.

As proven by Geoffrion [41], if $(x^*, y^*)$ is optimal in $P_1$, then $y^*$ must be optimal in (5-1). If $y^*$ is optimal in (5-1) and $x$ achieves the infimum in (5-2) with $y = y^*$, then $(x^*, y^*)$ is optimal in $P_1$.

As mentioned before, it is possible that not all combinations of risers and chills chosen have feasible
solutions. If one of the previously discussed shrinkage function approximations is utilized it is trivial to identify those combinations infeasible—simply evaluate the functional approximation for each combination. However, if the solidification simulation techniques are sped up adequately so that functional approximations are no longer required, one could not easily identify those feasible combinations initially. One could determine which riser-chill combinations are infeasible by enumerating each combination. Then, for each riser-chill combination, the respective risers and chills would be strategically placed and at their maximum dimensions. The casting with this rigging design would then be simulated. If the solidification simulation results indicated a shrinkage defect, the respective combination would be removed from V. Even if solidification simulation techniques were sped up tenfold, this approach would still require considerable CPU time. It also would be inefficient as a certain amount of knowledge should be obtained from each combination evaluated. That is, the results from one combination should be able to indicate other combinations that are either feasible or infeasible. What would then be desirable would be a strategic plan to evaluate a number of riser-chill combinations initially to determine whether or not other combinations are infeasible. One possible strategy
could be to simulate all combinations including all chills and only one riser. If any of these combinations were found to be infeasible, clearly any combination including only that riser and any chill combination would also be infeasible. This strategy could be extended to test all combinations of two risers and so on. None of the proposed strategies, short of complete enumeration, guarantee that all infeasible solutions are identified initially. Since infeasible combinations cannot be identified initially, it would be advantageous to employ some scheme for identifying combinations not in \( V \) throughout the algorithm. Geoffrion [41] presents a theorem which constructs constraints to determine combinations not in \( V \). The theorem is stated below.

Theorem 5.1 (V-Representation) Assume that \( X \) is a non-empty convex set and \( G \) is convex on \( X \) for each fixed \( y \in Y \). Assume also that the set

\[
Z_y = \left\{ z \in \mathbb{R}^m : G(x, y) \leq z \text{ for some } x \in X \right\}
\]

is closed for each fixed \( y \in Y \). Then the point \( \bar{y} \in Y \) is also in the set \( V \) if and only if \( \bar{y} \) satisfies the infinite system

\[
\inf_{x \in X} \lambda^t G(x, y) \leq 0 \quad \forall \lambda \in \Lambda
\]

where: \( \Lambda = \left\{ \lambda \in \mathbb{R}^m : \lambda_i \geq 0 \text{ and } \sum_{i=1}^{m} \lambda_i = 1 \right\} \)
The above theorem, although important to a generalized application, is not important in this case due to the fact that infeasibility is handled separately because of the unique problem structure. A discussion of this is found later in this section.

Another important concept is that of V-representation.

Theorem 5.2: Geoffrion [41] Assume \( X \) is a non-empty convex set and that \( f \) and \( G \) are convex on \( x \) for all \( y \) \( Y \). Assuming that for each \( y \in Y \cap V \), \( v(y) \) is finite and possesses an optimal multiplier vector, then by duality

\[
v(y) = \sup_{u \geq 0} \left[ \inf_{x \in X} f(x, y) + u^t G(x, y) \right]
\]

for all \( y \in Y \cap V \).

The proof found in [41] is based on nonlinear duality theory, in particular primal-dual equality. This theorem is more relaxed than some shown in other articles regarding Benders' Decomposition. Usually included is a possibility of a primal solution being unbounded. This, however, is impossible for the proposed problem due to positive cost coefficients and non-negativity requirements on each variable.

Utilizing theorems 5.1 and 5.2 the original problem can be constructed as:

\[
\text{minimize } y \in Y \left[ \sup_{u \geq 0} \left( \inf_{x \in X} f(x, y) + u^t G(x, y) \right) \right]
\]
subject to \[ \inf_{x \in X} \lambda^t G(x, y) \leq 0 \quad \forall \lambda \in \Lambda \]

where \( \Lambda = \{\lambda \in \mathbb{R}^m : \lambda \geq 0 \text{ and } \sum_{i=1}^{n} \lambda_i = 1\} \)

or, after rewriting the problem using the definition of supremum as the smallest upper bound and letting \( y_0 \) represent that upper bound one gets:

minimize \( y_0 \)

subject to: \( y_0 \geq \inf \left[ f(x, y) + u^t G(x, y) \right] \quad \forall u \geq 0 \) \hspace{1cm} (5-7)

\[ 0 \geq \left\{ \inf \left[ \lambda^t G(x, y) \right] \right\} \quad \forall \lambda \in \Lambda \] \hspace{1cm} (5-8)

It is important to note that this formulation may generate a large number of constraints, perhaps an uncountably infinite number. The \( u^t \) represent extreme points if it were linear in \( x \), while \( \lambda^t \) would represent all extreme rays.

However, it is supposed that in the optimal solution only a small number of constraints will be binding. Thus, relaxation was suggested as the best, perhaps only, way to proceed. The problem then is twofold: first, how a solution to a relaxed version can be tested for feasibility with respect to the ignored constraints, and second how a violated constraint is generated. First, any feasible solution must satisfy (5-8). Constraints (5-7) are satisfied from Theorem 5.2 if the solution to \( v(\tilde{y}) \) is
feasible and if and only if $\hat{y}_o > v(y)$. Due to the subproblem being infeasible, violated constraints need not be generated explicitly as they are accounted for by a fathoming procedure discussed later in this section. However, if the subproblem is feasible, violated constraints must be explicitly generated. The method suggested by Geoffrion [41, 42] is to generate the most violated constraint. The constraint is formed utilizing the dual variable from the subproblem solution. Thus, it is important that the methodology utilized in the subproblem optimization be dual-adequate.

The final concern is that of convergence. For this problem $Y$ is a finite discrete set, each element representing a combination of risers and chills. The following theorem taken from Geoffrion [41] addresses convergence:

**Theorem 5.3:** Assume $y$ is a finite discrete set, that theorem 5.1 holds and theorem 5.2 holds. Then, the generalized Benders' decomposition procedure terminates in a finite number of steps for any $\epsilon > 0$ or even for $\epsilon = 0$.

The algorithm applied to the casting problem is similar to the previous description except for the method for representing the set $V$. Due to the structure of the problem, if a particular combination of risers and chills cannot be sized and placed to satisfy shrinkage requirements (i.e. is an infeasible solution), then this can lead to assuredness of the infeasibility of other combinations. As an
example see Figure 33. It is shown that if a single riser and three chills yield an infeasible solution, then surely a single riser and one chill yield an infeasible solution. In this way, certain combinations are fathomed.

Finally, the following algorithm is utilized to select combinations of risers and/or chills. (See the flow chart, Figure 34.)

Algorithm:

1. All $z_i = 1$ for $i=1,2,...,n$, that is, allow all risers and chills to be in solution set
   a. set $\text{UBD} = \infty$
   b. $P = 1$

2. Solve subproblem

3. If subproblem is infeasible, stop; there are no feasible combinations. Else set $\text{UBD} =$ subproblem and continue.

4. Construct master problem
   
   minimize $y_0$
   
   $y \in \mathcal{Y}$
   
   subject to: $y_0 \geq \inf \left[ f(x,y) + u^j G(x,y) \right]_{j=1,2,..,p}$

5. If $\text{UBD} = Z_0$ stop $\Rightarrow$ optimal solution

6. Is subproblem solution $\text{UBD}$? Yes $\Rightarrow$ $\text{UBD} =$ subproblem solution.

7. Solve new subproblem.

8. If solution to subproblem is infeasible fathom appropriate combinations and return to 4, else return to 4.
The following combinations are also infeasible:

Figure 33. Fathoming of Infeasible Combinations
Figure 34. Benders' Flow Chart
The final theoretical consideration is that of how to evaluate the explicit form of the constraint to add to the master problem each iteration. It is generally unlikely, according to Geoffrion [41], that one could deal with functions in their infimal value representation. The final requirement is then that Pi have the following property, as stated by Geoffrion [41], Property P. This property is:

For any \( u \geq 0 \) the infimum of \( f(x,y) + u^t(x,y) \) over \( X \) can be taken essentially independent of \( y \) so that the function can be expressed explicitly. Thus,

\[
\inf_{x \in X} [f(x,y) + uG(x,y)] = f(x_u,y) + uG(x,y)
\]

The most important fact is that Property P holds for functions linearly separable in both \( f \) and \( G \), (stated in Geoffrion [41]. As one can see below, the structure of both the objective function and the constraint set is linearly separable.

As stated earlier, the objective function is:

\( f(x,y) = CZ' + BX + XQX' \) which is clearly linearly separable in \( X \) and \( Z \).

The constraints are all in \( X \) only with one exception—the existence constraint set. This set is \( W_i + H_i + P_i - MZ_i \leq 0 \ i=1,2,...,n \). Clearly, this set of constraints is linearly separable in \( X \) and \( Z \). Thus, the original problem satisfies Property P.
The constraint to be added for this problem is as follows:

\[ y_0 \leq f^j(X) + \sum_{i=1}^{n} z_i \left( CZ_i - M \left( \text{Dual For Existence Constraints} \right) \right)^j \]

where:

- \( f^j(X) \) = subproblem solution without the fixed costs of \( Z_i \)'s included in the solution
- \( j = \) iteration of master problem

the master problem then is:

minimize \( y_0 \)

subject to: \( y_0 \leq f^j(X) + \sum_{i=1}^{n} z_i \left( CZ_i - M \left( \text{Dual For Existence Constraints} \right) \right)^j \)

\( j=1,2,\ldots,p \)

For the problem tested in this research the master problem is solved by complete enumeration due to the relatively low number of combinations. If more combinations were included a branch and bound technique or some other partial enumeration scheme could be implemented.

In the algorithm above the initial feasible solution is assured by allowing all risers and chills to be included. Obviously, this is the best choice of obtaining an initial
feasible solution. Appendix A gives an example problem of riser-chill selection using the modified Benders approach previously discussed.

Subproblem Optimization

In this section optimization of the subproblem is discussed. After a combination of the (0-1) variables has been selected, the subproblem is one in continuous variables only. The constraints are all bounding constraints with the exception of the shrinkage constraint which is approximated by either a first or second-order approximation. The objective function is quadratic. The restriction of the objective function being quadratic is not binding, although its being convex is. The subproblem then is:

\[
\text{minimize } CX' + XQX'
\]

subject to \( g(X) \leq \text{shrink to } L \)

\[
\text{for } x \in X
\]

where \( X \) has bounds for each variable and non-negativity requirements.

Because of the decomposition's methodology requirement that the subproblem solution be dual adequate, the Kuhn-Tucker conditions were explicitly constructed. These conditions are shown in Table 2. There are two categories of Kuhn-Tucker conditions, the first being the variable differential conditions and the second being constraint conditions. When a variable is not valued at its upper bound, the
Table 2. Kuhn-Tucker Conditions

In this table the Kuhn-Tucker conditions are specified. For ease of later discussion, the conditions are divided into two categories: (1) differential conditions and (2) constraint conditions.

Differential Kuhn-Tucker Conditions

\[
\begin{align*}
\text{CWID}_i + 2(\text{CWID}_2)_iW_i + \text{DWIDBD}_i &+ \frac{3q}{qW_i}\text{DUALSHRNK} + \text{DEXIS}_i - \text{DWID}_i = 0 \quad i=1,2,...,n \\
\text{CHGT}_i + 2(\text{CHGT}_2)_iH_i &+ \text{DHGTBD}_i \\
+ \frac{3q}{qH_i}\text{DUALSHRNK} + \text{DEXIS}_i - \text{DHGT}_i = 0 \quad i=1,2,...,n \\
\text{CPOS}_i + 2(\text{CPOS}_2)_iP_i &+ \frac{3q}{qP_i}\text{DUALSHRNK} + \text{DEXIS}_i - \text{DPOS}_i = 0 \quad i=1,2,...,n
\end{align*}
\]

Where:

- \(\text{DWIDBD}_i\) = dual variable associated with the upper bound constraint for width in variable \(i\)
- \(\text{DHGTBD}_i\) = dual variable associated with the upper bound constraint for height in variable \(i\)
- \(\text{DWIDPSBD}_i\) = dual variable associated with the upper bound constraint for both width and position to maintain correct relative position with respect to the casting for variable \(i\)
- \(\text{DEXIS}_i\) = dual variable associated with the existence constraint for variable \(i\)
Table 2 (continued)

DUALSHRNK = dual variable associated with the shrinkage constraint

\[ \text{DWID}_i = \text{dual variable associated with the non-negativity constraint for width variable } i \]

\[ \text{DHGT}_i = \text{dual variable associated with the non-negativity constraint for height variable } i \]

\[ \text{DPOS}_i = \text{dual variable associated with the non-negativity constraint for position variable } i \]

Constraint Kuhn-Tucker Conditions

\[ \text{DUALSHRNK} (g(X) - \text{SHRINKTOL}) = 0 \]

\[ \text{DWID}_i (W_i - \text{WIDTHBND}_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ \text{DHGT}_i (H_i - \text{HEIGHTBND}_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ \text{DWIDPSBD}_i (W_i + P_i - \text{WIDPSBND}_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ \text{DEXIS}_i (W_i + H_i + P_i - (M)Z_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ \text{DWID}_i (-W_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ \text{DHGT}_i (-H_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ \text{DPOS}_i (-P_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ (g(X) - \text{SHRINKTOL}) = 0 \]

\[ (W_i - \text{WIDTHBND}_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ (H_i - \text{HEIGHTBND}_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ (W_i + P_i - \text{WIDPSBND}_i) = 0 \text{ } i=1,2,\ldots,n \]

\[ (W_i + H_i + P_i - (M)Z_i) = 0 \text{ } i=1,2,\ldots,n \]
<table>
<thead>
<tr>
<th>Variable</th>
<th>Condition</th>
<th>Index Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUALSHRNK</td>
<td>$&gt; 0$</td>
<td></td>
</tr>
<tr>
<td>DWIDBD&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>DHGTBD&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>DWIDPSBD&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>DEXIS&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>DWID&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>DHGT&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>DPOS&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>W&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>H&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
<tr>
<td>P&lt;sub&gt;i&lt;/sub&gt;</td>
<td>$&gt; 0$</td>
<td>$i=1, 2, ..., n$</td>
</tr>
</tbody>
</table>
dual variable corresponding to that bound must be zero to satisfy its constraint condition. Similarly, when a variable is not at its lower bound, the dual variable corresponding to that bound must also be zero. As one can see, to then satisfy the differential condition for that particular variable the dual variable corresponding to the shrinkage constraint must act alone. Thus, when a variable is not valued at its bound, either upper or lower, the value of the dual variable corresponding to the shrinkage constraint is of great importance.

If one knew initially those variables at the optimal solution not at their bounds the Kuhn-Tucker conditions could be satisfied directly. To accomplish this one would construct a system of linear equations from the Kuhn-Tucker differential conditions representing those variables not at their bounds and then simply solve the system of linear equations. The system of equations is shown below.

\[
\begin{align*}
2(CWID_2_i)W_i &+ \frac{\partial g}{\partial W_i} (DUALSHK) = -CWID_i \\
&\forall W_i \neq 0 \text{ and } W_i \neq WIDBND_i \\
2(CHGT_2_i)H_i &+ \frac{\partial g}{\partial H_i} (DUALSHK) = -CHGT_i \\
&\forall H_i \neq 0 \text{ and } H_i \neq HGTBND_i \\
2(CPOS_2_i)P_i &+ \frac{\partial g}{\partial P_i} (DUALSHK) = -CPOS_i \\
&\forall P_i \neq 0 \text{ and } W_i + P_i \neq WIDPSBND_i
\end{align*}
\]
\[
\sum_{i \in C} \frac{\partial g}{\partial p_i} p_i + \sum_{i \in C} \frac{\partial g}{\partial w_i} w_i + \sum_{i \in C} \frac{\partial g}{\partial h_i} h_i = \text{SHRINKDIFF}
\]

where \( C \) = set of indexes for \( w_i, h_i, \) and \( p_i \) included.

Obviously, the problem is then selecting which variables should be at their upper bounds, which at their lower, and which at neither upper or lower.

The determination of the combination of variables not at their bounds satisfying the Kuhn-Tucker conditions could be guaranteed by total enumeration of all possible variable combinations. Figure 35 shows an enumeration scheme to ensure that every combination is examined. Also, a lexicographic enumeration scheme could be employed to ensure every combination is considered and in turn that the optimal solution, if it exists, is found. However, either approach could require for a subproblem with eighteen risers and/or chills, more than thirty-eight million variable combinations to be considered. Although this algorithm guarantees an optimal solution, if one in fact exists, it could tend to become numerically burdensome.

Due to enumeration difficulties a search technique was required. It is clear that if a linear approximation to the shrinkage equation is utilized, the problem is simply one of a quadratic objective function and linear constraints. Efficient algorithms such as gradient projection, reduced gradient, etc. exist for solving this problem. They were
Note:  
L = variable set at lower bound  
M = variable set at neither bound  
U = variable set at upper bound

Figure 35. Enumeration Tree
not implemented because if the solidification simulation were utilized an explicit expression for the shrinkage equation would not be available. It should also be remembered that a goal of this research was to develop an optimization technique that could interact with the solidification simulation. To this end a penalty method was utilized.

A penalty method was incorporated with an unconstrained nonlinear programming technique to arrive at the optimal solution. In this approach a positive term is added to the objective function, in the case of minimization, which prescribes a high cost for constraint violation. A parameter is used to determine the severity of the penalty and, in turn, how well the unconstrained problem approximates the constrained one. As the parameter approaches infinity the approximation becomes increasingly accurate. As the parameter [44] is increased to yield a good approximation to the constrained problem the corresponding structure of the unconstrained problem becomes increasingly unfavorable, thereby slowing the convergence rate of many algorithms.

Initially the plan of this research was to use the solidification simulation whenever a value to the shrinkage equation was required. This approach was clearly impractical as the simulation required excess computing time. The decision to use a penalty method as opposed to a barrier
method was made with the hope that the solidification model could be later sped up sufficiently to be utilized as planned. Since it starts infeasible, a penalty method would use less computing time with the solidification simulation than would a barrier method which is always feasible. As demonstrated in Figure 36, this is due to the fact that an infeasible solution may have only six riser-casting contact nodes, while a feasible solution may have sixteen. The speed of solidification simulation is greatly affected by the number of riser-casting contacts. Hence a penalty method should lower solidification simulation time.

A penalty method transforms a constrained problem into an unconstrained one as shown below.

\[
\begin{align*}
\text{minimize } & f(x) \\
\text{subject to } & g(x) \leq 0 \quad x \geq 0 \\
\text{minimize } & f(x) + u_k \left[ (g(x))^2 \right] \\
k = \text{iteration number}
\end{align*}
\]

where \( m_k = \text{a penalty value increasing each iteration.} \)

A flow chart of the algorithm used to solve this problem is shown in Figure 37. Later in this chapter the specifics of the application to the Benders subproblem will be given.

The following is a general discussion [44, 45, 46] of the convergence properties of penalty methods. The method defines a sequence \( \{ u_k \} \), \( k = 1, 2, \ldots, n \) tending toward infinity
Figure 36. Effect Of Number Of Riser-Casting Nodes On Solidification Simulation Time

Note: Each box represents two nodes
Figure 37. Penalty Method Flow Chart

PROBLEM IS INFEASIBLE

IS PENALTY TOO LARGE?

FEASIBLE

INCREASE PENALTY

OPTIMIZE $f(x) + u(g(x))^2$

SET PENALTY AT STARTING VALUE

OPTIMAL

yes

no
for each $k$ where $u_k \geq 0$ and $u_k + 1 > u_k \forall k$ for the following problem:

$$\begin{align*}
&\text{minimize } f(x) \\
&\text{subject to } g(x) \leq 0
\end{align*}$$

define the function

$$P(x) = [\max(0, g(x))]^2$$

$$q(u,x) = f(x) + uP(x)$$

$u=$penalty

Then for each $k$ solve the unconstrained problem:

$$\begin{align*}
&\text{minimize } q(u_k,x) \\
&\text{obtaining a solution point } x^k.
\end{align*}$$

So as to show convergence two Lemmas are required.

Lemma 1 follows and can be found in Luenberger [44]:

Lemma 1:

$$q(u_k,x_k) \leq q(u_{k+1},x_{k+1})$$

$$P(x_k) \geq P(x_{k+1}) \quad f(x_k) \leq f(x_{k+1})$$

The second Lemma and its proof is also found in Luenberger [44].

Lemma 2: Let $x^*$ be a solution to the problem. For each $k$:

$$f(x^*) \geq q(u_k,x_k) \geq f(x_k)$$

The following theorem (see [44, 45, 46]) states the convergence of penalty methods in general:
Theorem 4.4: Let \( \{x_k\} \) be a sequence generated by the penalty method. Then, any limit point of the sequence is a solution to the original problem.

The previous discussion should indicate that the penalty method guarantees an optimal solution if:

1. \( f \) is continuous
2. \( g \) is continuous
3. for each penalty update the unconstrained problem has a solution

All of these conditions are met in the riser-gating subproblems. Discussed in Chapter VI are the convergence properties of different penalty update factors, different initial penalty values, and other factors affecting the performance of the algorithm.

As was previously discussed with regard to Benders' Decomposition, infeasible solutions to subproblems must be identified. The penalty size indicates whether or not a specific subproblem is infeasible. The assumption is if the penalty is large and the problem values are near their bounds it is assumed infeasible. Because an approximation to the shrinkage equation is utilized, all infeasible combinations could be determined initially. The reader is cautioned, however, that the problem is formulated and implemented with the hope that a more efficient solidification simulation can be developed.
As discussed in the first section of this chapter, there are four classifications of constraints which must be satisfied. The penalty method utilized in this application controls only one—the shrinkage constraint. The other constraints are maintained feasible by the line search which will be discussed later. The problem could have been formulated with a number of penalties, but since the constraints, other than the shrinkage constraint, were simple bounds they were more easily controlled via the line search. The penalty function converts the constrained subproblem to one of an unconstrained subproblem. However, the task of minimizing the unconstrained problem is still before us. Again, consider the following problem:

$$\text{minimize } f(x) + uP(x)$$

where

$$P(x) = \left[ \max \left( \text{SHRINKAGE VALUE, 0} \right) \right]^2$$

There are a number of unconstrained optimization methods one could utilize to solve the problem. However, the fundamental underlying concept to any of the algorithms is to select an initial starting point using a fixed rule, determine a direction of movement, and move along a line in that direction to a minimum value of the objective function. At the new point a new direction is determined and the process is continued. The basic difference among most algorithms is determination of the direction of movement. One basic idea of many algorithms is to move along the
negative direction gradient. The negative gradient for
minimization problems points in the direction of the great­
est instantaneous decrease in the objective function. 
Show below is an example of an algorithm incorporating
the gradient $\nabla f(x)$. It is termed Steepest Descent [44, 45, 
46, 47].

1. determine $X_0$, $\beta = f(x_0)$, $k=0$
2. determine $\nabla f(x_k)$
3. determine direction $d_k = -\nabla f(x_k)$
4. determine minimum $\{f(x_k) + \alpha d_k\}_{\alpha \geq 0}$
5. $x_{k+1} = x_k + \alpha d_k$
6. $\gamma = f(x_{k+1})$
7. optimality check $|\beta - \gamma| < \varepsilon$ 
   \begin{cases} 
   \text{yes stop answer optimal} \\
   \text{no } \beta = f(x_{k+1}) \text{ return to 2} \\
   k = k+1 
   \end{cases}

Many algorithms incorporate the gradient in some
manner in the determination of the search direction.
Basically, the search direction incorporating a gradient
is analogous to approximating the objective function by a
series of linear approximations. Some other algorithms use
a second-order approximation if the second partials are
readily available. As a general description of these
approaches a quadratic approximation of $f(x)$ can be made
via the Taylor series neglecting third-order terms and
higher. The direction is obtained by differentiating the Taylor series approximation. Then, in a way similar to Steepest Descent, a line search for a minimum is carried out. This direction, utilizing a second-order approximation, can be derived by examining the Taylor series approximation neglecting third-order terms and higher.

\[ f(x_{k+1}) = f(x_k) + \nabla f(x_k)(x_{k+1} - x_k) + \frac{1}{2}(x_{k+1} - x_k) Q(x_{k+1} - x_k) \]

the derivative \( f(x) \) with respect to \( \Delta x_k \) is

\[ \Delta x_k = -[F(x_k)]^{-1} \nabla f(x_k) \]

the direction of the line search then is

\[ d = [F(x_k)]^{-1} \nabla f(x_k) \]

where: \( F(x_k) = \text{Hessian} \)

Algorithms of this general type are termed Newton's Methods. Figure 38 illustrates the differences between the Steepest Descent and Newton algorithm on a small two-dimensional problem.

The advantage of Steepest Descent is that it is simple to implement, only the gradient need be known. Newton's Method has faster convergence, however, it requires the inverse Hessian to be computed each iteration in the non-quadratic case. Thus, it seems reasonable that a nice compromise would be an algorithm that uses an approximation
Figure 38. Comparison Of Steepest Descent And Newton's Method Search Direction
to the Hessian but does not require its update each iteration. An algorithm which does this and which was utilized in this research is the Davidon-Fletcher-Powell algorithm. The Davidon-Fletcher-Powell (D-F-P) was one of the earliest presented that approximates the inverse Hessian throughout the iterative procedure. The algorithm found in Luenberger [44] is as follows:

1. determine starting point $x_0$
2. set $S_0 = I$, $\beta = f(x_0)$, $k = 0$
3. set $d_k = -S_k \nabla f(x_k)$
4. minimize $f(x_k + \alpha d_k)$
   $\alpha > 0$
5. obtain $x_{k+1} = x_k + \alpha d_k$
6. determine $\gamma = f(x_{k+1})$
7. is $|\gamma - \beta| < \epsilon$ (yes $\Rightarrow$ stop optimal solution
   no $\Rightarrow$ continue
8. $P_k = \alpha_k d_k$
9. obtain $\nabla f(x_{k+1})$
10. $q_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
11. $S_{k+1} = S_k + \frac{P_k \cdot P_k'}{P_k'q_k} - \frac{S_k q_k \cdot q_k' S_k}{q_k' S_k q_k}$
12. return to 3

A flow chart for the above algorithm is depicted in Figure 39.
Figure 39. Davidon-Fletcher-Powell Flow Chart
Assuming that $S_k$ is the inverse Hessian, this method is equivalent to Newton's Method. It has been shown that the approximation given in the D-F-P does indeed approximate the inverse Hessian. The following derivation is given in Luenberger [44] and Himmelblau [46]:

from the gradient of $f(x)$

$$\nabla f(x_{k+1}) - \nabla f(x_k) = F(x_k)[(x_{k+1} - x_k)]$$

multiply both sides by $F^{-1}(x_k)$

$$F^{-1}(x_k)[\nabla f(x_{k+1}) - \nabla f(x_k)] = F^{-1}(x_k)F(x_k)(x_{k+1} - x_k)$$

$$x_{k+1} - x_k = F^{-1}(x_k)[\nabla f(x_{k+1}) - \nabla f(x_k)]$$

$F^{-1}(x_k)$ is to be approximated by $S_k$, thus:

$$F^{-1}(x_k) \approx S_{k+1} = S_k + \Delta S_k$$

allowing $q_k = \nabla f(x_{k+1}) - \nabla f(x_k)$

$$F^{-1}(x_k)q_k = x_{k+1} - x_k$$

allowing $p_k = x_{k+1} - x_k$, where from the D-F-P

$$p_k = \alpha d_k$$

$$x_{k+1} = \alpha d_k + x_k$$

$$p_k = \alpha d_k + x_k - x_k = \alpha d_k$$

then $F^{-1}(x_k)q_k = p_k$
substituting the approximation of $F^{-1}(x_k)$

$$S_{k+1}q_k \approx p_k$$

$$(S_k + \Delta S_k)q_k \approx p_k$$

$$S_kq_k + \Delta S_kq_k \approx p_k$$

$$\Delta S_kq_k \approx p_k - S_kq_k$$

by direct substitution the solution then is

$$S_k = \frac{p_kp_k'}{p_kq_k} - \frac{S_kq_kq_k'S_k}{q_k'S_kq_k}$$

remembering $S_{k+1} = S_k + \Delta S_k$

$$S_{k+1} - S_k = \Delta S_k$$

$$S_{k+1} = S_k \frac{p_kp_k'}{p_kq_k} - \frac{S_kq_kq_k'S_k}{q_k'S_kq_k}$$

simplifying notation

$$S_{k+1} = S_k + A - B$$
One may note that the second and third matrices are symmetric, thus if $S_k$ is symmetric so is $S_{k+1}$. This is proven in [44, 45, 46 and 48]. The A matrix ensures that $S_k$ is positive definite of all stages and in the limit cancels out $S'$. Because the matrix $S_k$ is positive definite at all steps the directions generated are always descent directions.

The D-F-P then has the following advantages over other algorithms [44]:

1. It requires only gradient information.

2. The directions are guaranteed to be directions of descent.

3. For a quadratic problem convergence is guaranteed to be n steps.

As indicated in item 3, if f is quadratic with positive definite Hessian, the D-F-P will converge in, at most, n steps. This is because the directions generated are conjugate directions which can be shown to converge in n steps for quadratic problems. (For further details see [44, 45, 46 and 48].) Chapter VI compares convergence of the D-F-P and Steepest Descent. Appendix C gives an example of D-F-P.

The next topic to consider with respect to the optimization methodology is that of the line search technique utilized in the D-F-P. Many line search techniques exist. Some are Fibonacci, Golden Section, Newton's Method,
Method of False Position, and Cubic and Quadratic Fit. The choice of line search technique did not appear to be a crucial decision in this research so Golden Section was used since it can be easily applied.

Golden Section [47] is a numerical approximation to the Fibonacci search. The Golden Section method does not use the exact Fibonacci numbers. Due to this, the numbers need not be computed or stored in the program. The algorithm found in Jacoby [47] is as follows:

1. Two values, \( b' \) and \( u' \), which bracket the minimum are assumed known;

determine:

\[
\lambda_1^1 = 0.382 (u^1 - b^1) + b^1 \\
\lambda_2^1 = 0.618 (u^1 - b^1) + b^1
\]

compute \( f_1(x + \lambda_1^1 d) \) and \( f_2(x + \lambda_2^1 d) \) GO TO 3

2. Find the following interior points:

\[
\lambda_1^k = 0.382 (u^k - b^k) + b^k \\
\lambda_2^k = 0.618 (u^k - b^k) + b^k
\]

compute:

\( f_1(x + \lambda_1^k d) \) or \( f_2(x + \lambda_2^k d) \)
3. If \( f_1(x + \lambda_1^k d) > f_2(x + \lambda_2^k d) \)

set \( b^{k+1} = \lambda_1^k \)

\[ \lambda_1^{k+1} = \lambda_2^k \]

\[ u^{k+1} = u^k \]

\( k = k + 1 \)

GO TO 4

else set \( u^{k+1} = \lambda_2^k \)

\[ \lambda_2^{k+1} = \lambda_1^k \]

\( b^{k+1} = b^k \)

\( k = k + 1 \)

GO TO 4

4. If \( \lambda_2^k - \lambda_1^k \) tolerance parameter = \( \varepsilon \) STOP

ELSE: return to 2 computing only appropriate \( \lambda_1^k \) or \( \lambda_2^k \)
Figure 40 exhibits the general methodology of the Golden Section line search. An implementation issue to consider is that of how to determine values bracketing the minimum and also maintaining feasibility. Feasibility for the shrinkage constraint is controlled via the penalty application. Feasibility for the remaining constraints' upper and lower bounds is controlled via the line search, as will be demonstrated. One approach to maintaining feasibility would be to constrain \( \alpha \) such that, given a current value of width, height, or position and its direction, the revised value would be feasible. The \( \alpha \) guaranteeing this would be determined as follows.

For the width of variable \( i \):

\[
W_i' = W_i + \alpha_{Wi} d_{Wi}
\]

where:

- \( W_i' \) = revised value for width \( i \)
- \( W_i \) = current value for width \( i \)
- \( \alpha_{Wi} \) = maximum line search parameters for \( w_i \) such that \( W_i' \) is feasible with respect to upper and lower bounds
- \( d_{Wi} \) = search direction for \( W_i \)

To ensure that \( W_i \) is feasible with respect to both its upper and lower bounds:
Figure 40. Golden Section
if $d_{Wi} < 0 \alpha_{Wi} = \frac{W_i}{d_{Wi}}$

if $d_{Wi} > 0 \alpha_{Wi} = \frac{\text{Upper Bound} - W_i}{d_{Wi}}$

To obtain an overall $\alpha$ ensuring feasibility of each and every continuous variable with respect to their upper and lower bounds, choose:

$$\alpha_{\max} = \min_{\forall n} (\alpha_{Wi}, \alpha_{Hi}, \alpha_{Pi})$$

Thus, the Golden Section line search bracket for this implementation strategy would be $[0, \alpha_{\max}]$. The drawback to this approach would be that choosing the minimum $\alpha$ would possibly disallow other variables to be increased within their bounds, thus improving the objective function.

Consider the following example:

\[
\begin{align*}
\min f(x) &= -3W - 2H - 3W^2 - 2H^2 \\
\text{subject to} &\quad W \leq 2 \\
&\quad H < 3 \\
\text{Current value for width and height:} &\quad W^1 = 1 \\
&\quad H^1 = 1
\end{align*}
\]
Consider the search directions:

\[ d_W = 2 \]
\[ d_H = 2 \]

Determine \( \alpha_{\text{max}} \)

\[ \alpha_W = \frac{2-1}{2} = \frac{1}{2} \]
\[ \alpha_H = \frac{3-1}{2} = 1 \]
\[ \alpha_{\text{max}} = \min \left( \frac{1}{2}, 1 \right) = \frac{1}{2} \]

The optimal solution for the line search given \( 0 \leq \alpha \leq \frac{1}{2} \) would be:

\[ W^2 = W^1 + \frac{1}{2} d_W = 1 + \frac{1}{2} (2) = 2 \]
\[ H^2 = H^1 + \frac{1}{2} d_H = 1 + \frac{1}{2} (2) = 2 \]
\[ f(x^2) = -30 \]

However, if the \( \alpha_{\text{max}} \) was chosen as the

\[ \max \left( \frac{1}{2}, 1 \right) = 1 \]

and the variables reset to their bounds, the optimal solution given \( 0 \leq \alpha \leq 1 \) would be

\[ W^2 = 2 \]
\[ H^2 = 3 \]
\[ f(x^2) = -42 \]

As shown in the example, the restriction of \( \alpha \) such that no variable could become infeasible did not lead to an
optimal line search. However, if one determines the $\alpha_{\text{max}}$
where:

$$\alpha_{\text{max}} = \max (\alpha_{W_i}, \alpha_{H_i}, \alpha_{P_i})$$

and resets the variables when they become infeasible, an improved line search technique is implemented. Thus, infeasibility is handled explicitly by resetting any variables that become infeasible.

One advantage to this method is that an extended area is considered for the line search thus allowing a greater probability of determining the optimum. Another advantage is that it will never search in an area where all variables are infeasible.

When implemented, a variable feasibility check is applied after each $\lambda_k^1$ and $\lambda_k^2$ is calculated. Any variables less than zero are set to zero and any variables exceeding their bounds are set to their bounds.

A drawback to this approach is that resetting variables can change the unconstrained convex objective function into one which is non-convex. This is shown in Figure 40. This seems to occur late in the optimization when the current D-F-P solution is over-feasible (i.e. the solution is not on the shrinkage boundary). The determination of the $\alpha_{\text{max}}$ usually yields very small values since by this point in the optimization process the search directions are of a large
magnitude relative to the variable values. For this reason double precision calculations are performed when $\alpha_{\text{max}}$ is of very small magnitude. The efficiency of the Golden Section search is discussed in Chapter VI.

From the previous discussion of Benders' Decomposition it was emphasized that the subproblem optimization must be dual adequate. In this case the D-F-P must be dual adequate. Assuming that the D-F-P arrives at a true optimum, this is no problem since the dual variables can be solved directly from the Kuhn-Tucker conditions. However, the D-F-P usually optimizes to a good value, within two to five percent of the true optimum. This accuracy is a function of the tolerance parameters in both the D-F-P and Golden Section stopping rules. Because of this discrepancy the Kuhn-Tucker conditions may not be satisfied. Thus, the variables must be altered slightly so as to satisfy the Kuhn-Tucker conditions. Figure 41 presents a flow chart of a methodology designed to accomplish this.

The underlying concept is to vary the variables slightly so as to satisfy the Kuhn-Tucker conditions. Given a solution obtained by D-F-P does not satisfy the Kuhn-Tucker conditions, it may be necessary to adjust a few variables away from boundary values and also to place a few variables currently close to but not at their bounds to their bounds.
One may recall the previously suggested algorithm that solved the Kuhn-Tucker conditions by total enumeration. This algorithm guarantees the optimal solution, if it exists, by a complete enumeration of the boundary possibilities. The idea proposed here is to use the D-F-P search algorithm to obtain a solution close to the optimum so that only a few iterations of a heuristic algorithm would be necessary to arrive at the true optimal solution.

As can be seen from Table 2, the dual variable representing the shrinkage constraint is in each and every Kuhn-Tucker condition. Furthermore, one can see that variables not at their bounds have only two terms in their respective Kuhn-Tucker conditions: (1) the objective function gradient and (2) the shrinkage function gradient multiplied by the dual variable representing the shrinkage function. Because a linear approximation to the shrinkage function was utilized, a linear system of equations must be satisfied. This linear system is defined as:

\[ C = \left\{ \begin{array}{l}
\text{Index Of } W_i', H_i', \quad W_i \neq 0 \text{ and } W_i \neq \text{WDDBND}_i \\
\text{and } P_i \text{ By Individual } H_i \neq 0 \text{ and } H_i \neq \text{HGTBND}_i \\
\text{Variables } P_i \neq 0 \text{ and } W_i + P_i \neq \text{WDPSBND}_i 
\end{array} \right. \]
\[ \text{CWID}_i + 2(\text{CWID2}_i)W_i + \frac{\partial g}{\partial W_i} \text{DUALSHRNK} = 0 \]

\[ \text{CHGT}_i + 2(\text{CHGT2}_i)H_i + \frac{\partial g}{\partial H_i} \text{DUALSHRNK} = 0 \]

\[ \text{CPOS}_i + 2(\text{CPOS2}_i)P_i + \frac{\partial g}{\partial P_i} \text{DUALSHRNK} = 0 \]

\[
\sum_{i \in C} \frac{\partial g}{\partial W_i} W_i + \sum_{i \in C} \frac{\partial g}{\partial H_i} H_i + \sum_{i \in C} \frac{\partial g}{\partial P_i} P_i = \text{SHRINK DIFF}
\]

where: \( \text{SHRINKDIFF} = \text{SHRINKTOL} - \text{CURRENT SHRINKAGE VALUE} \)

In simpler notation, allowing \( \frac{\partial f}{\partial W_i}, \frac{\partial f}{\partial H_i}, \frac{\partial f}{\partial P_i} \) to represent the gradient of the objective function, the system of simultaneous equations then is:

\[
\frac{\partial f}{\partial W_i} + \frac{\partial g}{\partial W_i} \text{DUALSHRNK} = 0 \ \forall \ i \in C
\]

\[
\frac{\partial f}{\partial H_i} + \frac{\partial g}{\partial H_i} \text{DUALSHRNK} = 0 \ \forall \ i \in C
\]

\[
\frac{\partial f}{\partial P_i} + \frac{\partial g}{\partial P_i} \text{DUALSHRNK} = 0 \ \forall \ i \in C
\]

\[
\sum_{i \in C} \frac{\partial g}{\partial W_i} W_i + \sum_{i \in C} \frac{\partial g}{\partial H_i} H_i + \sum_{i \in C} \frac{\partial g}{\partial P_i} P_i = \text{SHRINK DIFF}
\]

where: \( \text{DUALSHRNK} \) represents the value for the dual variable associated with the shrinkage constraint
The problem is that the D-F-P yields a solution which is overfeasible. That is, the variable values for width, height, and position are such that the shrinkage constraint is not satisfied by equality. Thus, the dual variable associated with the shrinkage constraint must be zero.

With the dual variable associated with the shrinkage constraint valued at zero, the Kuhn-Tucker conditions for variables not at their bounds cannot be satisfied. Because of this the dual variable representing the shrinkage constraint must have value, and in turn the shrinkage constraint must be satisfied by equality. To accomplish this equation (5-13) is added to the system of equations. Equation 5-13 should force the solution to a boundary of the shrinkage constraint. The shrinkage difference, right side of equation (5-13), is the distance the current solution needs to move to reach the shrinkage boundary (i.e. current shrinkage value-shrinkage tolerance).

The algorithm for satisfying the Kuhn-Tucker conditions is to utilize the D-F-P as previously discussed, set up the system of simultaneous equations for those variables not at their bounds, invert the matrix and solve for the values which satisfy the Kuhn-Tucker conditions. Although this approach seems simple, there are two pitfalls to its total implementation. First, a variable could be assigned to its bound from the D-F-P, but at the true optimal it is not.
Hence, it would not be included in the equation C.

Secondly, a variable in the D-F-P solution might not be at a bound even though at the optimal it should be. Both of these pitfalls can be handled in the following way.

First, one may consider a variable which currently is not at a bound but should be. This variable will be included in the system C, but it should not be. When the system of equations is solved it will yield a variable to be infeasible (i.e. greater than its bound). This indicates that the variable should have been at its bound. The algorithm then sets the variable to its bound, resets the remaining variables to their original values, reformulates the system by altering the set C and then resolves the system of equations. This iteration procedure is continued until all variables are feasible and the Kuhn-Tucker conditions are satisfied.

The second implementation pitfall was that a variable in the D-F-P solution might not be at a bound even though it is at the optimal solution it should be. These variables are identified by the remaining Kuhn-Tucker conditions. Once the system of simultaneous equations has been set up and a feasible solution found, the remaining Kuhn-Tucker conditions are evaluated. If any of the Kuhn-Tucker conditions are now not satisfied, the variable corresponding
to the violated condition is assumed not at its bound. This is proper as the determination of set C and the subsequent solution of its system of simultaneous equations guaranteed that all variables not at their bounds satisfied the Kuhn-Tucker conditions. The variable not satisfying a condition is entered into set C as it now should be altered from its bound. From the revised set C, the algorithm sets up the simultaneous equations and resolves them.

While there is no guarantee that this heuristic algorithm will arrive at the optimal since there is no proof that the solution would not bounce back and forth from one boundary decision to another, this did not seem to be a problem from the practical experience gained thus far. A flow chart of the algorithm is given in Figure 41.

Review of Chapter V

An overall picture of the procedure which was utilized has been shown in the flow chart in Figure 32 (page 131). As depicted, the first step is to enter the casting geometry.

The next step is to determine a set or risers and/or chills from which the program can then select combinations. This set is selected based upon the foundryman's knowledge and experience, but should be slightly over-specified (e.g. it should allow three risers to be utilized if only two are felt to be necessary). The foundryman must also input a
Figure 41. Kuhn-Tucker Algorithm
number of costs associated with the risers and chills proposed.

Once these data have been entered by the user, Benders' Decomposition is utilized to decompose the problem into a master problem which solves for integer variables representing the existence of chills and/or risers. The master problem indicates a subproblem which solves for riser and chill size and location. The interrelationship between the master problem and the subproblem is captured through the addition of violated constraints in the master problem. The subproblem is solved to determine which is the most violated constraint. Through an enumeration procedure the optimal solution to each master problem is determined.

For each combination of risers and/or chills selected the subproblem selects the width, height, and position of each riser or chill. This subproblem is a constrained nonlinear optimization problem. It was formulated as a simple bounded variable problem by utilizing a penalty on the more complex shrinkage constraint. This bounded variable minimization problem was solved by a modification of the D-F-P method. The line search required by the D-F-P algorithm was a Golden Search technique. Results of these algorithms are given in Chapter VI.

Finally, once the solution to the D-F-P has been determined, the dual variables must be evaluated so as to
pursue a new combination of risers and/or chills. An heuristic technique was proposed for solving the Kuhn-Tucker conditions by varying slightly the output from the D-F-P. The proposed algorithms appear to work well and are evaluated in Chapter VI.
Chapter VI

RESULTS

This chapter represents a culmination of one year of research, including nearly five-hundred computer runs. Regretably, it was discovered that the solidification simulation which was discussed in Chapter IV required an unacceptable amount of CPU time to obtain an adequate simulation. To obtain the solidification gradient two-hundred minutes of CPU time may be required, or perhaps more if second-order approximations are utilized. Actually, this is only an initial gradient calculation. What is required in reality is a shrinkage gradient update for each iteration. The shrinkage gradient should also be calculated for each subproblem. Chapter IV presented a brief example of a problem which updated the shrinkage gradient as required by the optimization routine. The solution moved toward an optimal solution and, according to the simulation the shrinkage cavity had been reduced by fifty percent after forty-five minutes of CPU time. To allow the optimization to proceed even this far the melting point had to be decreased to 2,760 degrees Fahrenheit. This only required a forty
degree drop until solidification occurred and, most importantly, did not allow for the inclusion of latent heat of fusion.

The long run times do not seem out of the ordinary according to Pehlke, et al. Pehlke maintains that a foundry should be prepared to spend between one thousand to five thousand dollars for simulation of one casting. A run of this magnitude on the OSU computer system could quite literally take weeks to perform.

Due to this excessive computing time a number of runs were made to obtain a number of examples of shrinkage gradients. In these examples the gradient for width and height of a riser and/or chill had a range of anywhere from .10 to .24 cubic inches per inch. Position, on the other hand, had exhibited little effect on shrinkage and tended toward zero. This could be expected as the castings simulated were of such geometric shape that position was relatively unimportant to the shrinkage. To test the optimization approach the shrinkage gradient was approximated by subjective values for a given casting. One should realize that these subjective values are based on a number of actual simulations. The hypothesis is that if the model can solve for an optimal solution for a wide range of shrinkage gradients, it should also solve for the optimal solution given the actual shrinkage gradient. Consistent
with this, two shrinkage gradients were utilized—one with a large variation in the individual variable gradients, and another with a rather small variation. It was felt that most gradients would fit somewhere in between.

The remainder of this chapter will be a discussion of the following:

1. General Discussion Of The Research
2. Solution Results
3. Efficiency Of Benders' Algorithm
4. Sensitivity Analysis Of The Subproblem
5. Comparison Of Different Initial Penalty Values
6. Comparison Of Steepest Descent And Davidon-Fletcher-Powell
7. Dual Adequacy Of The Subproblem
8. Convexity Problems

General Discussion

Eighteen different problems were solved using the proposed algorithm. In each case the Benders' optimal tolerance parameter ε was equal to zero. This also represented the solution of nearly two-hundred subproblems by use of the Davidon-Fletcher-Powell algorithm using a penalty function for the shrinkage constraint. In all, this represents 1,754 iterations of the Davidon-Fletcher-Powell algorithm and 18,344 iterations of the Golden Section. Thus, a fairly lengthy trial period was executed.
Appendix A gives an example problem utilizing Benders' algorithm and Appendix B contains a problem utilizing the Davidon-Fletcher-Powell algorithm. Appendix C gives an example problem utilizing the Golden Section line search.

The average CPU utilized for a problem using four zero-one variables was 16.45 seconds with a standard deviation of 0.44. For a problem with six zero-one variables the average CPU time was 21.66 seconds with a standard deviation of 2.4 seconds. All of the computing was done on an Amdahl V8 computer. Thus, for an even larger casting problem CPU time should not be a problem with respect to the optimization algorithm, as even a large casting problem would probably only use ten to twenty riser-chills, at most.

When the algorithm starts a number of informational outputs are available. A few of them are given in Figure 42. The algorithm begins with an initial solution consisting of all of the zero-one variables set to the value of one. If there is not a feasible solution to the Benders subproblem with all risers and chills in the solution, there can be no feasible solution. As an initial starting point for the subproblem the individual widths, heights, and positions are set to their respective average values (selection of initial values has been discussed in Chapter V).

After the initialization phase the shrinkage gradient is determined and then applied to determine the overall
Figure 42. General Informational Outputs

GRAPHIC DISPLAY OF CASTING GEOMETRY

CASTING GEOMETRY CODE

0 - INDICATES EXTERNAL AIR BARRIER
1 - INDICATES SAND MATERIAL
2 - INDICATES CHILL MATERIAL
3 - INDICATES RISER MATERIAL
4 - INDICATES CASTING AREA IN CONTACT WITH SAND
5 - INDICATES SAND AREA IN CONTACT WITH CASTING

OBJECTIVE FUNCTION COEFFICIENT INFORMATION:

<table>
<thead>
<tr>
<th>TABLE</th>
<th>TYPE</th>
<th>LENGTH (in)</th>
<th>WIDTH (in)</th>
<th>HEIGHT (in)</th>
<th>WIDTH HEIGHT POSITION</th>
<th>WIDTH HEIGHT POSITION</th>
<th>WIDTH HEIGHT POSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RISE</td>
<td>1000.00</td>
<td>100.00</td>
<td>100.00</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>RISER</td>
<td>1200.00</td>
<td>300.00</td>
<td>250.00</td>
<td>60.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>CHILL</td>
<td>1300.00</td>
<td>320.00</td>
<td>500.00</td>
<td>50.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>CHILL</td>
<td>1400.00</td>
<td>410.00</td>
<td>310.00</td>
<td>100.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
### ZERO - ONE VARIABLE 1

<table>
<thead>
<tr>
<th>Corner Extreme Positions</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Left Corner</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Upper Right Corner</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>Lower Left Corner</td>
<td>21</td>
<td>12</td>
</tr>
<tr>
<td>Lower Right Corner</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>

Bound on Height or Width Depending on Configuration = 4
Side of Zero - One Variable Contacting Side of Rectangle - 1
Rectangle Number Which Zero - One Variable Contacts - 4

### ZERO - ONE VARIABLE 2

<table>
<thead>
<tr>
<th>Corner Extreme Positions</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Left Corner</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>Upper Right Corner</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>Lower Left Corner</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Lower Right Corner</td>
<td>7</td>
<td>16</td>
</tr>
</tbody>
</table>

Bound on Height or Width Depending on Configuration = 4
Side of Zero - One Variable Contacting Side of Rectangle - 4
Rectangle Number Which Zero - One Variable Contacts - 2
Side of Rectangle Which Zero - One Variable Contacts - 1

### ZERO - ONE VARIABLE 3

<table>
<thead>
<tr>
<th>Corner Extreme Positions</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Left Corner</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>Upper Right Corner</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>Lower Left Corner</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>Lower Right Corner</td>
<td>13</td>
<td>7</td>
</tr>
</tbody>
</table>

Bound on Height or Width Depending on Configuration = 3
Side of Zero - One Variable Contacting Side of Rectangle - 1
Rectangle Number Which Zero - One Variable Contacts - 1
Side of Rectangle Which Zero - One Variable Contacts - 4

### ZERO - ONE VARIABLE 4

<table>
<thead>
<tr>
<th>Corner Extreme Positions</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Left Corner</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>Upper Right Corner</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>Lower Left Corner</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>Lower Right Corner</td>
<td>14</td>
<td>11</td>
</tr>
</tbody>
</table>

Bound on Height or Width Depending on Configuration = 3
Side of Zero - One Variable Contacting Side of Rectangle - 1
Rectangle Number Which Zero - One Variable Contacts - 4
Side of Rectangle Which Zero - One Variable Contacts - 4

---

**Figure 42 (continued)**
objective function gradient. A Golden Section line search is performed. In Figure 43 note that an optimality check is performed for the Davidon-Fletcher-Powell algorithm. This check is performed by comparing the difference between the previous D-F-P solution and the current D-F-P solution to a predetermined stopping parameter. Once optimality is achieved via the D-F-P the exact primal and dual solution is found via the algorithm discussed in Chapter IV.

The key dual variables with respect to the choosing of successive combinations of the zero-one variables are those associated with the existence constraints. In Figure 44 a number of Benders constraints are shown. Under the constraints are listed the actual constraint value for each combination of risers and chills presented for this iteration. Also shown is the current maximum value for all calculated constraints for each combination of risers and chills.

The next step is then to choose that combination having the overall current minimum value and to solve its respective subproblem. Figure 44 exhibits the optimal solution. Notice the difference of zero circled on the Figure for the Benders tolerance parameter.

Optimization Algorithm Efficiency Discussion

As stated in Chapter IV, the solidification simulation CPU time is unacceptable. Because of the shrinkage gradient
Figure 43. Golden Section Sample Output

Table: Golden Section Results - Davidon-Fletcher-Powel

<table>
<thead>
<tr>
<th>Variable</th>
<th>Original Value</th>
<th>New Value</th>
<th>Lambda</th>
<th>Direction Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.1</td>
<td>0.122</td>
<td>-0.172</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>1.1</td>
<td>0.122</td>
<td>-0.172</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>1.1</td>
<td>0.122</td>
<td>-0.172</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
<td>1.1</td>
<td>0.122</td>
<td>-0.172</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>1.1</td>
<td>0.122</td>
<td>-0.172</td>
</tr>
<tr>
<td>6</td>
<td>1.0</td>
<td>1.1</td>
<td>0.122</td>
<td>-0.172</td>
</tr>
</tbody>
</table>

D-F-P Optimality Check
CURRENT ITERATION -  17

Subproblem Solution

<table>
<thead>
<tr>
<th>Iteration</th>
<th>CHILL</th>
<th>RISER</th>
<th>KISER</th>
<th>CHILL</th>
<th>CHILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.142689E+05</td>
<td>0.100E+04</td>
<td>0.120E+04</td>
<td>0.140E+04</td>
<td>0.119E+04</td>
</tr>
<tr>
<td>2</td>
<td>0.09490E+05</td>
<td>0.200E+04</td>
<td>0.180E+04</td>
<td>0.264E+04</td>
<td>0.108E+04</td>
</tr>
<tr>
<td>3</td>
<td>0.65034E+05</td>
<td>0.100E+04</td>
<td>0.160E+04</td>
<td>0.202E+04</td>
<td>0.106E+04</td>
</tr>
<tr>
<td>4</td>
<td>0.160E+06</td>
<td>0.265E+04</td>
<td>0.120E+04</td>
<td>0.289E+04</td>
<td>0.277E+04</td>
</tr>
<tr>
<td>5</td>
<td>0.23145E+06</td>
<td>0.100E+04</td>
<td>0.357E+07</td>
<td>0.583E+07</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>6</td>
<td>0.13695E+06</td>
<td>0.244E+04</td>
<td>0.126E+04</td>
<td>0.263E+08</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>7</td>
<td>0.13926E+06</td>
<td>0.100E+04</td>
<td>0.120E+04</td>
<td>0.271E+08</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>8</td>
<td>0.12010E+06</td>
<td>0.144E+04</td>
<td>0.126E+08</td>
<td>0.140E+04</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>9</td>
<td>0.75356E+06</td>
<td>0.133E+08</td>
<td>0.126E+04</td>
<td>0.144E+08</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>10</td>
<td>0.75952E+06</td>
<td>0.164E+04</td>
<td>0.120E+04</td>
<td>0.151E+08</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>11</td>
<td>0.31069E+06</td>
<td>0.112E+08</td>
<td>0.935E+07</td>
<td>0.114E+08</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>12</td>
<td>0.30342E+06</td>
<td>0.140E+08</td>
<td>0.104E+08</td>
<td>0.140E+08</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>13</td>
<td>0.33674E+05</td>
<td>0.100E+04</td>
<td>0.120E+04</td>
<td>0.130E+08</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>14</td>
<td>0.23803E+03</td>
<td>0.115E+07</td>
<td>0.149E+07</td>
<td>0.140E+04</td>
<td>0.100E+04</td>
</tr>
<tr>
<td>15</td>
<td>0.21930E+05</td>
<td>0.100E+04</td>
<td>0.170E+07</td>
<td>0.114E+08</td>
<td>0.372E+07</td>
</tr>
<tr>
<td>16</td>
<td>0.1483E+06</td>
<td>0.126E+08</td>
<td>0.126E+08</td>
<td>0.140E+08</td>
<td>0.140E+08</td>
</tr>
<tr>
<td>17</td>
<td>0.14824E+05</td>
<td>0.100E+04</td>
<td>0.112E+04</td>
<td>0.140E+04</td>
<td>0.140E+04</td>
</tr>
</tbody>
</table>

All Benders Constraints Generated Thus Far

Actual Value of each (0-1) combination for most recent constraint

<table>
<thead>
<tr>
<th>Number</th>
<th>CHILL</th>
<th>RISER</th>
<th>RISER</th>
<th>CHILL</th>
<th>CHILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>39</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Benders Master Problem Results

Figure 44. Benders Decomposition Sample Output

Benders Constraint

All Benders Constraints Generated Thus Far

Actual Value of each (0-1) combination for most recent constraint

Minimum constraint value for each combination
**Figure 44 (continued)**

The optimum solution to the master problem is:

- **Minimum Value of Enumeration**: 0.1708535e+05
- **Combination Number**: 26

---

**Results of Master Problem Optimization Check**

- **Tolerance Parameter**: 0.000001
- **Current Upper Bound**: 0.17084e+05
- **Current Value**: 0.1708535e+05
- **Difference**: 0.0

---

Optimality (Tolerance = 0)
estimation it was important to test the optimization algorithm thoroughly. The hypothesis then is if a wide range of shrinkage gradient values were utilized and a solution found efficiently, actual shrinkage gradients within the range will also efficiently yield solutions.

Since it is in solving the continuous subproblem that extensive use of the solidification simulation would occur, and such simulations are by far the most costly operation in the optimization process, then to the extent that subproblems are avoided in a Benders implementation is indicative of the efficiency of the approach. Specifically, one may define this efficiency measured as follows:

\[
\text{Efficiency} = \left(1 - \frac{\text{number of continuous subproblems solved}}{\text{total number of feasible continuous subproblems}}\right) \times 100\%
\]

There are many factors one could consider which effect efficiency. Some of them are:

1. initial penalty value
2. penalty function type
3. penalty update factor
4. stopping parameter size in D-F-P
5. stopping parameter size in Golden Section
6. number of risers-chills
7. ratio of risers to chills
8. if the solidification package were implemented:
   a. size of nodes
   b. time step increment
   c. alloy type, etc.
9. spread or range of the zero-one variable cost coefficients
10. spread or range of the shrinkage gradient values
11. spread or range of the continuous variable cost coefficients

To consider all of these in a properly designed statistical experiment would be too time consuming, so only four of these factors were considered. A $2^4$ complete factorial experiment with one replication was then conducted.

The factors considered were:
1. number of risers and chills that form the possible combinations
2. cost coefficients for the continuous variables
3. cost coefficients for the discrete zero-one variables
4. shrinkage gradient values

These factors seemed reasonable to test since the constraint generated to determine successive subproblems by the Benders master problem is:

$$y^* \leq f^j(X) + \sum_{i=1}^{n} Z_i [CZ_i - M(DEXIS_i)]$$

where: $DEXIS_i$ = dual variable associated with existence constraints.

As can be seen there are two factors in this constraint. The first is the dual variable associated with the existence...
constraints discussed in Chapter V. The second is the value of the zero-one variable cost coefficients. If a zero-one variable is included in the current subproblem the dual variable associated with its existence constraint would be zero. However, if a zero-one variable is not included in the current subproblem the dual variable associated with its constraint could accept some value. The dual variable associated with the existence constraint is determined by:

\[
D_{\text{EXIS}}_i = \min \left\{ \min [0, \frac{\partial f}{\partial W_i} + \frac{\partial g}{\partial W_i} D_{\text{DUALSHK}}] + \min [0, \frac{\partial f}{\partial H_i} + \frac{\partial g}{\partial H_i} D_{\text{DUALSHK}}] + \min [0, \frac{\partial f}{\partial P_i} + \frac{\partial g}{\partial P_i} D_{\text{DUALSHK}}] \right\}
\]

where:

\(D_{\text{DUALSHK}}\) = dual variable associated with the shrinkage constraint

\(D_{\text{EXIS}}_i\) = dual variable associated with existence constructed for zero-one variable \(i\)

\(f\) = objective function for continuous variables

\(g\) = shrinkage optimization

As can be seen, the quadratic cost coefficients along with the shrinkage gradient values affect the determination of the
dual variable associated with the existence constraints. This was the motivation for including the quadratic cost coefficients as factors.

For the first factor, number of risers and chills, the high effect was six zero-one variables while the low effect was four zero-one variables. The second factor's quadratic cost coefficients of the continuous variable's effects were measured by the spread of the quadratic cost coefficients.

Similarly, for the third factor cost coefficients for the zero-one variables the spread of their values was also utilized as the effect. The last factor, shrinkage gradient values, also utilized the spread in the shrinkage values as a measure of its effect. The spread of each of these effects was measured by the standard deviation of their respective values. It was felt that the standard deviation would best capture the spread of the values even though the values were obtained subjectively. Table 3 gives all data utilized in the experiment along with the averages and standard deviations of the data.

The high-low values for each factor are summarized in the following list:
Table 3. Data Utilized In Efficiency Experimental Design

<table>
<thead>
<tr>
<th>CASTING #1</th>
<th>CASTING #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISERS</td>
<td>RISERS</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CHILL</td>
<td>CHILL</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>CHILL</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>CHILL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RISERS</th>
<th>CASTING #1</th>
<th>CASTING #2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SHRINKAGE GRADIENT</th>
<th>High</th>
<th>Low</th>
<th>SHRINKAGE GRADIENT</th>
<th>High</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>W₁</td>
<td>-.20</td>
<td>-.16</td>
<td>W₁</td>
<td>-.2</td>
<td>-.14</td>
</tr>
<tr>
<td>H₁</td>
<td>-.10</td>
<td>-.15</td>
<td>H₁</td>
<td>-.12</td>
<td>-.12</td>
</tr>
<tr>
<td>P₁</td>
<td>-.002</td>
<td>-.002</td>
<td>P₁</td>
<td>-.003</td>
<td>-.003</td>
</tr>
<tr>
<td>W₂</td>
<td>-.20</td>
<td>-.13</td>
<td>W₂</td>
<td>-.23</td>
<td>-.13</td>
</tr>
<tr>
<td>H₂</td>
<td>-.15</td>
<td>-.12</td>
<td>H₂</td>
<td>-.16</td>
<td>-.12</td>
</tr>
<tr>
<td>P₂</td>
<td>-.003</td>
<td>-.004</td>
<td>P₂</td>
<td>-.003</td>
<td>-.002</td>
</tr>
<tr>
<td>W₃</td>
<td>-.2</td>
<td>-.15</td>
<td>W₃</td>
<td>-.22</td>
<td>-.14</td>
</tr>
<tr>
<td>H₃</td>
<td>-.22</td>
<td>-.13</td>
<td>H₃</td>
<td>-.14</td>
<td>-.13</td>
</tr>
<tr>
<td>P₃</td>
<td>-.003</td>
<td>-.003</td>
<td>P₃</td>
<td>-.002</td>
<td>-.002</td>
</tr>
<tr>
<td>W₄</td>
<td>-.15</td>
<td>-.14</td>
<td>W₄</td>
<td>-.15</td>
<td>-.15</td>
</tr>
<tr>
<td>H₄</td>
<td>-.21</td>
<td>-.13</td>
<td>H₄</td>
<td>-.21</td>
<td>-.12</td>
</tr>
<tr>
<td>P₄</td>
<td>-.002</td>
<td>-.001</td>
<td>P₄</td>
<td>-.002</td>
<td>-.002</td>
</tr>
<tr>
<td>W₅</td>
<td>-.18</td>
<td>-.14</td>
<td>W₅</td>
<td>-.18</td>
<td>-.13</td>
</tr>
<tr>
<td>H₅</td>
<td>-.20</td>
<td>-.12</td>
<td>H₅</td>
<td>-.12</td>
<td>-.12</td>
</tr>
<tr>
<td>P₅</td>
<td>-.004</td>
<td>-.005</td>
<td>P₅</td>
<td>-.004</td>
<td>-.004</td>
</tr>
<tr>
<td>W₆</td>
<td>-.22</td>
<td>-.19</td>
<td>W₆</td>
<td>-.22</td>
<td>-.19</td>
</tr>
<tr>
<td>H₆</td>
<td>-.1</td>
<td>-.13</td>
<td>H₆</td>
<td>-.1</td>
<td>-.13</td>
</tr>
<tr>
<td>P₆</td>
<td>-.01</td>
<td>-.112</td>
<td>P₆</td>
<td>-.01</td>
<td>-.112</td>
</tr>
<tr>
<td>Average =</td>
<td>-.18</td>
<td>-.14</td>
<td>σₓ = - .042</td>
<td>.042</td>
<td>.012</td>
</tr>
</tbody>
</table>
Table 3. (continued)

<table>
<thead>
<tr>
<th>CASTING #1</th>
<th>CASTING #2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RISERS</strong></td>
<td><strong>RISERS</strong></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>CHILL 5</td>
<td>CHILL 4</td>
</tr>
<tr>
<td>CHILL 6</td>
<td>CHILL 3</td>
</tr>
</tbody>
</table>

**QUADRATIC COSTS**

<table>
<thead>
<tr>
<th></th>
<th>High</th>
<th>Low</th>
<th></th>
<th>High</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>W₁</td>
<td>100</td>
<td>100</td>
<td>W₁</td>
<td>4000</td>
<td>100</td>
</tr>
<tr>
<td>H₁</td>
<td>120</td>
<td>120</td>
<td>H₁</td>
<td>3900</td>
<td>120</td>
</tr>
<tr>
<td>P₁</td>
<td>100</td>
<td>100</td>
<td>P₁</td>
<td>3000</td>
<td>100</td>
</tr>
<tr>
<td>W₂</td>
<td>2000</td>
<td>200</td>
<td>W₂</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>H₂</td>
<td>5200</td>
<td>220</td>
<td>H₂</td>
<td>150</td>
<td>250</td>
</tr>
<tr>
<td>P₂</td>
<td>500</td>
<td>50</td>
<td>P₂</td>
<td>120</td>
<td>80</td>
</tr>
<tr>
<td>W₃</td>
<td>550</td>
<td>550</td>
<td>W₃</td>
<td>550</td>
<td>320</td>
</tr>
<tr>
<td>H₃</td>
<td>320</td>
<td>320</td>
<td>H₃</td>
<td>320</td>
<td>400</td>
</tr>
<tr>
<td>P₃</td>
<td>300</td>
<td>300</td>
<td>P₃</td>
<td>300</td>
<td>50</td>
</tr>
<tr>
<td>W₄</td>
<td>150</td>
<td>150</td>
<td>W₄</td>
<td>2500</td>
<td>410</td>
</tr>
<tr>
<td>H₄</td>
<td>250</td>
<td>250</td>
<td>H₄</td>
<td>1500</td>
<td>310</td>
</tr>
<tr>
<td>P₄</td>
<td>60</td>
<td>60</td>
<td>P₄</td>
<td>1200</td>
<td>100</td>
</tr>
<tr>
<td>W₅</td>
<td>2500</td>
<td>250</td>
<td>Average = 1478</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>H₅</td>
<td>1500</td>
<td>150</td>
<td>σₓ = 1492</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>P₅</td>
<td>1200</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W₆</td>
<td>4000</td>
<td>400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H₆</td>
<td>3000</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P₆</td>
<td>2000</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average = 1325</td>
<td>213</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>σₓ = 1519</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3. (continued)

### CASTING #1

<table>
<thead>
<tr>
<th>Risers</th>
<th>High</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1300</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>1200</td>
</tr>
<tr>
<td>1</td>
<td>3500</td>
<td>1400</td>
</tr>
<tr>
<td>5</td>
<td>400</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>1600</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>800</td>
</tr>
</tbody>
</table>

Average = 1158

\[
\sigma_x = 1313
\]

### CASTING #2

<table>
<thead>
<tr>
<th>Risers</th>
<th>High</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2200</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>200</td>
<td>1200</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>1300</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>800</td>
</tr>
</tbody>
</table>

Average = 1325

\[
\sigma_x = 1252
\]
Figure 45 shows the efficiency results from the experiment design.

A $2^4$ complete factorial design was conducted with only a single replicate. It is common that when the number of factors to be considered is large only a single replicate can be conducted [49]. With only a single replicate, however, it is impossible to compute an estimate of the experimental error. The common practice is then to assume a number of the interactions negligible and then combine their mean squares to estimate the experimental error. This procedure is subject to criticism on experimental grounds, but has acceptance in the field. One possible error would be to assume an interaction term negligible when it is not. This would yield an overestimated sample error term and in turn underestimate all other factors and interactions. As a result of this, some factors that are significant could become statistically insignificant. As suggested by Montgomery [49] all two-factor interactions were assumed significant. If most of the two-factor terms are small it seems likely that higher order
Figure 45. Efficiency Results For Optimization Algorithm
terms also would be small. The results are shown in Figure 46. Statistical Analysis System (SAS) was used to perform the analysis [50].

From Figure 46 it is apparent given the data that the following are highly significant:

1. number of risers-chills chosen
2. standard deviation of the zero-one variable coefficient
3. the interaction between the number of risers-chills chosen and the standard deviation of the zero-one variable coefficient

The following are fairly significant:

1. the standard deviation of the quadratic terms
2. the interaction between the number of risers-chills and the standard deviation of the quadratic terms

The following conclusions can be drawn given the data and statistical analysis. The first is that the number of risers-chills chosen have an impact on the efficiency of the algorithm (as the number increases so does the efficiency). The results indicate that the spread, standard deviation of the zero-one variable cost coefficients is significant. This is logical as the value of the zero-one cost coefficient has a considerable impact upon the selection of the successive subproblem chosen in a Benders implementation.
The following is a condensed version of the output of a PROC ANOVA routine performed by the statistical package SAS.

**ANALYSIS OF VARIANCE PROCEDURE**

**CLASS LEVEL INFORMATION**

<table>
<thead>
<tr>
<th>CLASS LEVELS</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero-One Cost Coefficient Spread</td>
<td>218 1282</td>
</tr>
<tr>
<td>Quadratic Cost Coefficient Spread</td>
<td>130 1505</td>
</tr>
<tr>
<td>Shrinkage Gradient Spread</td>
<td>0.012 0.042</td>
</tr>
<tr>
<td>Number of Risers and Chills</td>
<td>4 6</td>
</tr>
</tbody>
</table>

**TOTAL NUMBER OF OBSERVATIONS IN DATA SET = 16**

**DEPENDENT VARIABLE = Efficiency Measure of Algorithm**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQUARES</th>
<th>MEAN SQUARE</th>
<th>F VALUE</th>
<th>PR &lt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL</td>
<td>10</td>
<td>0.67235850</td>
<td>0.067235850</td>
<td>5.23</td>
<td>0.041</td>
</tr>
<tr>
<td>ERROR</td>
<td>5</td>
<td>0.06422650</td>
<td>0.012845300</td>
<td></td>
<td>STD DEV</td>
</tr>
<tr>
<td>CORRECTED TOTAL</td>
<td>15</td>
<td>0.73658500</td>
<td>0.011334</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>ANOVA SS.</th>
<th>F VALUE</th>
<th>PR &gt; F</th>
<th>R-SQUARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMB</td>
<td>1</td>
<td>0.131406</td>
<td>10.23</td>
<td>0.024</td>
<td>0.912805</td>
</tr>
<tr>
<td>QUAD</td>
<td>1</td>
<td>0.046872</td>
<td>3.65</td>
<td>0.114</td>
<td></td>
</tr>
<tr>
<td>SHRNK</td>
<td>1</td>
<td>0.004096</td>
<td>0.32</td>
<td>0.596</td>
<td></td>
</tr>
<tr>
<td>Z-ONE</td>
<td>1</td>
<td>0.220430</td>
<td>17.16</td>
<td>0.009</td>
<td></td>
</tr>
<tr>
<td>NUMB*QUAD</td>
<td>1</td>
<td>0.080089</td>
<td>6.23</td>
<td>0.054</td>
<td></td>
</tr>
<tr>
<td>NUMB*SHRNK</td>
<td>1</td>
<td>0.018360</td>
<td>1.43</td>
<td>0.285</td>
<td></td>
</tr>
<tr>
<td>NUMB*Z-ONE</td>
<td>1</td>
<td>0.159201</td>
<td>12.39</td>
<td>0.016</td>
<td></td>
</tr>
<tr>
<td>QUAD*SHRNK</td>
<td>1</td>
<td>0.008742</td>
<td>0.68</td>
<td>0.446</td>
<td></td>
</tr>
<tr>
<td>QUAD*Z-ONE</td>
<td>1</td>
<td>0.000081</td>
<td>0.01</td>
<td>0.939</td>
<td></td>
</tr>
<tr>
<td>SHRNK*Z-ONE</td>
<td>0.003080</td>
<td>0.24</td>
<td>0.645</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where:  
NUMB = number of risers and chills factor  
QUAD = Quadratic Cost Coefficient Factor  
SHRNK = Shrinkage Gradient Spread Factor  
Z-ONE = Zero-one Cost Coefficient Spread Factor

**Figure 46. Output From SAS - Optimization Efficiency**
As developed in Chapter V, the constraint to be added each iteration is: 
\[ y_0 - f_j^*(X) + \sum_{i=1}^{n} z_i [CZ_i - M(DEXIS_i)] \]
where: CZ\(_i\) = zero-one variable cost coefficient.

Along with others constructed in previous Benders iterations this constraint dictates the subproblem to optimize next. The larger the magnitude of the zero-one variable cost coefficient relative to other zero-one variable cost coefficients, the less likely the respective zero-one variable would be included in a successive subproblem.

As the standard deviation increases it becomes increasingly easier for the algorithm to better judge which combination to use to improve the solution. Finally, the shrinkage gradient spread seemed to have little effect on the performance of the algorithm.

It then seems that the performance of the algorithm can be reasonably predicted from the results of the experiment and from knowledge of the shrinkage gradient parameters, despite the fact that the optimization was not performed with an embedded shrinkage simulation. This certainly strengthens the position of accepting the hypothesis.

**Sensitivity**

In formulating the Benders constraints in the master problem the dual variables in the subproblem are needed.
discussed in Chapter V, the dual variables are determined via a heuristic algorithm that satisfies the Kuhn-Tucker conditions. As in linear programming the dual variables represent the instantaneous change in the objective function with respect to the change in the resource for the particular constraint considered. Consider the problem shown in Figure 47. The current bound on the height of the riser is four inches. The value of the dual variable corresponding to the constraint is 26,007. This indicates that the objective function could decrease if the bound on height is increased. Thus, if a larger flask could be used to increase the height slightly, a decrease in cost would be achieved. Because an increase in height allows the riser to stay liquid longer, the riser should be more effective and thus be able to feed a larger area. As can be seen in Figure 47, the increase in the riser height bound allowed a reduction in the chill size. This decreased the objective function by 23,310. This is close to the dual variable value. It need not be exactly the value as it would be in linear programming due to the fact that it only represents the instantaneous change in the objective function.

After the problem was solved for a bound of five inches the dual variable for height was found to be 20,623. Thus, an additional gain in the objective function could still be
Figure 47. Sensitivity Analysis Of Dual Variables Corresponding To Upper Bound Of Riser Height
obtained by modifying the height bound an additional inch. The bound was increased another inch and a decrease of 19,080 was achieved in the objective function. Again, this corresponds closely with the dual variable value.

If the algorithm were implemented in an interactive computer environment the subproblem dual variable could be used to identify and display which bounds would prove profitable to modify. The foundry engineer, knowing flask limitations, process limitations, etc., could vary the resources to attain an even more cost effective design.

**Steepest Descent Vs. Davidon-Fletcher-Powell**

Throughout this discussion the D-F-P algorithm was utilized. An example problem is worked out in Appendix C using this algorithm. Steepest Descent is similar to the D-P except that the D-F-P utilizes an approximation to the inverse Hessian. The two were compared. The reader should remember that the penalty method makes it somewhat difficult to compare the two. This is due to the directions chosen by the particular algorithm that is in different stages.

for any given penalty value the D-F-P solution at any stage in the optimization process was superior to the Steepest Descent solution.
Penalty Discussion

The initial penalty value for all optimization testing was 100. An experimental design was used to test for a change in the algorithm efficiency using an initial penalty of 10,000. The dependent variable was the number of penalty updates. This was chosen because the author felt that it best represented the advantage of using a higher penalty since a low number of penalty updates should imply a lower number of D-F-P iterations. The main effects tested were the quadratic function, standard deviation, and the standard deviation of the shrinkage gradient. In all cases the larger penalty value attained the same optimal value as did the smaller penalty value. These results are given in Table 4. The experimental design results are shown in Figure 48. There is insufficient data to draw any statistical conclusion. The results indicate that an increased initial penalty value could speed the convergence of the algorithm. This could be significant if in later research the solidification simulation was embedded within the optimization algorithm.

Convexity Problems Related To Golden Section

As discussed earlier, the Golden Section line search will allow variables to exceed either of their bounds, both upper and lower. The eigen values of the quadratic function
Table 4. Penalty Results

<table>
<thead>
<tr>
<th>QUADRATIC FUNCTION OBJECTIVE COEFFICIENT STANDARD DEVIATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH</td>
</tr>
<tr>
<td>HIGH</td>
</tr>
<tr>
<td>LOW</td>
</tr>
</tbody>
</table>

Results: Difference in number of penalty updates between small initial penalty and large initial penalty
The following is a condensed version of the output of a PROC ANOVA routine performed by the statistical package SAS.

**ANALYSIS OF VARIANCE PROCEDURE**

**CLASS LEVEL INFORMATION**

<table>
<thead>
<tr>
<th>CLASS LEVELS</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadratic Cost Coefficient Spread</td>
<td>2 0 1</td>
</tr>
<tr>
<td>Shrinkage Gradient Spread</td>
<td>2 0 1</td>
</tr>
</tbody>
</table>

TOTAL NUMBER OF OBSERVATIONS IN DATA SET = 4

**DEPENDENT VARIABLE = Number of Penalty Updates**

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>SUM SQUARES</th>
<th>MEAN SQUARE</th>
<th>F VALUE</th>
<th>PR &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL</td>
<td>2</td>
<td>0.003700</td>
<td>0.00185</td>
<td>4.63</td>
<td>0.3123</td>
</tr>
<tr>
<td>ERROR</td>
<td>1</td>
<td>0.000400</td>
<td>0.00040</td>
<td>STD DEV</td>
<td></td>
</tr>
<tr>
<td>CORRECTED TOTAL</td>
<td>3</td>
<td>0.004100</td>
<td></td>
<td></td>
<td>0.020</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>DF</th>
<th>ANOVA SS</th>
<th>F VALUE</th>
<th>PR &gt; F</th>
<th>R-SQUARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUAD</td>
<td>1</td>
<td>0.0036000</td>
<td>9.00</td>
<td>0.2048</td>
<td>0.9024</td>
</tr>
<tr>
<td>SHRINK</td>
<td>1</td>
<td>0.0001000</td>
<td>0.25</td>
<td>0.7048</td>
<td></td>
</tr>
</tbody>
</table>

Where: QUAD = Quadratic Cost Coefficient Factor  
SHRINK = Shrinkage Gradient Spread Factor

The data above represents the results from the experimental design to test the effect of the quadratic cost coefficients and the shrinkage gradient values upon the number of required penalty updates to arrive at an optimal solution.

Figure 48. Output From SAS - Penalty Experiment
all being positive imply that the matrix is positive
definite and in turn the objective function convex. However,
this assumes that all the variables are unbounded. Figure
49 shows a graph of the subjective function versus the line
search parameter $\alpha$ for both the conditions: (1) where
variables are allowed to be unbounded, and (2) where
variables are restricted to their bounds. As one can see,
the graph of the objective function where the variables are
held at their bounds is not convex. This can cause problems
with the Golden Section line search as the solution, for
the non-convex case could possibly not yield an improvement
in the objective function value. To overcome this a double
precision line search was implemented. It was assumed that
the non-convex condition occurred late in the optimization,
thus the penalty values were large, and in turn the search
directions were large. Due to the size of the search
directions the Golden Section optimization stopped in an
area where all variables were at their upper bounds. Thus,
the shrinkage constraint was over-satisfied, and a disimprove-
ment occurred in the objective function value. If this
occurred a double precision line search was conducted in the
first part of the feasible area. For all cases tested an
improvement in the objective function was found. It cannot
be proven that this occurs in all cases, but it seems to work
well for this particular application.
The Line Search Is

\[
\begin{bmatrix} W \\ H \\ P \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix} + a \begin{bmatrix} -3021.8 \\ -1482.3 \\ 53.52 \end{bmatrix}
\]

Cost = 200W + 1000W^2 + 100H + 800H^2 + 50P + 100P^2

+ 1.0E7(max(0, g(W, H, P)))^2

where:

\[ g(W, H, P) = 3.56 - 0.2W - 0.1H - 0.005P \]

subject to: 0 \leq W \leq 4 \quad 0 \leq H \leq 4 \quad P \geq 0

Figure 49. Convexity Problem
Dual Adequate Solution Procedure

The procedure for determining the dual adequate solution was discussed in Chapter V. The particular procedure described was not guaranteed to determine an optimal solution. It would also seem that this particular procedure would tend to bounce back and forth from one part of an algorithm to another and stay in an infinite loop. From the experience of solving nearly five-hundred sub-problems it has been shown that this is not the problem. The dual adequate procedure has in some cases decreased the objective function by thirty percent. However, usually it is a matter of a few percentage points.
Chapter VII
FUTURE RESEARCH

The purpose of this chapter is to provide suggestions for future research. This chapter was an easy one to construct since the research into optimization of rigging design via solidification models was a virgin topic. The suggestions are in no specific order and are discussed in the following separate sections.

Formulation Of Padding

As discussed in Chapters II and III, there are three major tools that can be utilized to obtain shrinkage defect-free castings. They are: (1) risers, (2) chills, and (3) padding. Both risers and chills were included in the research, but padding was not. As Figure 50 demonstrates, there are different ways in which padding can be incorporated into the two-dimensional design with the inclusion of a few variables and a few constraints. The variables could be:

\[ P = \text{position padding slants} \]
\[ \theta = \text{angle of inclination of padding} \]
\[ L = \text{length of padding} \]
Figure 50. Padding Formulation
Associated with padding, as with the continuous variables of risers and chills, would be the following bounding constraints: (1) a maximum angle of inclination, (2) a maximum length over which the padding could occur, and (3) the necessity of the padding butting against a side and the casting. Because of these factors the length of the padding and its position must sum to the bound over which it can occur. Mathematically, the constraints would be:

\[ \theta \leq \text{BOUND} \]
\[ \theta \leq \tan^{-1} \left( \frac{H}{L} \right) \]
\[ L \leq \text{BOUND} \]
\[ P + L \leq W \]

It will probably be difficult to determine the shrinkage gradients due to the solidification model sensitivity. This is because the grid may not vary sufficiently to provide a reliable shrinkage gradient. Obviously, this could be overcome by increasing the sensitivity (number of nodes per square inch) for the gradient calculation. However, as discussed in Chapter IV, this would greatly increase CPU time.

**Solidification Model-Gradient Update**

As discussed in Chapters V and VI, a major problem with this research was the unacceptable speed of the
solidification model. Before this research can be fully adopted the solidification model must be sped up. One possible method of accomplishing this would be to maintain the temperature of each node after the transient state has passed, for a number of iterations. Then, one would determine the gradient of temperature change/iteration for each node and use this to update a number of iterations. This could cut the solidification simulation time considerably. The author tried a few runs with this approach, but due to a limited amount of computer funds did not pursue it extensively. Another approach might be to apply some type of time series analysis to each node to try and forecast the future, possibly one hundred iterations ahead. If the solidification simulation run time could be decreased to a few seconds, many new optimization approaches could be implemented.

Gating Design

The model should be extended to include gating for the casting. Gating can also promote directional solidification if designed properly. However, one should note that gating also has a more important purpose—that of filling the casting cavity quickly while minimizing mold erosion. Thus, there could be a trade-off between directional solidification advantages of the gating design and that of mold erosion
and mold cavity filling pressure requirements. The gating formulation would be better suited for a three-dimensional design. Finally, the gating placement has similar properties as that of the riser and chill—it must be moldable, it must be located so as to minimize flask size (sand requirements), and its placement must be correct to avoid casting defects.

Three Dimensional Extension

First, the solidification model must be considerably faster before the suggestion of three-dimensional extension could even be seriously considered. However, the modeling of the problem should not be terribly difficult. Only one additional variable, the angle of location of the riser or chill, need be included. Figure 51 shows an orthogonal projection of a possible formulation. The additional variables would be:

\[ \theta = \text{angle of location} \]

\[ P = \text{position (inches), actually a radius} \]

\[ D = \text{diameter of riser} \]

\[ H = \text{height of riser} \]

As would be intuitively expected, there is only one additional variable to advance from two dimensions to three dimensions. The constraints would simply be bounds on each of the variables. They might be:
Figure 51. Three Dimension Model Extension
\[ D + P \leq \frac{W}{\sin \theta} \]

\[ D \leq \text{bound on diameter} \]

\[ \tan^{-1} \frac{D}{2L-D} \leq \theta \leq \tan^{-1} \frac{D}{2W-D} \]

The reader should note that the added constraints are nonlinear. However, due to the way that bounds are handled in the golden section line search this should not be a problem.

Also shown in Figure 51 is another formulation with two position variables, \( p_1 \) and \( p_2 \), each at ninety degrees to each other. The constraints for this formulation are simpler than the previous ones. However, the author feels it would be more difficult to quantify the shrinkage gradient meaningfully with this approach.

**Solidification Model**

Alternative solidification simulation models should be experimented with so as to speed up the solidification simulation time.

**Non-Simulation Solidification Models**

For the past fifty years or more empirical research has been conducted to try and determine equations for riser, chill, and padding design. Many foundrymen claim that they are eighty to ninety percent accurate in their calculations.
It may be possible to completely circumvent the simulation process and directly utilize the time-tested models as constraints. The riser or chill height, width, and position changes would simply allow for increased casting modulus (discussed in Chapter III) to be serviceable by the riser. This seems to hold great promise.

**Exothermic Riser Aids**

As the model presently exists, exothermic riser aids would be a simple formulation addition. Exothermic risering, discussed in Chapter II, is sleeves placed around risers which tend to insulate the riser thus extending its solidification time. A riser then could be made smaller with the addition of the exothermic sleeve and still service its originally intended area. To formulate this proposal an additional zero-one variable would need to be added for each riser which would have the possibility of using an exothermic sleeve. The additional formulation would be:

\[
\begin{align*}
    y_i &= \begin{cases} 
    0 & \text{if exothermic sleeve is not used with riser } i \\
    1 & \text{else}
    \end{cases} \\
    W_i' &= \text{width riser } i \text{ using sleeve} \\
    W_i &= \text{width riser } i \text{ else} \\
    H_i' &= \text{height riser } i \text{ using sleeve} \\
    H_i &= \text{height riser } i \text{ else}
\end{align*}
\]
\[ P_i^f = \text{position riser } i \text{ using sleeve} \]
\[ P_i^e = \text{position riser } i \text{ else} \]
\[ \min CY_i + f(W_i^f, H_i^f, P_i^f) + f(W_i, H_i, P_i) \]
subject to \[ W_i^f + H_i^f + P_i^f \leq MY_i \]
\[ W_i + H_i + P_i \leq (1-Y_i) M \]
\[ W_i + W_i^f + H_i + H_i^f + P_i + P_i^f \leq MZ_i \]

Notice the formulation allows either the riser with the exothermic sleeve to be used or the riser without this aid to be used. This eliminates putting two risers in the same place.

If the solidification models were to advance sufficiently, a variable could be added to actually determine the exothermic sleeve's height and thickness.

**Alloy Considerations**

Presently there are no considerations with respect to alloy type other than the specific heat and thermal conductivity. As discussed in Chapter II, some alloys have an extended mushy zone. Depending upon the degree of constitutional supercooling, this mushy zone could promote considerable dendrite formation. This dendrite formation tends to discourage the feeding capability of a riser. For increased detail this could be modeled.
**Isolated Shrinkage Cavities**

Throughout the text shrinkage cavities, if unacceptable, have been referred to as shrinkage cavity defects. This is because there could be areas of a casting in which shrinkage cavities could be allowed, and areas where they would be totally unacceptable. An example would be weights placed on tractors to maintain stability. It could be that only a certain area of this casting would need to be shrinkage-free. This would not be a difficult extension, and could be accomplished via the shrinkage cavity determination. The solidification simulation determines the shrinkage cavity size by counting the number of nodes remaining liquid after the total riser-casting contact has solidified. In shrinkage permissible areas the liquid nodes would simply be disregarded. The shrinkage gradients would then strive toward eliminating defects in non-permissible areas.

**Defect Analysis - Metal Penetration**

One of the most costly defects from a cost standpoint is that of metal penetration. This is a condition where the metal has a tendency to penetrate into the sand of the mold or core. One of the major factors effecting the severity of metal penetration is the length of time the molten metal is in contact with the sand. The longer the
contact the greater the chance for metal penetration. Thus, "hot spots", areas where the sand and metal are in contact at elevated temperatures for a prolonged period of time, could be identified and costs associated with the positioning of the riser could be appropriately applied. If the cost is sufficient it would prevent the positioning of a riser in a hot spot if it were to be a defect. The reader should take note that metal penetration is only a defect if it is unacceptable to the customer.

Defect Analysis - Hot Tearing

Hot tears are formed at elevated temperatures shortly after the solidification of the casting has started. According to most authorities this occurs during the mushy stage of solidification. If the mechanical properties of both the sand and metal were known as a function of temperature, some model could possibly be developed to try and minimize the occurrence of this defect. Similar to metal penetration, a cost could be associated with the defect and if the cost were sufficiently high the model should then prevent the defect.

Grain Size

Grain size is largely determined by the cooling rate of the casting. A chilled area usually has a fine grain structure. Grain size largely affects machinability, and
for most materials small grain size is desirable [4]. It would then seem possible to chill different areas of the casting which were to be machined later to promote the grain size desired for the particular application. It is possible that chilling an area could not only promote desirable directional solidification, but decrease grain size as well. This would provide better machinability. Of course, the opposite could also occur.

To conclude this chapter, the following is a brief list of topics one might consider as future research in the area of optimization of rigging design via solidification models:

1. macrosegregation considerations
2. sand density considerations and its effect on solidification time
3. incorporation of constitutional supercooling
4. blind versus open risers
5. internal chill usage
6. verifying models by actual costing production
7. various chill materials
BIBLIOGRAPHY


225


Appendix A
EXAMPLE PROBLEM - BENDERS' DECOMPOSITION

This appendix describes an example of the decomposition portion of the optimization algorithm. A sample problem involving four discrete (0-1) variables is solved. For ease of example, only continuous variables for width and height are modeled. Also for demonstration purposes, the objective function and constraints are linear.

A foundry engineer must suggest some set of discrete variables, risers and chills, the optimization algorithm can later choose from. For this example, consider a set of two risers and two chills. This would lead to a possibility of $2^4$, or sixteen different combinations the optimization package could test. The original casting is given below, along with, the foundry engineer's proposed set.
Inclusion of a riser or chill is represented by a discrete variable. It is included if the zero-one variable associated with it has value one. Otherwise, the riser or chill is not in the solution.

Each different continuous variable has associated with it an upper bound. The following drawing exhibits all the variable, both continuous and discrete, and their bounds.

The bound definitions are the same as those given in Chapter V. For clarity they are repeated below:

\[
\begin{align*}
W_{i}^\text{WIDBND}_i &= \text{Maximum allowable width for width}_i \\
H_{i}^\text{HGTBND}_i &= \text{Maximum allowable height for height}_i 
\end{align*}
\]
For this example consider the following values:

<table>
<thead>
<tr>
<th>i</th>
<th>WIDBND&lt;sub&gt;i&lt;/sub&gt;</th>
<th>HGBTBD&lt;sub&gt;i&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

The variable definitions are also the same as those given in Chapter V. Again, for clarity they are presented below:

\[ Z_i = \begin{cases} 
1 & \text{if discrete variable } i \text{ is included in a subproblem} \\
0 & \text{if discrete variable } i \text{ is not included in a subproblem} 
\end{cases} \]

\[ w_i = \text{variable representing width of variable } i \]

\[ h_i = \text{variable representing height of variable } i \]

\[ n = \text{number of discrete variables} \]

For this example problem the objective function is linear. The objective function is given below:

\[
\text{Cost} = \sum_{i=1}^{n} Z_i (ZC_i) + \sum_{i=1}^{n} w_i (\text{CWID}_i) + \sum_{i=1}^{n} h_i (\text{CHGT}_i)
\]

Where:

\[ ZC_i = \text{Cost coefficient associated with fixed cost of } Z_i \]

\[ \text{CWID}_i = \text{Cost coefficient associated with width for variable } i \]

\[ \text{CHGT}_i = \text{Cost coefficient associated with height for variable } i \]
For this example:

<table>
<thead>
<tr>
<th>i</th>
<th>ZC&lt;sub&gt;i&lt;/sub&gt;</th>
<th>CWID&lt;sub&gt;i&lt;/sub&gt;</th>
<th>CHGT&lt;sub&gt;i&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>300</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

The constraints discussed in Chapter V were divided into four categories: (1) shrinkage constraint, (2) upper bound constraint, (3) exist constraints, and (4) non-negativity constraint. Each is set up below:

**Shrinkage Constraint**

\[ g(w_1, w_2, \ldots, w_n, h_1, h_2, \ldots, h_n, P_1, P_2, \ldots, P_n) \leq \text{SHRINKAGE TOLERANCE} \]

For this example, a linear function is utilized as an approximation to the shrinkage function. Note that each coefficient in the linear approximation represents the partial derivative of shrinkage with respect to the width or height for each chill and riser. The constraint for this problem then is:

\[
\sum_{i=1}^{n} a_i w_i + \sum_{i=1}^{n} b_i h_i \leq \text{SHRINKTOL}
\]

Where for this example:

<table>
<thead>
<tr>
<th>i</th>
<th>a&lt;sub&gt;i&lt;/sub&gt;</th>
<th>b&lt;sub&gt;i&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Upper Bound Constraint

The upper bounds are:

\[ w_i \leq \text{WIDBND}_i \quad i=1,2...n \]

\[ h_i \leq \text{HGTBND}_i \quad i=1,2...n \]

For this example the bounds are:

\[ w_1 \leq 10 \]

\[ w_2 \leq 15 \]

\[ w_3 \leq 15 \]

\[ w_4 \leq 20 \]

\[ h_1 \leq 10 \]

\[ h_2 \leq 15 \]

\[ h_3 \leq 20 \]

\[ h_4 \leq 20 \]

Non-negativity Constraints

\[ w_i \geq 0 \quad i=1,2...n \]

\[ h_i \geq 0 \quad i=1,2...n \]

Existence Constraints

These constraints control the width and height accepting a value when their respective zero-one variable is not included in the subproblem. The constraint then is:

\[ w_i + h_i \leq 1000Z_i \quad i=1,2...n \]
The mathematical formulation for the complete problem then is:

\[
\begin{align*}
\text{min } & 100Z_1 + 200Z_2 + 300Z_3 + 400Z_4 + \\
& 10w_1 + 5h_1 + 10w_2 + 15h_2 + 5w_3 + h_3 + 2w_4 + h_4
\end{align*}
\]

\[
\begin{align*}
& u_1 \leq 10 \\
& u_2 \leq 15 \\
& u_3 \leq 15 \\
& u_4 \leq 20 \\
& u_5 \leq 10 \\
& u_6 \leq 15 \\
& u_7 \leq 20 \\
& u_8 \leq 20 \\
& u_9 - h_1 - w_2 - h_2 - w_3 - h_3 - w_4 - h_4 \leq -30 \\
& u_{10} w_1 + h_1 \leq 1000Z_1 \\
& u_{11} w_2 + h_2 \leq 1000Z_2 \\
& u_{12} w_3 + h_3 \leq 1000Z_3 \\
& w_4 + h_4 \leq 1000Z_4 \\
& u_{14} - w_1 \leq 0 \\
& u_{15} - w_2 \leq 0 \\
& u_{16} - w_3 \leq 0 \\
& u_{17} - w_4 \leq 0 \\
& u_{18} - h_1 \leq 0 \\
& u_{19} - h_2 \leq 0 \\
& u_{20} - h_3 \leq 0 \\
& u_{21} - h_4 \leq 0
\end{align*}
\]
The Kuhn-Tucker conditions are important for determining the dual variables that are necessary in developing constraints for the Benders master problem. The dual variables were shown on the previous formulation. In this example, they will be represented by the variable $u$. The Kuhn-Tucker conditions for this example are as follows:

<table>
<thead>
<tr>
<th>Objective Function</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10 + u_1$</td>
<td>$-u_9 + u_{10}$ $-u_{14}$</td>
</tr>
<tr>
<td>$10 + u_2$</td>
<td>$-u_9 + u_{11}$ $-u_{15}$</td>
</tr>
<tr>
<td>$5 + u_3$</td>
<td>$-u_9 + u_{12}$ $-u_{16}$</td>
</tr>
<tr>
<td>$2 + u_4$</td>
<td>$-u_9 + u_{13}$ $-u_{17}$</td>
</tr>
<tr>
<td>$5 + u_5$</td>
<td>$-u_9 + u_{10}$ $-u_{18}$</td>
</tr>
<tr>
<td>$15 + u_6$</td>
<td>$-u_9 + u_{11}$ $-u_{19}$</td>
</tr>
<tr>
<td>$1 + u_7$</td>
<td>$-u_9 + u_{12}$ $-u_{20}$</td>
</tr>
<tr>
<td>$1$</td>
<td>$-u_8 - u_9$ $-u_{13}$ $-u_{21}$</td>
</tr>
</tbody>
</table>

**Note:** All above conditions = 0


case

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1 (w_1 - 10) = 0$</td>
<td>$u_9 (30 - w_1 - w_2 - w_3 - w_4 - h_1 - h_2 - h_3 - h_4) = 0$</td>
</tr>
<tr>
<td>$u_2 (w_2 - 15) = 0$</td>
<td>$u_{10} (w_1 + h_1 - 1000Z_1) = 0$</td>
</tr>
<tr>
<td>$u_3 (w_3 - 15) = 0$</td>
<td>$u_{11} (w_2 + h_2 - 1000Z_2) = 0$</td>
</tr>
<tr>
<td>$u_4 (w_4 - 20) = 0$</td>
<td>$u_{12} (w_3 + h_3 - 1000Z_3) = 0$</td>
</tr>
<tr>
<td>$u_5 (h_1 - 10) = 0$</td>
<td>$u_{13} (w_4 + h_4 - 1000Z_4) = 0$</td>
</tr>
<tr>
<td>$u_6 (h_2 - 15) = 0$</td>
<td>$u_{14} (-w_1) = 0$ $u_{17} (-w_4) = 0$</td>
</tr>
<tr>
<td>$u_7 (h_3 - 20) = 0$</td>
<td>$u_{15} (-w_2) = 0$ $u_{18} (-h_1) = 0$ $u_{20} (-h_3) = 0$</td>
</tr>
<tr>
<td>$u_8 (h_4 - 20) = 0$</td>
<td>$u_{16} (-w_3) = 0$ $u_{19} (-h_2) = 0$ $u_{21} (-h_4) = 0$</td>
</tr>
</tbody>
</table>
Note: that the remaining Kuhn-Tucker conditions are the previously discussed constraints.

After constructing the general problem, the next step is to construct the master problem. It is as follows:

minimize $y_o$
$z \in \mathbb{Z}$
subject to:

$y_o \geq L^k = f(x^k) + 100z_1 + 200z_2 + 300z_3 + 400z_4$

- $u_{10}(1000z_1) - u_{11}(1000z_2)$
- $u_{11}(1000z_3) - u_{13}(1000z_4)$

where:

$k = 1, 2, \ldots, p$

$f(x^k) =$ objective function value evaluated at subproblem continuous variables only
$p =$ counter for Benders iterations

Simplifying the constraint:

$L = f(x^k) + z_1(100-1000u_{10}^k) + z_2(200-1000u_{11}^k)$

$+ z_2(300-1000u_{12}^k) + z_4(400-1000u_{13}^k)$

The subproblem is the general structure previously discussed with the exception that the zero-one variable will have values.

Before the optimization algorithm can start a series of initializations must occur. These preliminary steps then are:
1. solve the subproblem with all $Z_i = 1$

2. determine initial dual variable $u_i^1$

3. set $p$, the counter on the number of master problems solved = 1

4. set initial upper bound = UBD = current subproblem solution

Each of these are accomplished below:

The initial subproblem with all discrete variables included is given below:

$$\min 10w_1 + 5h_1 + 10w_2 + 15h_2 + 5w_3 + h_3 + 2w_4 + h_4$$

$$w_1 \leq 10$$

$$w_2 \leq 15$$

$$w_3 \leq 15$$

$$w_4 \leq 20$$

$$h_1 \leq 10$$

$$h_2 \leq 15$$

$$h_3 \leq 20$$

$$h_4 \leq 20$$

$$-w_1 - h_1 - w_2 - h_2 - w_3 - h_3 - w_4 - h_4 \leq -30$$
By inspection the optimal solution is:

\[
\begin{align*}
\text{w}_1 &= 0 & \text{h}_1 &= 0 \\
\text{w}_2 &= 0 & \text{h}_2 &= 0 & \text{Cost} = 1030 \\
\text{w}_3 &= 0 & \text{h}_3 &= 10 & f(X^1) = 30 \\
\text{w}_4 &= 0 & \text{h}_4 &= 20
\end{align*}
\]

Set up the Kuhn-Tucker conditions and solve for the dual variables:

\[
\begin{align*}
10 & -u_9 & -u_{14} & = 0 \\
10 & -u_9 & -u_{15} & = 0 \\
5 & -u_9 & -u_{16} & = 0 \\
2 & -u_9 & -u_{17} & = 0 \\
5 & -u_9 & -u_{18} & = 0 \\
15 & -u_9 & -u_{19} & = 0 \\
1 & -u_9 & & = 0 \\
1 + u_8 & -u_9 & & = 0
\end{align*}
\]
Dual Variable Solution:

\[
\begin{align*}
&u_1 = 0 & u_8 = 0 & u_{15} = 9 \\
&u_2 = 0 & u_9 = 1 & u_{16} = 4 \\
&u_3 = 0 & u_{10} = 0 & u_{17} = 1 \\
&u_4 = 0 & u_{11} = 0 & u_{18} = 4 \\
&u_5 = 0 & u_{12} = 0 & u_{19} = 4 \\
&u_6 = 0 & u_{13} = 0 & u_{20} = 0 \\
&u_7 = 0 & u_{14} = 9 & u_{21} = 0 \\
\end{align*}
\]

Set \( p = 1 \)

Set \( \text{UBD} = 1030 \)

**MASTER PROBLEM \( p = 1 \)**

Determine constraint to master problem:

\[
L^1 = 30 + Z_1 (100-1000u_{10}) + Z_2 (200-1000u_{11}) + Z_3 (300-1000u_{12}) + Z_4 (400-1000u_{13})
\]

The master problem then is:

\[
\begin{align*}
\text{minimize} & \quad y_o \\
\text{subject to} & \quad y_o \leq 30 + 100Z_1 + 200Z_2 + 300Z_3 + 400Z_4
\end{align*}
\]
Solving the master problem by complete enumeration, the possible combination values then are:

<table>
<thead>
<tr>
<th>$Z_1$</th>
<th>$Z_2$</th>
<th>$Z_3$</th>
<th>$Z_4$</th>
<th>$L_1$</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>930</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>130</td>
<td>MINIMUM</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>830</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1030</td>
<td>Initial Choice</td>
</tr>
</tbody>
</table>

Solve the following subproblem:

minimize $100 + 10w_1 + 5h_1$

subject to:

\[
\begin{align*}
w_1 & \leq 10 \\
h_1 & \leq 10 \\
-w_1 - h_1 & \leq -30 \\
w_1 + h_1 & \leq 1000 \\
w_1 & \geq 0 \\
h_1 & \geq 0
\end{align*}
\]
SOLUTION TO THE SUBPROBLEM IS INFEASIBLE - RETURN TO THE
MASTER PROBLEM AND DETERMINE A NEW COMBINATION

Resolve the master problem via complete enumeration:

<table>
<thead>
<tr>
<th>$z_1$</th>
<th>$z_2$</th>
<th>$z_3$</th>
<th>$z_4$</th>
<th>$L_1$</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>Infeasible</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>230</td>
<td>MINIMUM $\gamma_0 = 230$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>630</td>
<td>Infeasible Solution</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>930</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>830</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1030</td>
<td>Initial Choice</td>
</tr>
</tbody>
</table>

Perform the optimality check:

\[
\text{if } (\text{UBD} - \gamma_0) = 0 \begin{cases} \text{yes stop optimal solution} \\ \text{no continue} \end{cases}
\]

\[
\text{if } (1030 - 230) = 0 \text{ CONTINUE}
\]
Solve the following subproblem:

\[
\text{minimize } 200 + 10w_2 + 15h_2
\]

subject to:

\[
\begin{align*}
w_2 & \leq 15 \\
h_2 & \leq 15 \\
-w_2 & -h_2 \leq -30 \\
w_2 + h_2 & \leq 1000 \\
w_2 & \geq 0 \\
h_2 & \geq 0
\end{align*}
\]

The optimal solution is:

\[
\begin{align*}
w_1 &= 0 \\
h_1 &= 0 \\
w_2 &= 15 \\
h_2 &= 15 \\
w_3 &= 0 \\
h_3 &= 0 \\
w_4 &= 0 \\
h_4 &= 0
\end{align*}
\]

Cost = 575

\[f(X^2) = 375\]

Reassign the upper bound:

\[
\begin{array}{c|c}
\text{is Cost } \leq \text{ UBD} & \text{Yes UBD = Cost} \\
\text{No CONTINUE} & \\
\end{array}
\]

\[575 \leq 1030 \quad \text{UBD} = 575\]
Set up and solve the Kuhn-Tucker conditions:

Objective Function Gradient

\[ \begin{array}{ccc}
10 & -u_9 & +u_{10} & -u_{14} \\
10 & +u_2 & -u_9 & = 0 \\
5 & -u_9 & +u_{12} & -u_{16} \\
2 & -u_9 & +u_{13} & -u_{17} \\
5 & -u_9 & +u_{10} & -u_{18} \\
15 & +u_6 & -u_9 & = 0 \\
1 & -u_9 & +u_{12} & -u_{20} \\
1 & -u_9 & +u_{13} & -u_{21} \\
\end{array} \]

Note: \( u_g \) represents the shrinkage gradient for each constraint implicitly along with representing the dual variable associated with the shrinkage constraint.

Solution to the Kuhn-Tucker conditions are:
Determine constraint to add to master problem:

\[
L^2 = 375 + Z_1(100 - 1000u_{10}) + Z_2(200 - 1000u_{11}) \\
+ Z_3(300 - 1000u_{12}) + Z_3(400 - 1000u_{13}) \\
= 375 + Z_1(100 - 1000(10)) + Z_2(200 - 1000(0)) \\
+ Z_3(300 - 1000(14)) + Z_4(400 - 1000(14)) \\
= 375 - 9000Z_1 + 200Z_2 - 13700Z_3 - 13600Z_4 .
\]

Solving the master problem by complete enumeration, the possible combination values are:
<table>
<thead>
<tr>
<th>$Z_1$</th>
<th>$Z_2$</th>
<th>$Z_3$</th>
<th>$Z_4$</th>
<th>$L^1$</th>
<th>$L^2$</th>
<th>$\max(L^1, L^2)$</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td></td>
<td></td>
<td>Infeasible solution</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>430</td>
<td>-13225</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>330</td>
<td>-13325</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>730</td>
<td>-26925</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>230</td>
<td>575</td>
<td>575</td>
<td>Problem P = 2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>630</td>
<td>-13025</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>530</td>
<td>-13125</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>930</td>
<td>-26725</td>
<td>930</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>Infeasible Solution</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>530</td>
<td>-23125</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>430</td>
<td>-23225</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>830</td>
<td>-36825</td>
<td>830</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>330</td>
<td>-9325</td>
<td>330</td>
<td>MINIMUM $y_0 = 330$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>730</td>
<td>-22925</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>630</td>
<td>-23025</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1030</td>
<td>-36625</td>
<td>1030</td>
<td>Problem P = 1</td>
</tr>
</tbody>
</table>

Optimality check:

is $575 - 330 = 0$ No, CONTINUE

Solution to Subproblem (primal and dual solution):

$w_1 = 10 \quad h_1 = 10$

$w_2 = 10 \quad h_2 = 0 \quad \text{Cost} = 550$

$w_3 = 0 \quad h_3 = 0 \quad f(x^3) = 250$

$w_4 = 0 \quad h_4 = 0$

Reassign UBD:

is $575 \leq 550$ No, UBD = 550
MASTER PROBLEM $P = 3$

Determine constraint to add to master problem:

$$L^3 = 250 + 100Z_1 + 200Z_2 - 8700Z_3 - 8600Z_4$$

Solving the master problem by complete enumeration, the possible combination values are:

<table>
<thead>
<tr>
<th>$Z_1$</th>
<th>$Z_2$</th>
<th>$Z_3$</th>
<th>$Z_4$</th>
<th>$L^1$</th>
<th>$L^2$</th>
<th>$L^3$</th>
<th>max($L^1, L^2, L^3$)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>430</td>
<td>-13325</td>
<td>-8350</td>
<td>430</td>
<td>Infeasible</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>330</td>
<td>-13325</td>
<td>-8450</td>
<td>330</td>
<td>MINIMUM</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>730</td>
<td>-26925</td>
<td>-17050</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>230</td>
<td>575</td>
<td>450</td>
<td>575</td>
<td>Prob. P=2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>630</td>
<td>-13025</td>
<td>-8150</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>530</td>
<td>-13125</td>
<td>-8250</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>930</td>
<td>-26725</td>
<td>-16850</td>
<td>930</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Infeasible</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>530</td>
<td>-23125</td>
<td>-8250</td>
<td>530</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>430</td>
<td>-23225</td>
<td>-8350</td>
<td>430</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>830</td>
<td>-36825</td>
<td>-16950</td>
<td>830</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>330</td>
<td>-9373</td>
<td>550</td>
<td>550</td>
<td>Prob. P=3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>730</td>
<td>-22925</td>
<td>-8050</td>
<td>730</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>630</td>
<td>-23025</td>
<td>-8150</td>
<td>630</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1030</td>
<td>-36625</td>
<td>-16750</td>
<td>1030</td>
<td>Prob. P=1</td>
</tr>
</tbody>
</table>

$y_0 = 330$
Optimality Check:
\[ \text{is} (530 - 330) = 0 \quad \text{No, CONTINUE} \]

Subproblem Solution:
\[
\begin{align*}
    w_1 &= 0 & h_1 &= 0 \\
    w_2 &= 0 & h_2 &= 0 & \text{Cost} &= 370 \\
    w_3 &= 10 & h_3 &= 20 & f(X^4) &= 76 \\
    w_4 &= 0 & h_4 &= 0
\end{align*}
\]

Reset UBD
\[ \text{is} \ 370 \leq 550 \quad \text{Yes, UBD} = 370 \]

Results from Kuhn-Tucker conditions:
\[
\begin{align*}
    u_1 &= 0 & u_8 &= 0 & u_{15} &= 5 \\
    u_2 &= 0 & u_9 &= 5 & u_{16} &= 0 \\
    u_3 &= 0 & u_{10} &= 0 & u_{17} &= 1 \\
    u_4 &= 0 & u_{11} &= 0 & u_{18} &= 0 \\
    u_5 &= 0 & u_{12} &= 0 & u_{19} &= 10 \\
    u_6 &= 0 & u_{13} &= 4 & u_{20} &= 0 \\
    u_7 &= 4 & u_{14} &= 5 & u_{21} &= 0
\end{align*}
\]

MASTER PROBLEM \( P = 4 \)

Constraint added to the master problem:
\[ L^4 = 70 + 100Z_1 + 200Z_2 + 300Z_3 - 3600Z_4 \]

Solve the master problem by complete enumeration, the possible combination values are:
\[ M = \max(L^1, L^2, L^3, L^4) \]

<table>
<thead>
<tr>
<th>( Z_1 )</th>
<th>( Z_2 )</th>
<th>( Z_3 )</th>
<th>( Z_4 )</th>
<th>( L^1 )</th>
<th>( L^2 )</th>
<th>( L^3 )</th>
<th>( L^4 )</th>
<th>( M )</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Infeasible</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>430</td>
<td>-13225</td>
<td>-8350</td>
<td>-3530</td>
<td>430</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>330</td>
<td>-13325</td>
<td>-8450</td>
<td>370</td>
<td>370</td>
<td>MINIMUM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>730</td>
<td>-26925</td>
<td>-17050</td>
<td>-3230</td>
<td>730</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>230</td>
<td>575</td>
<td>450</td>
<td>270</td>
<td>575</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>630</td>
<td>-13025</td>
<td>-8150</td>
<td>-3330</td>
<td>630</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>530</td>
<td>-13125</td>
<td>-8225</td>
<td>570</td>
<td>570</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>930</td>
<td>-26725</td>
<td>-16850</td>
<td>-3030</td>
<td>930</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Infeasible</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>530</td>
<td>-23125</td>
<td>-8250</td>
<td>-3430</td>
<td>530</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>430</td>
<td>-23225</td>
<td>-8350</td>
<td>470</td>
<td>470</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>830</td>
<td>-36825</td>
<td>-16950</td>
<td>-3130</td>
<td>830</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>330</td>
<td>-9325</td>
<td>550</td>
<td>370</td>
<td>550</td>
<td>Prob. P = 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>730</td>
<td>-22925</td>
<td>-8050</td>
<td>-3230</td>
<td>730</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>630</td>
<td>-23025</td>
<td>-8150</td>
<td>670</td>
<td>670</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1030</td>
<td>-36625</td>
<td>-16750</td>
<td>-2930</td>
<td>1030</td>
<td>Prob. P = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ y_0 = 370 \]

Optimality Check:

\[ |(370 - 370)| = 0 \] YES STOP OPTIMAL SOLUTION

Optimal Solution is:

\[
\begin{align*}
  w_1 &= 0 \\
  w_2 &= 0 \\
  w_3 &= 10 \\
  w_4 &= 0 \\
  h_1 &= 0 \\
  h_2 &= 0 \\
  h_3 &= 20 \\
  h_4 &= 0 \\
  Z_1 &= 0 \\
  Z_2 &= 0 \\
  Z_3 &= 1 \\
  Z_4 &= 0 
\end{align*}
\]
Appendix B

EXAMPLE PROBLEM - PENALTY METHOD

In this appendix, presents a brief example demonstrating the penalty method. As discussed in Chapter V, a penalty can convert a constrained optimization problem into an approximating unconstrained optimization problem. Consider the following one-dimensional problem:

\[
\text{minimize } f(x) = 4x + 2x^2 \\
\text{subject to } g(x) = (4 - x) \leq 0
\]

A graph of the constrained problem is shown below. Note the cross-hatch area is the feasible region.
The penalty method converts the constrained problem into the following approximating unconstrained problem:

$$\minimize P(x) = f(x) + u[\max(0,g(x))]^2$$

The simplicity of the example problem allows the solution to be determined via first order necessary conditions.

A number of different penalties, in increasing order, were tested. Some are listed below.

As an example allow $u = 0$:

$$P(x) = 4x + 2x^2$$

$$\frac{dP(x)}{dx} = 4 + 4x = 0$$

$$x = -4$$

As an example allow $u = 20$:

$$P(x) = 4x + 2x^2 + 20(4 - x)^2$$

$$\frac{dP(x)}{dx} = 44x - 156 = 0$$

$$x = 3.545$$

As an example allow $u = 30$:

$$P(x) = 4x + 2x^2 = 30(4 - x)^2$$

$$\frac{dP(x)}{dx} = 64x - 236 = 0$$

$$x = 3.68$$

By inspection of the original problem, one can easily see that the optimal solution is:

$$x^* = 4 \quad f(x^*) = 48$$
Shown below are the results using a variety of penalties. As can be seen, the larger the initial penalty the faster one converges to the optimal solution. However, one should be cautioned that large penalties tend to lead to illdefined functions.

<table>
<thead>
<tr>
<th>u</th>
<th>x</th>
<th>f(x)</th>
<th>P(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-4.00</td>
<td>16.0</td>
<td>16.0</td>
</tr>
<tr>
<td>10</td>
<td>3.166</td>
<td>32.7</td>
<td>39.56</td>
</tr>
<tr>
<td>20</td>
<td>3.54</td>
<td>39.22</td>
<td>43.45</td>
</tr>
<tr>
<td>30</td>
<td>3.68</td>
<td>41.80</td>
<td>44.87</td>
</tr>
<tr>
<td>40</td>
<td>3.76</td>
<td>43.32</td>
<td>45.62</td>
</tr>
<tr>
<td>100</td>
<td>3.90</td>
<td>46.02</td>
<td>47.02</td>
</tr>
<tr>
<td>1000</td>
<td>3.990</td>
<td>47.80</td>
<td>47.90</td>
</tr>
<tr>
<td>10000</td>
<td>3.999</td>
<td>47.86</td>
<td>47.87</td>
</tr>
</tbody>
</table>
Appendix C
EXAMPLE PROBLEM - DAVIDON-FLETCHER-POWELL

This appendix solves an optimization problem using the D-F-P algorithm. The example is taken form Himmelblau [46].

Consider the problem:

Minimize $4(x_1 - 5)^2 + (x_2 - 6)^2$

The recursion relation to update $X$ then is:

$$X^{k+1} = X^k + \alpha^k \nabla f(X^k)$$

where:

$$\nabla f(X^k) = \begin{bmatrix} 8(x_1 - 5) \\ 2(x_2 - 6) \end{bmatrix}$$

$S^k$ is derived as the approximation to the Hessian.

The derivation can be found in Chapter V.

To start the algorithm:

$$S^k = I$$

$$X^o = \begin{bmatrix} 8 \\ 9 \end{bmatrix} \quad f(X^o) = 45$$

The new $X$ vector is computed from:

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} 8 \\ 9 \end{bmatrix} - \alpha \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 24 \\ 6 \end{bmatrix}$$
A line search is performed resulting in $\alpha^* = 0.1307$.

The new $x'$ vector then is:

$$
\begin{bmatrix}
4.862 \\
8.215
\end{bmatrix}
$$

and $f(x') = 4.985$.

The new search direction is determined by:

$$
\begin{bmatrix}
-3.13 & 0 \\
0 & 1
\end{bmatrix}
+ 
\begin{bmatrix}-25.108 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}3.13 & -0.785 \\
-0.785 & 0
\end{bmatrix}
\begin{bmatrix}
-25.108 \\
-1.569
\end{bmatrix}
\begin{bmatrix}0 & 1 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}1.27 \times 10^{-1} & -3.149 \times 10^{-2} \\
-3.149 \times 10^{-2} & 1.0038
\end{bmatrix}
\begin{bmatrix}1 & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}1.27 \times 10^{-1} & -3.149 \times 10^{-2} \\
-3.149 \times 10^{-2} & 1.0038
\end{bmatrix}
\begin{bmatrix}1 & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}-1.108 \\
4.431
\end{bmatrix}
$$

$x^2$ can then be computed by:

$$
\begin{bmatrix}
x_1^2 \\
x_2^2
\end{bmatrix}
= 
\begin{bmatrix}4.862 \\
8.215
\end{bmatrix}
- \alpha^*
\begin{bmatrix}1.27 \times 10^{-1} & -3.149 \times 10^{-2} \\
-3.149 \times 10^{-2} & 1.0038
\end{bmatrix}
\begin{bmatrix}-1.108 \\
4.431
\end{bmatrix}
$$

A line search is performed with the resulting $\alpha^* = 0.494$.

This yields $x^2$ as:
\[ X^2 = \begin{bmatrix} 5 \\ 6 \end{bmatrix} \quad f(X^2) = 9.06 \times 10^{-15} \]

A final iteration results in no improvement in the objective function. The optimal solution then is:

\[ X^* = \begin{bmatrix} 5 \\ 6 \end{bmatrix} \quad f(X^*) = 0.0 \]

A comparison of the D-F-P approximation to the inverse Hessian and the true inverse Hessian is shown below. The correlation, obviously, is excellent.

\[
\begin{bmatrix}
0.125 & 0 \\
0 & 0.5
\end{bmatrix}
\approx
\begin{bmatrix}
0.125 & 1.387 \times 10^{-17} \\
-1.387 \times 10^{-17} & 0.5
\end{bmatrix}
\]

Finally the trajectory of the search is given below:
Appendix D

EXAMPLE PROBLEM - GOLDEN SECTION LINE SEARCH TECHNIQUE

In this appendix, a simple Golden Section line search is performed. The procedure for the search is discussed in Chapter V.

Consider the following problem:

minimize \( f(x) = 3x^2 + 4x - 5 \)

The graph \( f(x) \) versus \( x \) is shown below. As one can see the function is convex.
Assuming the bracket of the minimum to be: \([0, 6]\) the search is performed:

Allow \(A=0\) and \(B=6\)

**Iteration Number 1**

\[ u_1 = 0.382(B-A) + A = 0.382(6-0) + 0 = 2.292 \]

\[ f(u_1) = 11.59 \]

\[ u_2 = 0.618(B-A) + A = 0.618(6-0) + 0 = 3.708 \]

\[ f(u_2) = 31.42 \]

**Iteration Number 2**

Because \(f(u_1) < f(u_2)\)

\[ B = u_2 = 3.708 \]

\[ u_2 = u_1 = 2.292 \]

\[ f(u_2) = f(u_1) = 11.59 \]

Then:

\[ u_1 = 0.382(3.708-0) + 0 = 1.416 \]

\[ f(u_1) = 5.35 \]

**Iteration Number 3**

Because \(f(u_1) < f(u_2)\)

\[ B = u_2 = 2.292 \]

\[ u_2 = u_1 = 1.416 \]

\[ f(u_2) = f(u_1) = 5.35 \]

Then:

\[ u_1 = 0.382(2.292-0) + 0 = 0.8755 \]

\[ f(u_1) = 3.797 \]
Iteration Number 4

Because \( f(u_1) = f(u_2) \)
\[
B = u_2 = 1.416
f(u_2) = f(u_1) = 3.797
u_2 = u_1 = .8755
\]

Then:
\[
u_1 = .382(1.416-0) + 0
= .541
f(u_1) = 3.714
\]

Iteration Number 5

Because \( f(u_1) = f(u_2) \)
\[
B = u_2 = .8755
u_2 = u_1 = .541
f(u_2) = f(u_1) = 3.714
\]

Then:
\[
u_1 = .382(.8755-0) + 0
= .3344
f(u_1) = 3.998
\]

Iteration Number 6

Because \( f(u_2) = f(u_1) \)
\[
A = u_1 = .3344
u_1 = u_2 = .541
f(u_1) = f(u_2) = 3.714
\]

Then:
\[
u_2 = .618(.8755-.3344) + .3344
= .668
f(u_2) = 3.66
\]
The search direction as they occurred are shown in the graph below in the region of the optimum.
Appendix E

COMPUTER PROGRAM LISTINGS

The FORTRAN listings of the two programs used in this research are contained in this Appendix. The first, SETUP, is a program used to align risers and chills to the respective casting geometry. Its output is used in the second program, OPTUM. In this program all of the optimization procedures are performed. Also in the program the solidification simulation is performed.

The program SETUP is given on the following pages.
SETUP

1. // JOB
2. // TIME=(1,01),REGION=256K
3. /*JOBPARM LINES=2000
4. /*JOBPARM V=E
5. // EXEC FTG1CG,TIME=(1,00)
6. //SYSPRINT DD DUMMY
7. //FORT. SYSPRINT DD *
8. COMMON /X/X(60,60)
9. INTEGER*2 X
10. DO 55 I=1,60
11. DO 65 J=1,60
12. X(I,J)=0
13. 65 CONTINUE
14. 55 CONTINUE
15. CALL INPUT
16. CALL INPUT2
17. CALL INPUT3
18. CALL SETUP
19. CALL COMBIN
20. CALL CENTER
21. CALL OUTPUT
22. CALL SAND
23. CALL OUTPUT
24. CALL FLASK
25. CALL OUTPUT
26. CALL OUTPUT1
27. STOP
28. END
29. SUBROUTINE INPUT
30. COMMON /IL/ILI20)
31. COMMON /IW/IW(20)
32. COMMON /NUMCON/NUMCON
33. COMMON /CONTAC/CONTAC(20,2)
34. INTEGER CONTAC
35. COMMON /CONINF/CONINF(20,2,2)
36. INTEGER CONINF
37. COMMON /NUMREC/NUMREC
38. INTEGER NUMREC
39. READ(5,10) NUMREC
40. 10 FORMAT(I2)
41. DO 20 I=1,NUMREC
42. 20 CONTINUE
43. READ(5,21) IL(I),IW(I)
44. 21 FORMAT(I2)
45. 20 CONTINUE
46. READ(5,30) NUMCON
47. 30 FORMAT(I2)
48. DO 50 I=1,NUMCON
49. 50 CONTINUE
50. READ(5,51) (CONTAC(I,J),J=1,2)
51. 51 FORMAT(2I2)
52. 50 CONTINUE
53. RETURN
54. END
55. SUBROUTINE INPUT2
56. COMMON /NOZERO/NOZERO
57. INTEGER NOZERO
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /ZCON/ZCON(10)
INTEGER ZCON
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /ZTYPE/ZTYPE(10)
INTEGER ZTYPE
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
READ(5,10) NOZERO
10 FORMAT(121
00 100 1=1iNOZERO
READ(5,101) ZTYPE(1),ZBOUND(1),ZSIDE(1),ZREC(I),ZCON(I)
101 FORMAT(5(I2.1X))
CONTINUE
RETURN
END
SUBROUTINE INPUT3
COMMON /SAMOL/SANDL
INTEGER SANOL
COMMON /SANQW/SANOW
INTEGER SANDW
READ(5,10) SANDL,SANDW
10 FORMAT(212)
RETURN
END
SUBROUTINE SETUP
COMMON /IL/IL(120)
COMMON /IW/IW(120)
COMMON /NUMCON/NUMCON
INTEGER CONTAC
COMMON /CONINF/CONINF(20,2,2)
INTEGER CONINF
COMMON /NUMREC/NUMREC
INTEGER NUMREC
COMMON /X/X(60,60)
INTEGER*2 X
COMMON /RECEST/RECEST(20)
INTEGER RECEST
COMMON /CORNER/CORNER(20,4,2)
INTEGER CORNER
COMMON /NEW/NEW
COMMON /GLD/GLOD
INTEGER NEW OLD

C SET UP FIRST RECTANGLE
C
CORNER(1,1,1)=41-IW(1)
CORNER(1,1,2)=30
CORNER(1,2,1)=CORNER(1,1,1)
CORNER(1,2,2)=30+IL(1)-1
CORNER(1,3,1)=40
CORNER(1,3,2)=30
CORNER(1,4,1)=40
CORNER(1,4,2)=30+IL(1)-1
II=IW(1)
JL=IL(1)
I1 = 1
J1 = 1
I2 = 41-1
J2 = 29+J
2 CONTINUE
1 CONTINUE
RECEST(I) = 1
DO 100 I=1, NUMCON
1 IF(RECEST(CONTAC(I,1)).EQ.1) NEW=2
2 IF(RECEST(CONTAC(I,1)).EQ.1) OLD=1
3 IF(RECEST(CONTAC(I,2)).EQ.1) NEW=1
4 IF(RECEST(CONTAC(I,2)).EQ.1) OLD=2
5 IF(CONINF(I,NEW,1).EQ.4 .AND. CONINF(I,OLD,1).EQ.1) CALL ONE(I)
6 IF(CONINF(I,NEW,1).EQ.2 .AND. CONINF(I,OLD,1).EQ.3) CALL TWO(I)
7 IF(CONINF(I,NEW,1).EQ.3 .AND. CONINF(I,OLD,1).EQ.2) CALL THREE(I)
8 IF(CONINF(I,NEW,1).EQ.1 .AND. CONINF(I,OLD,1).EQ.4) CALL FOUR(I)
100 CONTINUE
RETURN
END
SUBROUTINE ONE(I)
COMMON /OLD/ OLD
INTEGER OLD
COMMON /NEW/ NEW
INTEGER NEW
COMMON /IL/ IL(20)
COMMON /IW/ IW(20)
COMMON /CONTAC/ CONTAC(20,2)
INTEGER CONTAC
COMMON /CONINF/ CONINF(20,2,2)
INTEGER CONINF
COMMON /NUMREC/ NUMREC
COMMON /X/X(60,60)
INTEGER X
COMMON /RECEST/ RECEST(20)
INTEGER RECEST
COMMON /CORNER/ CORNER(20,4,2)
INTEGER CORNER
INTEGER NEWC, OLD
NEWC=NEW
OLD=OLD
NEW=CONTAC(I,NEW)
OLD=CONTAC(I,OLD)
NEWC=NEW
OLD=OLD
NEW=CONTAC(I, NEW)
OLD=CONTAC(I, OLD)
C NEW TOP OLD BOTTOM
C
CORNER(NEW,3, 1) = CORNER(OLD,3, 1) - 1
CORNER(NEW,3, 1) = CORNER(NEW,3, 1)
CORNER(NEW,2, 1) = CORNER(OLD,2, 1) - IW(NEW)
CORNER(NEW, 2, 1) = CORNER(OLD,2, 1) - IW(NEW)
CORNER(NEW,4, 1) = CORNER(OLD,4, 1) + CONINF(I, OLD, 2)
CORNER(NEW,4, 1) = CORNER(NEW, 1, 2)
CORNER(NEW,2, 2) = CORNER(NEW,2, 2)
CORNER(NEW,4, 2) = CORNER(NEW,4, 2)
RECEST(NEW) = 1
CALL SET
RETURN
END
SUBROUTINE TWO(I)
COMMON /OLD/ OLD
INTEGER OLD
COMMON /NEW/NEW
INTEGER NEW
COMMON /IL/IL(20)
COMMON /IW/IW(20)
COMMON /CONTAC/CONTAC(20,2)
INTEGER CONTAC
COMMON /CONINF/CONINF(20,2,2)
INTEGER CONINF
COMMON /NUMREC/NUMREC
COMMON /X/X(60,60)
INTEGER NEWC.OLDC
NEWC=NEW
OLDC=ULD
NEW=CONTAC(1,NEW)
OLDC=CONTAC(1,OLD)
C = LEFT NEW = RIGHT
C SET UP CORNERS
C SET UP J
C

SUBROUTINE THREE(I)
COMMON /OLD/OLD
INTEGER OLD
COMMON /NEW/NEW
INTEGER NEW
COMMON /IL/IL(20)
COMMON /IW/IW(20)
COMMON /CONTAC/CONTAC(20,2)
INTEGER CONTAC
COMMON /CONINF/CONINF(20,2,2)
INTEGER CONINF
COMMON /NUMREC/NUMREC
COMMON /X/X(60,60)
INTEGER NEWC.OLDC
NEWC=NEW
OLDC=ULD
NEW=CONTAC(1,NEW)
OLDC=CONTAC(1,OLD)
C = LEFT NEW = RIGHT
C SET UP CORNERS
C SET UP J
C

CALL SET
RETURN
END
260. COMMON /CORNER/CORNER(20,4,2)
261. INTEGER CORNER
262. INTEGER NEWC,OLDc
263. NEWC=NEW
264. OLDc=OLD
265. NEW=CONTAC(1,NEW)
266. OLD=CONTAC(1,OLD)
267. C
268. C OLD RIGHT NEW LEFT
269. C
270. C
271. C SET UP CORNERS
272. C
273. C
274. C SET UP I
275. C
276. CORNER(NEW,2,1)=CORNER(OLD,1,1)+CONINF(1,OLDc,2)
277. CORNER(NEW,4,1)=CORNER(NEW,2,1)+IW(NEW)-1
278. CORNER(NEW,1,1)=CORNER(NEW,2,1)
279. CORNER(NEW,3,1)=CORNER(NEW,4,1)
280. C
281. C SET UP J
282. C
283. CORNER(NEW,2,2)=CORNER(OLD,1,2)-1
284. CORNER(NEW,4,2)=CORNER(NEW,2,2)
285. CORNER(NEW,1,2)=CORNER(NEW,2,2)-IL(NEW)+1
286. CORNER(NEW,3,2)=CORNER(NEW,1,2)
287. RECEST(NEW)=1
288. CALL SET
289. RETURN
290. END
291. SUBROUTINE FOUR(1)
292. COMMON /OLD/OLD
293. INTEGER OLD
294. COMMON /NEW/NEW
295. INTEGER NEW
296. COMMON /IL/IL(20)
297. COMMON /IW/IW(20)
298. COMMON /CONTAC/CONTAC(20,2)
299. INTEGER CONTAC
300. COMMON /CONINF/CONINF(20,2,2)
301. INTEGER CONINF
302. COMMON /NUMREC/NUMREC
303. COMMON /X/X(60,60)
304. INTEGER*2 X
305. COMMON /RECEST/RECEST(20)
306. INTEGER RECEST
307. COMMON /CORNER/CORNER(20,4,2)
308. INTEGER CORNER
309. INTEGER NEWC,OLDc
310. NEWC=NEW
311. OLDc=OLD
312. NEW=CONTAC(1,NEW)
313. OLD=CONTAC(1,OLD)
314. C
315. C OLD TOP NEW BOTTOM
316. C
317. C
318. C SET UP CORNER
319. C
SUBROUTINE SET

corner(NEW,1,1) = corner(OLD,3,1) + 1

corner(NEW,2,1) = corner(NEW,1,1)

corner(NEW,3,1) = corner(NEW,1,1) + IWIN(NEW) - 1

corner(NEW,4,1) = corner(NEW,3,1)

corner(NEW,1,2) = corner(NEW,1,2) + CQNRINF(I,OLOC,2)

corner(NEW,3,2) = corner(NEW,1,2)

corner(NEW,2,2) = corner(NEW,1,2) + IWIN(NEW) - 1

corner(NEW,4,2) = corner(NEW,2,2)

call set

recest(NEW) = 1

return

end

SUBROUTINE SET

corner(NEW,1,1) = corner(OLD,3,1) + CQNINF(I,OLOC,2)

corner(NEW,2,1) = corner(NEW,1,1)

corner(NEW,3,1) = corner(NEW,1,1) + IWIN(NEW) - 1

corner(NEW,4,1) = corner(NEW,3,1)

call set

recest(NEW) = 1

return

end

SUBROUTINE CQMSIN

common /new/new

integer new

common /x/x(60,60)

integer*2 x

common /corner/corner(20,4,2)

integer corner

i1 = corner(NEW,1,1)

i2 = corner(NEW,3,1)

j1 = corner(NEW,1,2)

j2 = corner(NEW,2,2)

do 100 is = i1, i2

do 110 js = j1, j2

x(is, js) = 2

110 continue

100 continue

return

end

SUBROUTINE COMBIN

common /nozero/nozero

integer nozero

common /zrec/zrec(10)

integer zrec

common /zcon/zcon(10)

integer zcon

common /zside/zside(10)

integer zside

common /ztype/ztype(10)

integer ztype

common /zbound/zbound(10)

integer zbound

common /corner/corner(20,4,2)

integer corner

common /x/x(60,60)

integer*2 x

common /zcorner/zcorner(10,4,2)

integer zcorner

do 10 i = 1, nozero

if (zcon(i) .eq. 1 .and. zside(i) .eq. 4) go to 100

if (zcon(i) .eq. 4 .and. zside(i) .eq. 1) go to 200

if (zcon(i) .eq. 2 .and. zside(i) .eq. 3) go to 300

if (zcon(i) .eq. 3 .and. zside(i) .eq. 2) go to 400

rec down z variable up
C REC UP Z VARIABLE DOWN

GO TO 500

C Z LEFT RECTANGLE RIGHT

GO TO 500

C Z VARIABLE RIGHT RECTANGLE RIGHT

DO 510 IS=U,I2

DO 511 JS=J1,J2

X(IS,JS)=ZTYPE1)

511 CONTINUE

510 CONTINUE

ZCORNR(I,1,1)=I1
ZCORNR(I,2,1)=I1
ZCORNR(I,3,1)=I2
ZCORNR(I,4,1)=I2
ZCORNR(I,1,2)=J1
ZCORNR(I,2,2)=J2
ZCORNR(I,3,2)=J1
ZCORNR(I,4,2)=J2

10 CONTINUE

RETURN
INTEGER NUMREC

DO 10 I=1,60
   DO 20 J=1,60
      IF(I,J).EQ.2) GO TO 25
      IF(I,J).EQ.3) GO TO 25
      IF(I,J).EQ.4) GO TO 25
   CONTINUE
   10 CONTINUE

ISTART=1

DO 30 I=ISTART,60
   DO 40 J=1,60
      IF(I,J).EQ.2) GO TO 30
      IF(I,J).EQ.3) GO TO 30
      IF(I,J).EQ.4) GO TO 30
   CONTINUE
   25 ISTART=I
   CONTINUE

GO TO 45

30 CONTINUE

45 IEN=1

WIDTH=IEN-ISTART
   WIDTH+WIDTH+2+SAN0W+SANDW

I2=ISTART

DO 50 I=1,WIDTH
   I1=I+1+SAN0W
   DO 60 J=1,60
      X(I,J)=X(I2,J)
   CONTINUE
   50 CONTINUE

GO TO 45

46 I2=I2+1

CONTINUE

50 CONTINUE

GO TO 45

48 J=1,60

DO 100 I=1,60
   DO 110 J=1,60
      IF(I,J).EQ.2) GO TO 115
      IF(I,J).EQ.3) GO TO 115
      IF(I,J).EQ.4) GO TO 115
   CONTINUE
   100 CONTINUE

GO TO 100

115 JSTART=J

DO 120 J=JSTART,60
   DO 130 I=1,WIDTH
      IF(I,J).EQ.2) GO TO 120
      IF(I,J).EQ.3) GO TO 120
      IF(I,J).EQ.4) GO TO 120
   CONTINUE
   120 CONTINUE

GO TO 120

125 JEND=J

CLENGT=JEND-JSTART
   LENGTH+CLENGT+2+SAN0W+SANDL

J2=JSTART

DO 150 J=1,CLENGT
   J1=J+1+SAN0W
   DO 160 I=1,60
      X(I,J1)=X(I,J2)
   CONTINUE
   150 CONTINUE

J2=J2+1

CONTINUE

160 J2=J2+1

CONTINUE

150 J1=LENGTH-SAN0W

J2=60

DO 500 I=1,60
   DO 510 J=J1,J2
      X(I,J)=0
I1 = WIDTH - SANDW
I2 = 60
DO 520 J = 1, 60
    DO 530 I = 11, 12
    XI(I, J) = 0
    520 CONTINUE
530 CONTINUE
510 CONTINUE

I2 = ISTART - 1 - SANDW - 1
DO 200 I = 1, NUMREC
    DO 210 J = 1, 4
        CORNER(I, J, 1) = CORNER(I, J, 1) - 12
    CONTINUE
200 CONTINUE
210 CONTINUE

J2 = JSTART - 1 - SANDW
DO 240 I = 1, NUMREC
    DO 250 J = 1, 4
        CORNER(I, J, 2) = CORNER(I, J, 2) - J2
    CONTINUE
240 CONTINUE
250 CONTINUE

DO 260 I = 1, NUMREC
    DO 270 J = 1, 4
        ZCORNER(I, J, 1) = ZCORNER(I, J, 1) - I2
    CONTINUE
260 CONTINUE
270 CONTINUE

DO 300 I = 1, NUMREC
    WRITE(6, 301) I
    301 FORMAT(' ' , 'RECTANGLE NUMBER ', 13)
    DO 310 J = 1, 60
        WRITE(6, 302) J, CORNER(I, J, 1), CORNER(I, J, 2)
        302 FORMAT(' ', 'CORNER ', 12, ' I COMPONENT ', 12, ' J COMPONENT ', 12)
    CONTINUE
300 CONTINUE
310 CONTINUE

DO 350 I = 1, NUMREC
    WRITE(6, 351) I
    351 FORMAT(' ', 'ZERO ONE VARIABLE ', 12, ' CORNER INFORMATION ',)
    DO 360 J = 1, 60
        WRITE(6, 362) J, ZCORNER(I, J, 1), ZCORNER(I, J, 2)
        362 FORMAT(' ', 'CORNER ', 12, ' I COMPONENT ', 12, ' J COMPONENT ', 12)
    CONTINUE
350 CONTINUE
360 CONTINUE

RETURN
END

SUBROUTINE F

COMMON /LENGTH/LENGTH
COMMON /WIDTH/WIDTH
COMMON /X/X(160, 60)
DO 10 J = 1, 60
    X(I, J) = 0
10 CONTINUE

SUBROUTINE FLAS
SUBROUTINE SANON
COMMON /LENGTH/LENGTH
INTEGER LENGTH
COMMON /WIDTH/WIDTH
INTEGER WIDTH
COMMON/X/X(60,60)
INTEGER*2X
DO 10 I=1,WIDTH
DO 20 J=1,LENGTH
IF(X(I,J).EQ.2) GO TO 20
IF(X(I,J).EQ.3) GO TO 20
IF(X(I,J).EQ.4) GO TO 20
X(I,J)=1
20 CONTINUE
10 CONTINUE
RETURN
END

SUBROUTINE OUTPUT
COMMON /LENGTH/LENGTH
INTEGER LENGTH
COMMON /WIDTH/WIDTH
INTEGER WIDTH
COMMON/X/X(60,60)
INTEGER*2X
IOUT=6
WRITE(IOUT,99)
99 FORMAT(' ',1,' ',' ')
DO 10 I=1,WIDTH
WRITE(IOUT,100) (X(I,J),J=1,LENGTH)
100 FORMAT(' ',6011)
10 CONTINUE
RETURN
END

SUBROUTINE OUTPUT1
COMMON/X/X(60,60)
INTEGER*2X
COMMON /LENGTH/LENGTH
INTEGER LENGTH
COMMON /WIDTH/WIDTH
INTEGER WIDTH
COMMON /NUMREC/NUMREC
INTEGER NUMREC
COMMON /IL/IL(20)
COMMON /IW/IW(20)
COMMON/ZCORNR/ZCORNR(10,4,2)
INTEGER ZCURNR
COMMON /NOZERO/NOZERO
INTEGER NOZERO
COMMON /ZTYPE/ZTYPE(10)
INTEGER ZTYPE
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /ZCON/ZCON(10)
INTEGER ZCON
IOUT=6
WRITE(IOUT,100) NUMREC
100 FORMAT(*,12)
DO 200 I=1,NUMREC
WRITE(IOUT,201) [I(I),W(I)]
201 FORMAT(*,212)
200 CONTINUE
WRITE(IOUT,300) NOZERO
300 FORMAT(*,12)
DO 310 I=1,NOZERO
WRITE(IOUT,320) ZTYPE(I),ZBOUND(I),ZSIDE(I),ZREC(I),ZCON(I)
320 FORMAT(*,512)
DO 330 J=1,4
WRITE(IOUT,340) ZCORNR(I,J),ZCORRI(I,J)
340 FORMAT(*,212)
330 CONTINUE
310 CONTINUE
WRITE(IOUT,400) LENGTH,WIDTH
400 FORMAT(*,212)
DO 410 I=1,WIDTH
WRITE(IOUT,420) (XI(I),J=1,LENGTH)
420 FORMAT(*,6011)
410 CONTINUE
RETURN
END
//GO.SYSIN DD *
2
8 1
8 1
1
1
1
1
4 0
1
4 5 4 2 1
2 2
/*
*/
OPTUM

1. // JOB
2. // TIME=(3,30) REGION=512K
3. /*JOBPARM LINES=29700
4. /*JOBPARM V=0
5. // EXEC FTG1CG,TIME=(3,20)
6. // SYSPRINT DD DUMMY
7. //FORT.SYSIN DD *
8. COMMON /BENINF/BENINF.
9. INTEGER * 2 BENINF
10. COMMON /PCONT/PCONT(50)
11. INTEGER * 2 PCONT
12. BENINF=0
12.1 WRITE(6,1)
12.2 1 FORMAT(*1*,/'EIGEN LOW HIGH SHRINK GRAD ZHIGH')
13. CALL INPUT
14. CALL START
15. CALL COMBO
16. CALL TEQUAN
17. CALL RONBEN
18. CALL FIRST
19. IF(BENINF.EQ.1) GO TO 100
20. CALL SEND
21. IF(PCONT(30).EQ.1) CALL OUTFIN
22. GO TO 200
23. 100 CALL OUBINF
24. 200 STOP
25. END
26. SUBROUTINE OUBINF
27. WRITE(6,1)
28. 1 FORMAT(*1*,/'THE PROBLEM IS COMPLETELY INFEASIBLE, BECAUSE OF THIS
29. STOP THE TOTAL OPTIMIZATION')
30. RETURN
31. END
32. SUBROUTINE DUAL
33. COMMON /DLFLAG/OLFLAG
34. INTEGER * 2 DLFLAG
35. COMMON /PCONT/PCONT(50)
36. INTEGER * 2 PCONT
37. COMMON /DFUNC2/DFUNC2
38. COMMON /OBJ/OBJ
39. CALL DULSAV
40. DLFLAG=0
41. 10 CALL OLIDEN
42. CALL OSETUP
43. CALL ORSET
44. CALL INVERS
45. CALL ORESL
46. CALL OVARAJ
47. CALL OCHECK
48. CALL OACTIVE
49. IF(DLFLAG.EQ.1) GO TO 10
50. CALL OBJVAL
51. DFUNC2=OBJ
52. RETURN
53. END
54. SUBROUTINE DULSAV
55. COMMON /WID/WID(10)
56. COMMON /HEIGHT/HEIGHT(10)
57. COMMON /POS/POS(10)
COMMON /NOZERO/NOZERO
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
COMMON /DULWDS/DULWDS(10)
COMMON /DULHTS/DULHTS(10)
COMMON /DULPSS/DULPSS(10)
DO 10 I=1,10
DULWDS(I)=0.0
DULHTS(I)=0.0
DULPSS(I)=0.0
10 CONTINUE
DO 20 I=1,NOZERO
IF(ZVAL(I).EQ.0) GO TO 20
DULWDS(I)=WID(I)
DULHTS(I)=HEIGHT(I)
DULPSS(I)=POS(I)
20 CONTINUE
RETURN
END
SUBROUTINE DULGET
COMMON /WID/WID(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /POS/POS(10)
COMMON /NOZERO/NOZERO
COMMON /DULWDS/DULWDS(10)
COMMON /DULHTS/DULHTS(10)
COMMON /DULPSS/DULPSS(10)
DO 10 I=1,NOZERO
WID(I)=DULWDS(I)
HEIGHT(I)=DULHTS(I)
POS(I)=DULPSS(I)
10 CONTINUE
RETURN
END
SUBROUTINE DULIDEN
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /RECWID/RECWID(20)
INTEGER RECWID
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /PIDEN/PIDEN(10)
INTEGER * 2 PIDEN
COMMON /HIDEN/HIDEN(10)
INTEGER * 2 HIDEN
COMMON /WIDEN/WIDEN(10)
INTEGER * 2 WIDEN
COMMON /NOZERO/NOZERO
COMMON /DFPGSW/DFPGSW(10)
COMMON /DFPSSH/DFPSSH(10)
COMMON /DFPSP/DFPSP(10)
COMMON /HGRAD/WGRAD(10),/P3RA0/PGRAD(10),/DULCHK/DULCHK(10,3)
INTEGER * 2 DULCHK
COMMON /OLFLAG/OLFLAG
INTEGER * 2 DLFLAG
COMMON /PCGNT/PCQNT(10)
INTEGER * 2 PCGNT
COMMON /WGRAD/WGRAD(10),/FCHKW(10),/FCHKP(10)
INTEGER * 2 FCHKW,FCHKH,FCHKP
DO 1=1,10
WGRAD(I)=0.0
HGRAO(I)=0.0
PGRAD(I)=0.0
CONTINUE
DO 3=1,NOZERO
IF(ZVAL(I)).EQ.5) GO TO 3
WGRAD(I)=DFPGRS(I)
HGRAO(I)=DFPGRS(I)
PGRAD(I)=DFPGRS(I)
CONTINUE
DO 10=1,NOZERO
IF(ZVAL(I)).EQ.0) GO TO 10
IF(FCHKW(I)).EQ.1) WIDEN(I)=0
IF(FCHKH(I)).EQ.1) HIDEN(I)=0
IF(FCHKP(I)).EQ.1) PIDEN(I)=0
WRITE(6,76) I,WIDEN(I),HIDEN(I),PIDEN(I),XBOUND(I),HEIGHT(I),PCGNT(I),
XBOUND,CHECK
CONTINUE
DO 10=1,NOZERO
IF(ZVAL(I)).EQ.0) GO TO 10
XBOUND=RECLEN(REC(I))
IF(ZSIDE(I)).EQ.1.OR.ZSIDE(I)).EQ.4) XBOUND=RECLEN(REC(I))
IF(WID(I)).EQ.XBOUND) WIDEN(I)=0
IF(WID(I)).LT.XBOUND) CHECK=POS(I)+WID(I)
IF(CHECK).EQ.XBOUND) PIDEN(I)=0
IF(HEIGHT(I)).EQ.ZBOUND(I)) HIDEN(I)=0
WRITE(6,76) I,WIDEN(I),HIDEN(I),PIDEN(I),XBOUND(I),HEIGHT(I),PCGNT(I),
XBOUND,CHECK
CONTINUE
IF(OLFLAG).EQ.0) RETURN
CONTINUE
IF(DULCHK(I,1)).EQ.1) WIDEN(I)=1
IF(DULCHK(I,2)).EQ.1) HIDEN(I)=1
IF(DULCHK(I,3)).EQ.1) PIDEN(I)=1
CONTINUE
DO 10=1,NOZERO
IF(ZVAL(I)).EQ.0) GO TO 10
IF(FCHKW(I)).EQ.1) WIDEN(I)=0
IF(FCHKH(I)).EQ.1) HIDEN(I)=0
IF(FCHKP(I)).EQ.1) PIDEN(I)=0
CALL ODLVAR
RETURN
274

170. END
171. SUBROUTINE OSETUP
172. COMMON /PIDEN/PIDEN(10)
173. INTEGER*2 PIDEN
174. COMMON /WIDEN/WIDEN(10)
175. INTEGER*2 WIDEN
176. COMMON /HIDEN/HIDEN(10)
177. INTEGER*2 HIDEN
178. COMMON /ZVAL/ZVAL(10)
179. INTEGER*2 ZVAL
180. COMMON /MID/MID(10)
181. COMMON /HEIGHT/HEIGHT(10)
182. COMMON /POS/POS(10)
183. COMMON /NOZERO/NOZERO
184. COMMON /COEFW/COEFW(10)
185. COMMON /COEFPH/COEFPH(10)
186. COMMON /COEFWH/COEFWH(10)
187. COMMON /COEFWH2/COEFWH2(10)
188. COMMON /COEFH2/COEFH2(10)
189. COMMON /COEFH2/COEFH2(10)
190. COMMON /COEFH2/COEFH2(10)
191. COMMON /COEFH2/COEFH2(10)
192. COMMON /COEFPH/COEFPH(10)
193. COMMON /A/A(30,30)
194. COMMON /NROW/NROW
195. COMMON /WGRAD/WGRAD(10)
196. COMMON /HGRAD/HGRAD(10)
197. COMMON /PGRAO/PGRAO(10)
198. NROW=1
199. DO 10 I=1,NOZERO
200. IF(WIDEN(I).EQ.1) NROW=NROW+1
201. IF(PIDEN(I).EQ.1) NROW=NROW+1
202. IF(HIDEN(I).EQ.1) NROW=NROW+1
203. 10 CONTINUE
204. DO 20 I=1,30
205. DO 21 J=1,30
206. A(I,J)=0.0
207. 21 CONTINUE
208. 20 CONTINUE
209. ICT=0
210. DO 100 I=1,NOZERO
211. IF(ZVAL(I).EQ.0) GO TO 100
212. IF(WIDEN(I).EQ.0) GO TO 100
213. ICT=ICT+1
214. A(ICT,ICT)=2.0*COEFW2(I)
215. 100 CONTINUE
216. DO 110 I=1,NOZERO
217. IF(ZVAL(I).EQ.0) GO TO 110
218. IF(HIDEN(I).EQ.0) GO TO 110
219. ICT=ICT+1
220. A(ICT,ICT)=2.0*COEFH2(I)
221. 110 CONTINUE
222. DO 120 I=1,NOZERO
223. IF(ZVAL(I).EQ.0) GO TO 120
224. IF(PIDEN(I).EQ.0) GO TO 120
225. ICT=ICT+1
226. A(ICT,ICT)=2.0*COEFPH2(I)
227. 120 CONTINUE
228. ICT=0
229. DO 200 I=1,NOZERO
230. IF(ZVAL(I).EQ.0) GO TO 200
231. IF(WIDEN(I).EQ.0) GO TO 200
232. ICT=ICT+1
233. A(ICT,NROW)=WGRAD(I)
234. A(NROW,ICT)=1.0*WGRAD(I)
235. 200 CONTINUE
236. 00 210 I=1,NOZERO
237. IF(ZVAL(I).EQ.0) GO TO 210
238. IF(WIDEN(I).EQ.0) GO TO 210
239. ICT=ICT+1
240. A(ICT,NROW)=HGRAD(I)
241. A(NROW,ICT)=1.0*HGRAD(I)
242. 210 CONTINUE
243. 00 220 I=1,NOZERO
244. IF(ZVAL(I).EQ.0) GO TO 220
245. IF(WIDEN(I).EQ.0) GO TO 220
246. ICT=ICT+1
247. A(ICT,NROW)=PGRAD(I)
248. A(NROW,ICT)=1.0*PGRAD(I)
249. 220 CONTINUE
250. 00 100 I=1,NOZERO
251. WRITE(6,78) WGRAD(I),HGRAD(I),PGRAD(I)
252. 78 FORMAT(10X,'GR ADI ENTS W H P ',3F10.8,10X,13)
253. 1000 CONTINUE
254. RETURN
255. END
256. SUBROUTINE DRHSET
257. COMMON /NOZERO/NOZERO
258. COMMON /ZVAL/ZVAL(10)
259. INTEGER ZVAL
260. COMMON /WID/WID(10)
261. COMMON /POS/POS(10)
262. COMMON /HEIGHT/HEIGHT(10)
263. COMMON /COEF/COEF(10)
264. COMMON /COEF2/COEF2(10)
265. COMMON /PCONT/PCONT(50)
266. INTEGER PCONT
267. COMMON /DSHK/V/DSHKVL
268. COMMON /SHKTOL/SHKTOL
269. CALL TEQVAL
270. ICT=0
271. 00 100 I=1,NOZERO
272. IF(ZVAL(I).EQ.0) GO TO 100
273. IF(WIDEN(I).EQ.0) GO TO 100
274. ICT=ICT+1
Ui Ui Ui
9 > 0 ' O ' O ' O ' U l U l U l t f l i n U ) U ) U i t n t n l ‘ « F ' £ « ' f > ^ . f « £ ' 4 ' ' f * £ ' U i U i U i U J U i u ru ru
*0 00
• •

£ ' U i f U f t - O ' 6 O D - 4 0 ' U l ‘f * U i r u r - > O ' O a > * 4 0 ' i n 4 ' U i t u ^ " O « 6 O O - > 4 O » u l O
l « l l ( l l l l l l l » l l l l l l l l l ( l l « l l l l
•

U i

ru

u
O
o
2
•H
2
C
m

O
O

o
C O
2 2
U> II H
O o i-^
2
*m O C
II
.n
e—

mm

J—

ft

••

2

—*
•ft
in
-ft
O
70
m
•**
mm

in

—*

Ui
M
0*
•

Ui
M
tn
i

Ui Ui
fu ru
-f' Ui
• 1

Ui
ru
N
•

Ui
ru
h•

W
IU
o
•

Ui Ui U)
p* r* »—
s0 00 -4
• i •

Ui Ui Ui Ui
1— h- »— p»
0* Ul •f' Ui
• •
•

ft

Ui Ui Ui Ui Ui Ui U) Ui Ui Ui Ui Ui
i— ►— I— o o O o o o o o o
fu
o vO 00 *4 O' UI
Ui ru h• • 1
• •
ft •

ft

o

m
H
O
33
m
—

Ci o
O O
2
fu
C

>

u
□

M
h

II
2
»
»■*
O
^ c «• •-« r.
*-» u m »~ it 1
r—
—*
II
> *•
II
mm
r- M
H pm
• H*
••
O
2
<

2

—
>
*-H
-

3>
*—
•—<
*•
*-*
*—

H
o

O in
□ rn
2 *H
—1
2
C
m

to m
□ H
-1 o
— 1 70
rnm o m
w 2 mrn
II •_» mm
c mm w
• *— II
o II O
o •
* o
c

o
D
2
H
M
2
C
m

>

2
u
2
ru »“ 73
C
M Cm
5.
II II
c- *- »—
W
II Ui Ui
O O o
•
o
2
<

a
o

O
O

O
mm

z
m
z
in

M
o
2
in
-ft
n
73
m
M
Ui
O
w

n
2 o
-ft z
m z
o o
m 2
70
* 70
O
(U O
2
"O -ft
n s*
o 73
2 n
— 1 CD

Ui

o

in
m

z
2
O
2

2
2
O
2

2

2

2

mm

03 C
70
O
c
-ft
mm

"«*
Z
3C
CD
£
\

J> >
M >*
z >
<
U)
> O
<*
73
O
Ui
£ < o
mm w
Ui
-ft

& UI
o wo
H
-ft
a

n n o m m 73
o o a X X X X X mm mm mm o o a X X X X X mm
o a n c 2 m •n t o a 73 II II II 11 II n 71 71 o o 70 u 11 II U II n 71
z
2
O
2

o

w
Ui
o
w

z
m
mm

2
<
m
73
in

-1 *■* X
c 73 in
o «
X
2 o 2
2 70
-M O

*-4

M
o
H

mm X

ui
CO

T

ft T
rn

CO •
ft o
ftw *
o
>
r

in
X

r- H
O
or
a I

f
o
>
m
2 X
-i X

X o M
II
o o *■*
n o ru a
n
a a • m
H
m m c 71
+
r-»
*n n # 73
s : 73 r> rnm
*D X o
m w
M H 71

2 X X X X
-h tn * ♦ ♦

2
C
m

ft ft ft

ft

o
o

fu

o

tn
c
Z
II
in
C
*J
+
*—
>

UI
ru
-4
•

II
1

h-

ft
o
*0
* *
ru
X £ X mm
m mm
O i-i mm
a #
Nh X *0
w h o
mm in

mm mm
mm mm
w

mm

ft

2
73 r>j m— - l
< fU r-«
s>
D
o 2
m r
C
mm m
2
II
mm •w
m—

•

*

ft m 2
m o
o • M
• o m
o mm 73

O

w

a
Cl
a

a

*H

•H o

O

hH* ru

fuo
o

O

X X X X X c
in + f + + ft
rnm mm
n o
mm o o ru o
n a o * ni
- i m m o 71
*«• “n 71 * X
n 73 £ o m
i X X D mm
•-» m mm
r• M M 71
o mm w> X
* « * ru
X ■u £ mm
mm
o
in O w
*
M X
W m
W *»•» t-i
o
X
H

0

o n
o o
H <x« «■»
2
ii X N» e - - ft
mm
<
o 2
H m r*
C
mm m
♦ 2
mm II
rhW
«
ft m 2
m o O
o ft PM
• o m
o w 73
o
o
o o
o
—1
H o
a

71

aa

ft

H K-

oo

a
30
X
in

X
II
X
+

o
H
w
II

n
o
m
71
£
*o

0m

1

X X X X
II II II ll
X X X o
+ + +
n o
o ru o
a • m
m o 71
71 # £
£ o
X c mm
0m, m W
K*i 7 i
M £
# fu
X
m 0*,

ft

1—
ft »-4
O w
ftp *
X "O
o
in N mm
o *
M X £
W -ft
w 0m O

mm

w

W

<

H

to
'J
CTl


DO 40 I=1,NROW
  BOTTOM(I)=AI(I,NROW)*AINV(I,I)
40  CONTINUE
DO 50 I=1,NROW
  BOTTOM(I)=BOTTOM(I)/SUM
50  CONTINUE
BOTTOM(NROW)=1.0/SUM
DO 60 J=1,NROW
  AINV(NROW,J)=BOTTOM(J)
60  CONTINUE
DO 70 I=1,NROW
  AINV(NROW,I)=BOTTOM(I)*STORE(I)*(-1.0)
70  CONTINUE
DO 100 J=1,NROW
  SET(J)=BOTTOM(J)*STORE(I)*(-1.0)
100 CONTINUE
IF(PCONT(39).EQ.1) CALL OOLINV
RETURN
END
SUBROUTINE DRESLT
COMMON /AINV/AINV(30,30)
COMMON /DRHS/DRHS(30)
COMMON /ORES/DRES(30)
COMMON /NROW/NROW
COMMON /DSHRNK/DSHRNK
DO 100 I=1,30
  ORES(I)=0.0
100 CONTINUE
DO 100 I=1,NROW
  ORES(I)=ORES(I)+(AINV(I,J)*DRHS(J))
100 CONTINUE
DSHRNK=DRES(NROW)
WRITE(6,301) I,DRES(1)
301 FORMAT(' INVERSE CHANGE TO EACH VARIABLE - RESULT =',I2,2X,F15.12)
RETURN
END
SUBROUTINE OVARAJ
COMMON /NROU/NROW
COMMON /ORES/ORES(30)
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
COMMON /WIOEN/WIOEN(10)
INTEGER * 2 WIOEN
COMMON /PIDEN/PIDEN(10)
COMMON /HIDEN/HIDEN(10)
COMMON /PGS/PGS(10)
COMMON /NOZERO/NOZERO
COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT
ICT=0
DO 100 I=1,NOZERO
IF(ZVAL(I).EQ.0) GO TO 1G0
IF(WIDED(I).EQ.0) GO TO 100
ICT=ICT+1
WID(I)=WID(I)+DRES(ICT)
CONTINUE
100 DO 110 I=1,NOZERO
IF(ZVAL(I).EQ.0) GO TO 110
IF(WIDED(I).EQ.0) GO TO 110
ICT=ICT+1
HEIGHT(I)=HEIGHT(I)+DRES(ICT)
CONTINUE
110 DO 120 I=1,NOZERO
IF(ZVAL(I).EQ.0) GO TO 120
IF(WIDED(I).EQ.0) GO TO 120
ICT=ICT+1
POS(I)=POS(I)+DRES(ICT)
CONTINUE
120 DO 200 I=1,NOZERO
IF(ZVAL(I).LT.0.0) WID(I)=0.0
IF(HEIGHT(I).LT.0.0) HEIGHT(I)=0.0
IF(POS(I).EQ.0.0) POS(I)=0.0
CONTINUE
200 DO 300 I=1,NOZERO
WRITE(6,301) WID(I),HEIGHT(I),POS(I)
301 FORMAT(9,REVISED VALUES #='*,F15.6,IX,'H='*,F15.6,'P='*,F15.
CONTINUE
300 IF(PCONT(40).EQ.1) CALL ODLRES
RETURN
END
SUBROUTINE DACTVE
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
COMMON /NOZERO/NOZERO
COMMON /OHGT/DHGT(10)
COMMON /OPOS/DPCS(10)
COMMON /LHSW/LHSW(10)
REAL LHS#
COMMON /LHS/LHS(10)
REAL LHP
COMMON /LHP/LHSP(10)
REAL LHP
COMMON /LHP/LHSP(10)
REAL LHP
COMMON /LHP/LHSP(10)
REAL LHP
COMMON /LHP/LHSP(10)
REAL LHP
COMMON /LHP/LHSP(10)
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /RECWID/RECWID(20)
INTEGER RECWID
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /WGRAD/WGRAD(10)
COMMON /HGRAD/HGRAD(10)
COMMON /PGRAD/PGRAD(10)
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /DULCHK/DULCHK(10,3)
INTEGER * 2 DULCHK
COMMON /DULTOL/DULTOL
COMMON /DLFLAG/DLFLAG
INTEGER *2 DLFLAG
COMMON /PCONT/PCONT(50)
INTEGER *2 PCONT
DO 10 I=1,10
DEXIS(I)=0.0
Dwidbo(I)=0.0
DHgtbo(I)=0.0
Dwidps(I)=0.0
Dwid(I)=0.0
DO 100 I = 1, NOZERO
   LHSW(I) = COEFW(I) * (2.0 * COEFW2(I) * WID(I))
   LHSW(I) = LHSW(I) + (COEFWH(I) * HEIGHT(I))
   LHSW(I) = LHSW(I) + (COEFWP(I) * POS(I))
   LHSW(I) = LHSW(I) + (COEFPH(I) * WID(I))
100 CONTINUE
DO 129 I = 1, NOZERO
   XBD = RECWID(ZREC(I))
   IF(ZSIDE(I).EQ.1 OR ZSIDE(I).EQ.4) XBD = RECLEN(ZREC(I))
   IF(WID(I).EQ.XBD) GO TO 130
   CHK = WID(I) + POS(I)
   IF(CHK .EQ. XBD) GO TO 130
   GO TO 155
129 CONTINUE
DO 121 I = 1, NOZERO
   XBND = RECWID(ZREC(I))
   IF(ZSIDE(I).EQ.1 OR ZSIDE(I).EQ.4) XBND = RECLEN(ZREC(I))
   IF(WID(I).EQ.XBD) GO TO 121
   OSHRNK = LHSW(I) - LHSP(I)
   OSHRNK = DSHRNK / DNOM
   GO TO 155
121 CONTINUE
DO 120 I = 1, NOZERO
   XBND = RECWID(ZREC(I))
   IF(ZSIDE(I).EQ.1 OR ZSIDE(I).EQ.4) XBND = RECLEN(ZREC(I))
   IF(WID(I).EQ.XBD) GO TO 120
   OSHRNK = LHSW(I) - LHSP(I)
   OSHRNK = DSHRNK / DNOM
   GO TO 155
120 CONTINUE
DO 140 I = 1, NOZERO
   XBND = RECWID(ZREC(I))
   IF(ZSIDE(I).EQ.1 OR ZSIDE(I).EQ.4) XBND = RECLEN(ZREC(I))
   IF(WID(I).EQ.XBD) GO TO 140
   OSHRNK = LHSW(I) - LHSP(I)
   OSHRNK = DSHRNK / DNOM
   GO TO 155
140 CONTINUE
DO 200 I = 1, NOZERO
   XBND = RECWID(ZREC(I))
   IF(ZSIDE(I).EQ.1 OR ZSIDE(I).EQ.4) XBND = RECLEN(ZREC(I))
   IF(WID(I).EQ.XBD) GO TO 200
   OSHRNK = LHSW(I) - LHSP(I)
   OSHRNK = DSHRNK / DNOM
   GO TO 200
200 CONTINUE
IF(LHSW(I).GT.XBD) DWID(I) = LHSW(I)
IF(LHSP(I).GT.XBD) DWIDPS(I) = -1.0 * LHSP(I)
DO 235 I = 1, NOZERO
   IF(DWIDPS(I).GT.0.0) DWIDPS(I) = -1.0 * LHSP(I)
   IF((LHSW(I).GT.XBD) OR (LHSP(I).GT.XBD)) DWID(I) = -1.0 * LHSW(I)
   IF((LHSW(I).LT.XBD) OR (LHSP(I).LT.XBD)) DWID(I) = -1.0 * LHSW(I)
235 CONTINUE
281

560.  DDWID(I)=LHSW(I)+DWIDPS(I)
561.  GO TO 200
562.  240 IF(WID(I).EQ.XBND) GO TO 245
563.  IF(LHS(I).GE.0.0) DPOS(I)=LHS(I)
564.  GO TO 200
565.  245 IF(LHS(I).GT.0.0) GO TO 247
566.  DWIDPS(I)=-1.0*LHS(I)
567.  IF(LHSW(I).LT.0.0.AND.LHSW(I).LT.LHSP(I)) DWIDBD(I)=-1.0*(LHSW(I)+
568.  XDWIDPS(I))
569.  GO TO 200
570.  247 DPOS(I)=LHSW(I)
571.  IF(LHSW(I).LT.0.0) DWIDBD(I)=-1.0*(LHSW(I)+
572.  XDWIDPS(I))
573.  GO TO 200
574.  250 CHECK=WID(I)+POS(I)
575.  IF(CHECK.EQ.XBND) GO TO 255
576.  GO TO 200
577.  255 DWIDPS(I)=-1.0*LHSW(I)
578.  CONTINUE
579.  DO 300 I=1,NOZERO
580.  XBND=ZBOUND(I)
581.  IF(ZVAL(I).EQ.0) GO TO 300
582.  IF(HEIGHT(I).EQ.0.0) GO TO 310
583.  IF(HEIGHT(I).GT.XBND) GO TO 300
584.  GO TO 300
585.  310 IF(LHSH(I).GT.0.0) DHGT(I)=LHSH(I)
586.  GO TO 300
587.  330 IF(LHSH(I).LT.0.0) DHGTBD(I)=-1.0*LHSH(I)
588.  CONTINUE
589.  DO 400 I=1,NOZERO
590.  IF(ZVAL(I).EQ.1) GO TO 400
591.  ID=1
592.  CHECK=LHSW(I)
593.  IF(LHSH(I).LT.CHECK) CHECK=LHSH(I)
594.  IF(LHSH(I).LT.CHECK) ID=2
595.  IF(LHSP(I).LT.CHECK) CHECK=LHSP(I)
596.  IF(LHSP(I).LT.CHECK) ID=3
597.  IF(CHECK.GT.0.0) GO TO 420
598.  DEXIS(I)=-1.0*CHECK
599.  420 WIDCK=LHSH(I)+DEXIS(I)
600.  HGTCK=LHSH(I)+DEXIS(I)
601.  XPOSCK=LHSP(I)+DEXIS(I)
602.  IF(WIDCK.GT.0.0) DWD(I)=WIDCK
603.  IF(HGTCK.GT.0.0) DHGT(I)=HGTCK
604.  IF(XPOSCK.GT.0.0) DPOS(I)=XPOSCK
605.  CONTINUE
606.  DO 480 J=1,10
607.  DO 481 K=1,3
608.  DULCHK(I,J,K)=0
609.  481 CONTINUE
610.  480 CONTINUE
611.  DO 500 I=1,NOZERO
612.  XH=LHSH(I)+DEXIS(I)+DWIDBD(I)+DWIDPS(I)-WID(I)
613.  XH=LHSH(I)+DEXIS(I)+DHGTBD(I)-DHGT(I)
614.  XH=LHSP(I)+DEXIS(I)+DWIDPS(I)-DPOS(I)
615.  WRITE(16,501) I,XH,XH,XP
616.  501 FORMAT(1X,'DUAL CONDITION TEST VAR*','I3',' WID HGT PCS*','I3','XW,XP')
617.  WRITE(16,502) DEXIS(I)
618.  502 FORMAT(1X,'DEXIS(I) = ',F15.8)
619.  WRITE(16,503) DWIDBD(I),DWIDPS(I)
**SUBROUTINE DCHECK**

- **COMMON** /NOZERO/NOZERO
- **COMMON** /WID/WID(10)
- **COMMON** /HEIGHT/HEIGHT(10)
- **COMMON** /POS/POS(10)
- **COMMON** /ZVAL/ZVAL(10)
- **INTEGER** * 2 ZVAL
- **INTEGER** ZBOUND
- **INTEGER** ZBOUND
- **INTEGER** /ZREC/ZREC(10)
- **INTEGER** ZREC
- **INTEGER** /RECLEN/RECLEN(20)
- **INTEGER** RECLEN
- **INTEGER** /RECWD/RECWD(20)
- **INTEGER** RECWD
- **COMMON** /ZSIDE/ZSIDE(110)
- **INTEGER** ZSIDE
- **COMMON** /DFLAG/DFLAG
- **INTEGER** DFLAG
- **COMMON** /DULDS/DULDS(10)
- **COMMON** /DULHTS/DULHTS(10)
- **COMMON** /DULPSS/DULPSS(10)
- **COMMON** /FINAL/FCHKW(10),FCHKH(10),FCHKP(10)

- **DO** 10 **I** = 1,NOZERO
- **IF** (ZVAL(I).EQ.0) **GO TO** 10
- **XBOUND** = RECWID(ZREC(I))

**1 CONTINUE**

- **DO** 10 **I** = 1,NOZERO
- **IF** (ZVAL(I).EQ.0) **GO TO** 10
- **XBOUND** = RECWID(ZREC(I))

**10 CONTINUE**

**RETURN**
283

10 CONTINUE
676. DO 20 I=1,NOZERO
677. IF(ZVAL(I).EQ.0) GO TO 20
678. XBOUND=RECHIDZREC(I)
679. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
680. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
681. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
682. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
683. CONTINUE
684. DO 30 I=1,NOZERO
685. IF(ZVAL(I).EQ.0) GO TO 30
686. XBOUND=RECHIDZREC(I)
687. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
688. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
689. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
690. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
691. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
692. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
693. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
694. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
695. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
696. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
697. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
698. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
699. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
700. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
701. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
702. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
703. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
704. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
705. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
706. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XBOUND=RECLNZREC(I)
707. RETURN
708. END

SUBROUTINE CDLVAP.

COMMON /PIDEN/PIDEN(10)
INTEGER * 2 PIDEN
COMMON /HIDEN/HIDEN(10)
INTEGER * 2 HIDEN
COMMON /WIDEN/WIDEN(10)
INTEGER * 2 WIDEN
COMMON /NOZERO/NOZERO
COMMON /ZVAL/ZVAL(10)
COMMON /ZBOUND/ZBOUND(10)
INTEGER * 2 ZVAL
COMMON /TYP/TYP(10,6)
LOGICAL * 1 TYP
RITE(6,1)

1 FORMAT(*"1",47X,"DUAL VARIABLE CONSIDERATION RESULTS'")
2 FORMAT(*"1",47X,"DUAL VARIABLE CONSIDERATION RESULTS'")
WRITE(6,3) 3 FORMAT('--',20X,'VARIABLE RESULTS') WRITE(6,4)
 DO 10 I=1,NOZERO IF(ZVAL(I).EQ.0) GO TO 20 WRITE(6,11) I,(TYP(I,J),J=1,6)
 11 FORMAT(' ',30X,'VARIABLE RESULTS') IF(WIDEN(I).EQ.1) WRITE(6,12)
 IF(HIDEN(I).EQ.1) WRITE(6,13)
 IF(PILEN(I).EQ.1) WRITE(6,14)
 12 FORMAT(' ',35X,'WIDTH CONTINUOUS VARIABLE IS INCLUDED') 13 FORMAT(' ',35X,'HEIGHT CONTINUOUS VARIABLE IS INCLUDED')
 14 FORMAT(' ',35X,'POSITION CONTINUOUS VARIABLE IS INCLUDED') WRITE(6,4)
 GO TO 10 WRITE(6,11) I,(TYP(I,J),J=1,6)
 WRITE(6,15)
 15 FORMAT(' ,35X,'DISCRETE VARIABLE IS NOT INCLUDED') 10 CONTINUE RETURN

END

SUBROUTINE GDLMAT
COMMON /NROW/NROW COMMON /A/A(30,30)
COMMON /W/DEN/WIDEN(10)
INTEGER * 2 WIDEN
COMMON /HIDEN/HIDEN(10)
INTEGER * 2 HIDEN
COMMON /P/I DEN/PIDEN(10)
INTEGER * 2 PIDEN
COMMON /DRHS/DRHS(10)
COMMON /NOZERO/NOZERO
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
WRITE(6,1) 1 FORMAT('-',59X,'DUAL MATRIX')
IF(NROW.LE.10) (P=NROW
IF(NROW.GT.10) P=10
WRITE(6,2)

2 FORMAT(' ', ' ') ICT=0
 DO 10 I=1,NOZERO IF(ZVAL(I).EQ.0) GO TO 10
 IF(WIDEN(I).EQ.0) GO TO 10 ICT=ICT+1
 WRITE(6,11) I,(A(ICT,J),J=1,P)
 11 FORMAT(' ',WIDTH',3X,12,**,2X,10E11.4)
 IF(NROW.LE.10) GO TO 13
 WRITE(6,12) (A(ICT,J),J=11,NROW)
 12 FORMAT(' ',8X,2X,1X,2X,10E11.4)
 ICT=ICT+1
 WRITE(6,15) ORHS(ICT)
 15 FORMAT(' ',10X,'RIGHT HAND SIDE = ',E15.7)
 WRITE(6,2)
 10 CONTINUE
 DO 20 I=1,NOZERO IF(ZVAL(I).EQ.0) GO TO 20
 ICT=ICT+1
 WRITE(6,21) I,(A(ICT,J),J=1,P)
785. 21 FORMAT(••, 'HEIGHT', 2X, 12••••, 2X, 10E11.4)
786. IF(NROW.LE.10) GO TO 25
787. WRITE(6,22) (A(ICT,J), J=1, NROW)
788. 22 FORMAT(••, 8X, 2X, 1X, 2X, 10E11.4)
789. 25 WRITE(6,15) DRHS(I CT)
790. WRITE(6,2)
791. 20 CONTINUE
792. DO 30 I=1, NOZERO
793. IF(ZVAL(I).EQ.0) GO TO 30
794. IF(PIDEN(I).EQ.0) GO TO 30
795. ICT=ICT+1
796. WRITE(6,31) I, (A(I CT,J), J=1, IP)
797. 31 FORMAT(••, 'POSITION', 12••••, 2X, 10E11.4)
798. IF(NROW.LE.10) GO TO 35
799. WRITE(6,32) (A(I CT,J), J=11, NROW)
800. 32 FORMAT(••, 8X, 2X, 1X, 2X, 10E11.4)
801. 35 WRITE(6,15) DRHS(I CT)
802. WRITE(6,2)
803. 30 CONTINUE
804. WRITE(6,40) (A(NROW,J), J=1, IP)
805. 40 FORMAT(••, 'SHRINKAGE', 12••••, 2X, 10E11.4)
806. IF(NROW.LE.10) GO TO 45
807. WRITE(6,41) (A(NROW,J), J=11, NROW)
808. 41 FORMAT(••, 8X, 2X, 1X, 2X, 10E11.4)
809. 45 WRITE(6,15) DRHS(NROW)
810. RETURN
811. END
812. SUBROUTINE ODLINV
813. COMMON /NRQW/NROW
814. COMMON /AINV/AINV(30, 30)
815. COMMON /WIDEN/WIDEN(10)
816. INTEGER *2 WIDEN
817. COMMON /AHDEN/AHDEN(10)
818. INTEGER *2 AHDE N
819. COMMON /PIDEN/PIDEN(10)
820. INTEGER *2 PIDEN
821. COMMON /DRHS/DRHS(10)
822. COMMON /NOZERO/NOZERO
823. COMMON /ZVAL/ZVAL(10)
824. INTEGER *2 ZVAL
825. WRITE(6,1)
826. 1 FORMAT(••••, 5X, 'INVERSE MATRIX')
827. IF(NROW.LE.10) IP=NROW
828. IF(NROW.GT.10) IP=10
829. WRITE(6,2)
830. 2 FORMAT(••••, 1)
831. ICT=0
832. DO 10 I=1, NOZERO
833. IF(ZVAL(I).EQ.0) GO TO 10
834. IF(WIDEN(I).EQ.0) GO TO 10
835. ICT=ICT+1
836. WRITE(6,11) I, (AINV(I CT,J), J=1, IP)
837. 11 FORMAT(••, 'WIDTH', 3X, 12••••, 2X, 10E11.4)
838. IF(NROW.LE.10) GO TO 13
839. WRITE(6,12) (AINV(I CT,J), J=11, NROW)
840. 12 FORMAT(••, 8X, 2X, 1X, 2X, 10E11.4)
841. 13 WRITE(6,2)
842. 10 CONTINUE
843. DO 20 I=1, NOZERO
844. IF(ZVAL(I).EQ.0) GO TO 20
IF(WIDEN(I)).EQ.0) GO TO 20
ICT=ICT+1
WRITE(6,21) I,(AINV(ICT,J),J=1,IP)
21 FORMAT(' ', 'HEIGHT', '2X','I2','*','2X','10E11.4')
IF(NROW.LE.10) GO TO 25
WRITE(6,22) (AINV(ICT,J),J=11,NROW)
22 FORMAT(' ', 'SHRINKAGE', '2X','I2','*','2X','10E11.4')
IF(NROW.LE.10) GO TO 25
WRITE(6,2)
20 CONTINUE
DO 30 I=1,NOZERO
IF(ZVAL(I)).EQ.0) GO TO 30
IF(PIDEN(I)).EQ.0) GO TO 30
ICT=ICT+1
WRITE(6,31) I,(AINV(ICT,J),J=1,IP)
31 FORMAT(' ', 'POSITION', '2X','I2','*','2X','10E11.4')
IF(NROW.LE.10) GO TO 35
WRITE(6,32) (AINV(NROW,J),J=11,NROW)
32 FORMAT(' ', 'SHRINKAGE', '2X','I2','*','2X','10E11.4')
30 CONTINUE
WRITE(6,40) (AINV(NROW,J),J=11,NROW)
40 FORMAT(' ', 'SHRINKAGE', '2X','I2','*','2X','10E11.4')
WRITE(6,41) (AINV(NROW,J),J=11,NROW)
41 FORMAT(' ', 'SHRINKAGE', '2X','I2','*','2X','10E11.4')
45 RETURN
END

SUBROUTINE ODLRES
COMMON /W10/W10(I10)
COMMON /POS/POS(I10)
COMMON /HEIGHT/HEIGHT(I10)
COMMON /ZVAL/ZVAL(I10)
INTEGER * 2 ZVAL
COMMON /WIDEN/WIDEN(I10)
COMMON /HIDEN/HIDEN(I10)
INTEGER * 2 WIDEN
COMMON /PIDEN/PIDEN(I10)
INTEGER * 2 PIDEN
COMMNon /NOZERO/NOZERO
COMMON /DULWOS/DULWOS(I10)
COMMON /DULHTS/DULHTS(I10)
COMMON /DULPSS/DULPSS(I10)
COMMON /TYP/TYP(I10,6)
LOGICAL * 1 TYP
WRITE(6,1)
1 FORMAT('-','52X','DUAL VARIABLE ADJUSTMENTS'
WRITE(6,3)
3 FORMAT('-','I')
WRITE(6,2)
2 FORMAT('-','THE FOLLOWING IS THE RESULTING CHANGE FROM THE DUAL VA
WRITE(6,31) I,(AINV(ICT,J),J=1,IP)
31 FORMAT(' ', 'POSITION', '2X','I2','*','2X','10E11.4')
IF(NROW.LE.10) GO TO 35
WRITE(6,32) (AINV(NROW,J),J=11,NROW)
32 FORMAT(' ', 'SHRINKAGE', '2X','I2','*','2X','10E11.4')
30 CONTINUE
WRITE(6,40) (AINV(NROW,J),J=11,NROW)
40 FORMAT(' ', 'SHRINKAGE', '2X','I2','*','2X','10E11.4')
WRITE(6,41) (AINV(NROW,J),J=11,NROW)
41 FORMAT(' ', 'SHRINKAGE', '2X','I2','*','2X','10E11.4')
45 RETURN
905. \[ \text{XP}=\text{DULPSS}(I)-\text{POS}(I) \]
906. \[ \text{IF(WIDEN(I)).EQ.1)} \text{WRITE}(6,12) \text{I,DULWDS}(I),\text{WID}(I),\text{XW} \]
907. \[ \text{IF(HIDEN(I)).EQ.1)} \text{WRITE}(6,13) \text{I,DULHTS}(I),\text{HEIGHT}(I),\text{XH} \]
908. \[ \text{IF(PIDEN(I)).EQ.1)} \text{WRITE}(6,14) \text{I,DULPSS}(I),\text{POS}(I),\text{XP} \]
909. \[ \text{IF(WIDEN(I)).EQ.0.AND.HIDEN(I)).EQ.1)} \text{WRITE}(6,15) \text{X} \]
910. \[ \text{WRITE}(6,3) \]
911. \[ \text{WRITE}(6,3) \]
912. \[ \text{12 FORMAT}'VARIABLE - ',(2,2X,'WIDTH'),3X,'OLD VALUE = ',E12.5,2X,'NEW VALUE = ',E12.5,2X,'DIFFERENCE = ',E12.5) \]
913. \[ \text{13 FORMAT}'VARIABLE - ',(2,2X,'HEIGHT'),2X,'OLD VALUE = ',E12.5,2X,'NEW VALUE = ',E12.5,2X,'DIFFERENCE = ',E12.5) \]
914. \[ \text{14 FORMAT}'VARIABLE - ',(2,2X,'POSITION'),2X,'OLD VALUE = ',E12.5,2X,'NEW VALUE = ',E12.5,2X,'DIFFERENCE = ',E12.5) \]
915. \[ \text{15 FORMAT}'VARIABLE - ',(2,2X,'NO CONTINUOUS VARIABLES WERE INCLUDED IN THE DUAL ADJUSTMENT') \]
916. \[ \text{10 CONTINUE} \]
917. \[ \text{RETURN} \]
918. \[ \text{END} \]
919. \[ \text{SUBROUTINE OOLCHK} \]
920. \[ \text{RETURN} \]
921. \[ \text{END} \]
922. \[ \text{SUBROUTINE SETSPG} \]
923. \[ \text{COMMON /NOZERO/NOZERO} \]
924. \[ \text{COMMON /ZVAL/ZVAL(10)} \]
925. \[ \text{INTEGER*2 ZVAL} \]
926. \[ \text{COMMON /PCONT/PCONT(50)} \]
927. \[ \text{COMMON /ZSIDE/ZSIDE(10)} \]
928. \[ \text{INTEGER*2 PCONT} \]
929. \[ \text{DO 10 I=1,NOZERO} \]
930. \[ \text{IF(ZVAL(I)).EQ.0) GO TO 10} \]
931. \[ \text{CALL SGRADP(I)} \]
932. \[ \text{CALL SGRADH(I)} \]
933. \[ \text{CALL SGRADW(I)} \]
934. \[ \text{10 CONTINUE} \]
935. \[ \text{RETURN} \]
936. \[ \text{END} \]
937. \[ \text{SUBROUTINE SGRADP(I)} \]
938. \[ \text{COMMON /ZREC/ZREC(10)} \]
939. \[ \text{COMMON /RECLEN/RECLEN(20)} \]
940. \[ \text{INTEGER*2 ZREC} \]
941. \[ \text{CALL SGRADW(I)} \]
942. \[ \text{COMMON /ZSIDE/ZSIDE(10)} \]
943. \[ \text{CALL SGRADH(I)} \]
944. \[ \text{COMMON /WID/WID(10)} \]
945. \[ \text{COMMON /POS/POS(10)} \]
946. \[ \text{COMMON /HEIGHT/HEIGHT(10)} \]
947. \[ \text{COMMON /POS/POS(10)} \]
948. \[ \text{COMMON /SHRINK/SHRINK} \]
949. \[ \text{COMMON /GRADNT/GRADNT(6,3,5)} \]
950. \[ \text{COMMON /BREAK/BREAK(6,3,5)} \]
951. \[ \text{ZSTP=RECWID(ZREC(I))} \]
952. \[ \text{IF(ZSIDE(I)).EQ.1 OR ZSIDE(I)).EQ.4) ZSTP=RECLEN(ZREC(I))} \]
953. \[ \text{REM1=POS(I)} \]
954. \[ \text{REM2=WID(I)} \]
955. \[ \text{WID(I)=0.0} \]
956. \[ \text{CALL TEST1} \]
PROCEDURE SGRADH.
COMMON /HEIGHT/ HEIGHT(10)
COMMON /ZBOUNO/ZBOUNO(10)
INTEGER ZBOUNO
COMMON /GRADNT/ GRADNT(6,3,5)
COMMON /BREAK/BREAK(6,3,5)
COMMON /SHRINK/SHRINK
COMMON /REM/ REM(2)
COMMON /REM1/ REM1
COMMON /REM2/ REM2

REMB=HEIGHT(1)
HEIGHT(1)=0.0
CALL TEST1
GRADNT(1,2,1)=SHRINK
BREAK(1,2,1)=HEIGHT(1)
XCREMT=ZBOUNO(1)/3.0
HEIGHT(1)=HEIGHT(1)+XCREMT
CALL TEST1
GRADNT(1,2,2)=SHRINK
BREAK(1,2,2)=HEIGHT(1)
HEIGHT(1)=HEIGHT(1)+XCREMT
CALL TEST1
GRADNT(1,2,3)=SHRINK
BREAK(1,2,3)=HEIGHT(1)
HEIGHT(1)=ZBOUNO(1)*1.00
CALL TEST1
GRADNT(1,2,4)=SHRINK
BREAK(1,2,4)=HEIGHT(1)
HEIGHT(1)=REMB
RETURN
END
SUBROUTINE SGRAOP(I)
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /RECWD/RECWD(20)
INTEGER RECWD
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /SHRINK/SHRINK
COMMON /GRADNT/GRADNT(6,3,5)
COMMON /BREAK/BREAK(6,3,5)
C 1=M
1041. C 2=H
1042. C 3=0
1043. ZSTP=RECWID(ZREG(I))
1044. IF(ZSIDE(I).EQ.1.AND.ZSIDE(I).EQ.4) ZSTP=RECLEN(ZREG(I))
1045. REM=POS(I)
1046. REM2=WIDTH(I)
1047. POS(I)=0.0
1048. CALL TEST1
1049. GRADNT(1,3,1)=SHRINK
1050. BREAK(1,3,1)=POS(I)
1051. XCREM=ZSTP/3.0
1052. POS(I)=POS(I)+XCREM
1053. CHK=WIDTH(I)+POS(I)
1054. IF(CHK.LT.ZSTP) GO TO 10
1055. WIDTH(I)=ZSTP-POS(I)
1056. CALL TEST1
1057. GRADNT(1,3,2)=SHRINK
1058. BREAK(1,3,2)=POS(I)
1059. WIDTH(I)=REM2
1060. POS(I)=POS(I)+XCREM
1061. CHK=WIDTH(I)+POS(I)
1062. IF(CHK.LT.ZSTP) GO TO 20
1063. WIDTH(I)=ZSTP-POS(I)
1064. CALL TEST1
1065. GRADNT(1,3,3)=SHRINK
1066. BREAK(1,3,3)=POS(I)
1067. WIDTH(I)=REM2
1068. POS(I)=ZSTP
1069. CHK=WIDTH(I)+POS(I)
1070. IF(CHK.LT.ZSTP) GO TO 30
1071. WIDTH(I)=ZSTP-POS(I)
1072. CALL TEST1
1073. GRADNT(1,3,4)=SHRINK
1074. BREAK(1,3,4)=POS(I)
1075. WIDTH(I)=REM2
1076. POS(I)=REM1
1077. RETURN
1078. END
1079. SUBROUTINE OUTSGD
1080. COMMON /NOZERO=YES,
1081. COMMON /ZVAL/ZVAL(10)
1082. INTEGER ZVAL,
1083. COMMON /GRADNT/GRADNT(6,3,5)
1084. CALL REDMON /BREAK/BREAK(6,3,5)
1085. WRITE(6,1)
1086. 1 FORMAT(*'THE FOLLOWING ARE THE RESULTS OF THE INITIAL CONSTANT
1087. XGRADIENT DETERMINATION')
1088. GO TO 1,NOZERO
1089. IF(ZVAL(I).EQ.0) WRITE(6,11) I
1090. 11 FORMAT(*'ZERO - ONE VARIABLE',I3) IS NOT IN THE CURRENT COMBINATION')
1091. X ATION)
1092. IF(ZVAL(I).EQ.0) GO TO 10
1093. WRITE(6,21) (BREAK(I,1,K),K=1,4)
1094. 21 FORMAT(*'WIDTH VALUES 'I,4(5X,F15.8))
1095. WRITE(6,25) [GRADNT(1,1,K),K=1,4]
1096. 25 FORMAT(*'SHRINKAGE VALUES 'I,4(5X,F15.12))
1097. WRITE(6,26)
1098. 26 FORMAT(*', 'I1)
1099. WRITE(6,22) (BREAK(I,2,K),K=1,4)
291
1100.          22 FORMAT(' ', 'HEIGHT VALUES ', 10X, F15.8)
1101.          WRITE(6, 25) (GRADNT(1, 2, K), K = 1, 4)
1102.          WRITE(6, 26)
1103.          WRITE(6, 23) (BREAK(I, 3, K), K = 1, 4)
1104.          23 FORMAT(' ', 'POSITION VALUES ', 10X, F15.8)
1105.          WRITE(6, 25) (GRADNT(1, 3, K), K = 1, 4)
1106.          10 CONTINUE
1107.          RETURN
1108.          END
1109.          SUBROUTINE CYCTRL
1110.          COMMON /CURSRK/ CURSRK
1111.          COMMON /OBJ/ OBJ
1112.          COMMON /PENLTY/ PENLTY
1113.          COMMON /PENFN2/ PENFN2
1114.          INTEGER* 2 PENFN2
1115.          COMMON /ZVAL/ ZVAL
1116.          COMMON /PCONT/ PCONT
1117.          INTEGER* 2 PCONT
1118.          COMMON /CYCSTP/ CYCSTP
1119.          COMMON /SHRINK/ SHRINK
1120.          COMMON /PENFN3/ PENFN3
1121.          INTEGER* 2 PENFN3
1122.          COMMON /C1/ C1
1123.          COMMON /CYITER/ CYITER
1124.          COMMON /ZVAL/ ZVAL
1125.          CALL OBJVAL
1126.          IF(PENFN2.EQ.1) GO TO 40
1127.          IF(PENFN3.EQ.1) GO TO 50
1128.          CYMIN = OBJ + PENLTY * (CURSRK**2)
1129.          GO TO 60
1130.          CYMIN = OBJ + PENLTY * (CURSRK + 1.0)**2
1131.          GO TO 60
1132.          CYMIN = OBJ + PENLTY * (CURSRK + 1.0)**2
1133.          CYMIN = OBJ + PENLTY * CURSRK
1134.          60 IF(PCONT(35).EQ.1) WRITE(6, 1)
1135.          1 FORMAT(' * * THE FOLLOWING IS THE CALCULATION OF THE CYCLIC
1136.          COORDINATE METHOD OF MINIMIZATION * *)
1137.          IF(PCONT(35).EQ.1) WRITE(6, 2) OBJ, CURSRK
1138.          2 FORMAT(' * * INITIAL OBJECTIVE VALUE = ', F15.6, ' * * INITIAL
1139.          SHRINKAGE = ', F15.9)
1140.          WRITE(6, 3) CYMIN
1141.          3 FORMAT(' * * TOTAL INITIAL FUNCTION VALUE WITH PENALTY = ', F15.
1142.          6)
1143.          200 CALL CYSAV1
1144.          CYITER = CYMIN
1145.          GO TO 10
1146.          IF(ZVAL(I).EQ.0) GO TO 10
1147.          CALL CYCWND(I)
1148.          IF(C1 GT CYITER) CALL CYSAV2
1149.          IF(C1 LT CYITER) CYITER = C1
1150.          IF(C1 LE CYITER) CALL CYSAV1
1151.          IF(PCONT(36).EQ.1) CALL OUTCHD(I)
1152.          CALL CYCHGT(I)
1153.          IF(C1 GT CYITER) CALL CYSAV2
1154.          IF(C1 LT CYITER) CYITER = C1
1155.          IF(C1 LE CYITER) CALL CYSAV1
1156.          IF(PCONT(38).EQ.1) CALL OUTCHTI(I)
1157.          CALL CYCPOS(I)
1158.          IF(C1 GT CYITER) CALL CYSAV2
1159.          IF(C1 LT CYITER) CYITER = C1
292

1160. IF(CI.LE.CYITER) CALL CYSAV1
1161. IF(PCONT(37).EQ.1) CALL OUTCPS(1)
1162. 10 CONTINUE
1163. CALL CYSAV2
1164. CALL OBJVAL
1165. IF(PENFN2.EQ.1) GO TO 300
1166. IF(PENFN3.EQ.1) GO TO 310
1167. 11 CONTINUE
1168. CALL CYSAV2
1169. CALL OBJVAL
1170. IF(PENFN2.EQ.1) 00 TO 300
1171. IF(PENFN3.EQ.1) GO TO 310
1172. VALNEW=OBJ+PENLTY*(SHRINK**2)
1173. GO TO 320
1174. 300 VALNEW=OBJ+PENLTY*(SHRINK+1)**2
1175. GO TO 320
1176. 310 VALNEW=OBJ+PENLTY*SHRINK
1177. 320 STOPCK=CYMIN-VALNEW
1178. IF(PCONT135).NE.1) GO TO 150
1179. WRITE I  6*4) VALNEW
1180. 4 FORMAT (',TOTAL OBJECTIVE FUNCTION VALUE AFTER AN ITERATION = ',
1181. XF15.8)
1182. WRITE!6? 5) OBJ,SHRINK
1183. 5  FORMAT (',OBJECTIVE FUNCTION = ',F15.8,' SHRINKAGE CAVITY = ',
1184. 'F10.7)
1185. WRITE!6,6) CYMIN
1186. 6 FORMAT (',OLD OBJECTIVE FUNCTION VALUE = ',F15.3)
1187. WRITE!6,7) STOPCK
1188. 7 FORMAT (',DIFFERENCE IN OBJECTIVE FUNCTION VALUE AFTER AN ITERAT
1189. XION OF CYCLIC COORDINATE = ',F15.9)
1190. 150 IF(STOPCK.GT.CYCSTP) CYMIN=VALNEW
1191. IF(STOPCK.GT.CYCSTP) GO TO 200
1192. CURSRK=SHRINK
1193. RETURN
1194. END
1195. SUBROUTINE CYSAV1
1196. COMMON /REMWID/REMWID(10)
1197. COMMON /REMPOS/REMPOS(10)
1198. COMMON /REMHGT/REMHGT(10)
1199. COMMON /WID/WID(10)
1200. COMMON /POS/POS(10)
1201. COMMON /HEIGHT/HEIGHT(10)
1202. COMMON /NOZERO/NOZERO
1203. COMMON /REMSHK/REMSHK
1204. COMMON /SHRINK/SHRINK
1205. REMSHK=SHRINK
1206. DO 10 I=1,NOZERO
1207. REMWID(I)=WID(I)
1208. REMPOS(I)=POS(I)
1209. REMHGT(I)=HEIGHT(I)
1210. 10 CONTINUE
1211. RETURN
1212. END
1213. SUBROUTINE CYSAV2
1214. COMMON /REMWID/REMWID(10)
1215. COMMON /REMPOS/REMPOS(10)
1216. COMMON /REMHGT/REMHGT(10)
1217. COMMON /WID/WID(10)
1218. COMMON /POS/POS(10)
1219. COMMON /HEIGHT/HEIGHT(10)
1220. COMMON /NOZERO/NOZERO
1221. COMMON /REMSHK/REMSHK
1222. COMMON /SHRINK/SHRINK
1223. SHRINK=REMSHK
1224. DO 10 I=1,NOZERO
1220  WID(I)=REMWID(I)
1221  POS(I)=REMPOS(I)
1222  HEIGHT(I)=REMHEIGHT(I)
1223   10 CONTINUE
1224  RETURN
1225  END
1226  SUBROUTINE CYCMID
1227  COMMON /WID/WIDI0)
1228  COMMON /HEIGHT/HEIGHT(10)
1229  COMMON /POS/POS(10)
1230  COMMON /RECWID/RECWID(20)
1231  INTEGER RECWID
1232  COMMON /RECWID(10)
1233  INTEGER ZSIDE
1234  COMMON /ZSIDE/ZSIDE(10)
1235  INTEGER ZREC
1236  COMMON /ZREC/ZREC(10)
1237  INTEGER ZREC
1238  COMMON /CLAM1/CLAM1
1239  COMMON /CLAM2/CLAM2
1240  COMMON /C1/C1
1241  COMMON /C2/C2
1242  COMMON /PCONT/PCONT(50)
1243  INTEGER* 2 PCONT
1244  COMMON /CYWSTP/CYWSTP
1245  COMMON /XWID/XWID
1246  COMMON /SHRINK/SHRINK
1247  COMMON /SHK1/SHK1
1248  COMMON /SHK2/SHK2
1249  XWID=RECWID(ZREC(1))
1250  XLOW=0.0
1251  IF(ZSIDE(I).EQ.1.0.ZSIDE(I).EQ.4) XWID=RECLEN(ZREC(I))
1252  U=XWID
1253  B=XLOW
1254  REM1=WID(I)
1255  REM2=POS(I)
1256  CLAM1=0.382*(U-B)+B
1257  CLAM2=0.510*(U-B)+B
1258  CALL CMI01(I)
1259  CALL CMID2(I)
1260   10 DIFF=CLAM2-CLAM1
1261  IF(PCONT(39).NE.1) GO TO 15
1262  WRITE(6,1)
1263  1 FORMAT('INTERMEDIATE CALCULATIONS OF THE WIDTH PORTION OF THE
1264  CYCLIC COORDINATE MINIMIZATION*)
1265  WRITE(6,2) CLAM1,CLAM2
1266  2 FORMAT('VALUE OF WIDTH ONE = ',F15.6,' VALUE OF WIDTH TWO = ',F15.6)
1267  X)
1268  WRITE(6,3) B,U
1269  3 FORMAT('VALUE OF B = ',F15.6,' VALUE OF U = ',F15.6)
1270  WRITE(6,4) DIFF
1271  4 FORMAT('DIFERENCE IN SEARCH VALUES = ',F15.10)
1272  WRITE(6,5) C1,C2
1273  5 FORMAT('C1 = ',F15.4,' C2= ',F15.4)
1274  15 IF(DIFF.LT.CYWSTP) GO TO 200
1275  IF(C1.GT.C2) GO TO 100
1276  U=CLAM2
1277  CLAM2=CLAM1
1278  B=B
1279  C2=C1
SHK2=SHK1
POS(I)=REM2
CLAM1=0.382*(U-B)+B
CALL CWID1(I)
GO TO 10

B=CLAM1
CLAM1=CLAM2
U=U
CI=C2
SHK1=SHK2
POS(I)=REM2
CLAM2=0.618*(U-B)+B
CALL CWID1(I)
GO TO 10

(IF(CL<LT.C2) GO TO 220
WID(I)=CLAM2
CI=C2
SHRINK=SHK2
POS(I)=REM2
CHECK=WID(I)+POS(I)
IF(CHECK<LT.XWID) GO TO 500
POS(I)=XWID-WID(I)
GO TO 500

220 WID(I)=CLAM1
CI=C1
SHRINK=SHK1
POS(I)=REM2
CHECK=WID(I)+POS(I)
IF(CHECK<LT.XWID) GO TO 500
POS(I)=XWID-WID(I)
500 RETURN

END

COMMON /WID/WID(I)
COMMON /POS/POS(I)
COMMON /CLAM/CLAM1
COMMON /XWID/XWID
COMMON /SHRINK/SHRINK
COMMON /PENFN/PENFN2
INTEGER*2 PENFN2
COMMON /C1/C1
COMMON /PCONT/PCONT(50)
INTEGER*2 PCONT
COMMON /OBJ/OBJ
COMMON /PEN/PENLT
COMMON /PENFN3/PENFN3
INTEGER*2 PENFN3
COMMON /SHK1/SHK1
COMMON /WID/I=CLAM1
CHECK=WID(I)+POS(I)
IF(CHECK<LT.XWID) GO TO 100
POS(I)=XWID-WID(I)
100 CALL TEST1
SHK1=SHRINK
CALL OBJVAL
IF(PENFN2.EQ.1) GO TO 10
IF(PENFN3.EQ.1) GO TO 20
CI=OBJ+PENLT*(SHRINK**2)
GO TO 30
10 CI=OBJ+PENLT*{(SHRINK+1.0)**2}
GO TO 30

20 C1=OBJ+PENLTY*SHRINK

30 IF(PCONT(39).NE.1) RETURN

WRITE(6,1)

1 FORMAT( ' ', 'INTERMEDIATE RESULTS IN CWID1!

WRITE(6,2) CHECK,XW10,POS(I),WID(I),C1*SHRINK

2 FORMAT( ' ', 'CHECK = ',F15.6,' XW10 = ',F10.5,' POSITION = ',F10.6,
X* WIDTH = ',F10.6,' OBJ = ',F15.6,' SHRINK = ',F10.3)

RETURN

END

SUBROUTINE CWID2(I)

COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /CLAM2/CLAM2
COMMON /XWID/XWID
COMMON /PENLTY/PENLTY
COMMON /SHRINK/SHRINK
COMMON /PENFN2/PENFN2

INTEGER*2 PENFN2
COMMON /OBJ/OBJ
COMMON /C2/C2
COMMON /PCONT/PCONT(50)

INTEGER*2 PCONT
COMMON /PENFN3/PENFN3

INTEGER*2 PENFN3
COMMON /SHK2/SHK2
WIDTH(I)=CLAM2
CHECK=WID(I)+POS(I)

IF(CHECK.LT.XWID) GO TO 100

POS(I)=XWID-WID(I)

CALL TEST1

SHK2=SHRINK
CALL OBJVAL

IF(PENFN2.EQ.1) GO TO 10
IF(PENFN3.EQ.1) GO TO 20

C2=OBJ+PENLTY*(SHRINK**2)
GO TO 30

GO TO 30

G0 TO 30

20 C2=OBJ+PENLTY*SHRINK

30 IF(PCONT(39).NE.1) RETURN

WRITE(6,1)

1 FORMAT( ' ', 'INTERMEDIATE RESULTS IN CWID2!

WRITE(6,2) POS(I),WID(I),XWID,CHECK,C2,SHRINK

2 FORMAT( ' ', 'POSITION = ',F10.6,' WIDTH = ',F10.6,' XWID = ',F10.6,
X* CHECK = ',F10.6,' OBJ C2 = ',F15.6,' SHRINK = ',F10.3)

RETURN

END

SUBROUTINE CYCPOS(I)

COMMON /POS/POS(10)
COMMON /WID/WID(10)
COMMON /RECWID/RECWID(20)

INTEGER RECWID
COMMON /RECN/RECN(20)

INTEGER RECN
COMMON /ZSIDE/ZSIDE(10)

INTEGER ZSIDE
COMMON /ZREC/ZREC(10)

INTEGER ZREC
COMMON /CLAM1/CLAM1
1400. COMMON /CLAM2/CLAM2
1401. COMMON /PCONT/PCONT(50)
1402. INTEGER* 2 PCONT
1403. COMMON /CYPSTP/CYPSTP
1404. COMMON /C1/C1
1405. COMMON /C2/C2
1406. COMMON /XWID/XWID
1407. COMMON /IFLAG1/IFLAG1
1408. INTEGER* 2 IFLAG1
1409. COMMON /IFLAG2/IFLAG2
1410. INTEGER* 2 IFLAG2
1411. COMMON /SHRINK/SHRINK
1412. COMMON /SHK1/SHK1
1413. COMMON /SHK2/SHK2
1414. XWID=RECWID(ZREC(I))
1415. XLO=0.0
1416. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XWID=RECLNIZREC(I)
1417. U=XWID-WID(I)
1418. B=XLOW
1419. REM1=WID(I)
1420. REM2=POS(I)
1421. CLAM1=0.382*(U-B)+B
1422. CLAM2=0.618*(U-B)+B
1423. CALL CP0S1(I)
1424. CALL CP0S2(I)
1425. 10 DIFF=CLAM2-CLAM1
1426. IF(PCONT(40).NE.1) GO TO 15
1427. WRITE(6,1)
1428. 1 FORMAT( 'FOLLOWING ARE INTERMEDIATE RESULTS OF THE CYCLIC COORD'
1429. 'INATE APPLIED TO POSITION)
1430. WRITE(6,2) I
1431. 2 FORMAT( 'VARIABLE = ',I4)
1432. WRITE(6,3) CL1,C2
1433. 3 FORMAT( 'OBJECTIVE FOR CLAM1 = ',F15.8,' FOR CLAM2 = ',F15.8)
1434. WRITE(6,4) CLAM1,CLAM2
1435. 4 FORMAT( 'CLAM1 = ',F15.9,' CLAM2 = ',F15.9)
1436. WRITE(6,5) U,B
1437. 5 FORMAT( 'U = ',F15.8,' B = ',F15.8)
1438. WRITE(6,6) DIFF
1439. 6 FORMAT( 'DIFFERENCE IN POSITION = ',F15.10)
1440. WRITE(6,7) CYPSTP,XWID
1441. 7 FORMAT( 'STOP CRITERION = ',F10.5,' XIWD = ',F8.3)
1442. 15 IF(DIFF.LT.CYPSTP) GO TO 200
1443. IF(C1.GT.C2) GO TO 100
1444. U=CLAM2
1445. CLAM2=CLAM1
1446. C2=C1
1447. SHK2=SHK1
1448. B=B
1449. CLAM1=0.382*(U-B)+B
1450. CALL CP0S1(I)
1451. GO TO 10
1452. 100 B=CLAM1
1453. CLAM1=CLAM2
1454. U=U
1455. C1=C2
1456. SHK1=SHK2
1457. CLAM2=0.618*(U-B)+B
1458. CALL CP0S2(I)
1459. GO TO 10
200 IF(C1.LT.C2) GO TO 220
201 C1=C2
202 SHRINK=SHK2
203 POS(I)=CLAM2
204 GO TO 500
220 C1=C1
221 SHRINK=SHK1
222 POS(I)=CLAM1
223 500 RETURN

END

SUBROUTINE CPS1(I)
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /CLAM1/CLAM1
COMMON /XWID/XWID
COMMON /SHRINK/SHRINK
COMMON /PENLTY/PENLTY
COMMON /PENFN2/PENFN2
INTEGER*2 PENFN2
COMMON /OBJ/OBJ
COMMON /C1/C1
COMMON /IFLAG1/IFLAG1
INTEGER*2 IFLAG1
COMMON /PCONT/PCONT(50)
INTEGER*2 PCONT
COMMON /PENFN3/PENFN3
INTEGER*2 PENFN3
COMMON /SHK1/SHK1
COMMON /OBJ/OBJ
COMMON /C1/C1
COMMON /IFLAG1/IFLAG1
COMMON /PCONT/PCONT(50)
COMMON /OBJ/OBJ
COMMON /C1/C1
IF(IFLAG1.LT.0) GO TO 100
IF(IFLAG1.NE.0) GO TO 15

WRITE(6,1)
1  FORMAT('INTERMEDIATE RESULTS OF CP0S1*)
WRITE(6,2) WID(I),POS(I),CLAM1,XWID
2  FORMAT('WID = ',F10.6,' POS = ',F10.6,' CLAM1 = ',F10.6,' XWID = ',F10.6)
WRITE(6,3) SHRINK,PENLTY,OBJ,C1
3  FORMAT('SHRINK = ',F15.8,' PENLTY = ',F15.2,' OBJ = ',F15.8,' C1 = ',F15.8)
15 RETURN

END

SUBROUTINE CPS2(I)
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /CLAM2/CLAM2
COMMON /XWID/XWID

298
COMMON /PENLTY/PENLTY
COMMON /SHRINK/SHRINK
COMMON /PENFN2/PENFN2
INTEGER*2 PENFN2
COMMON /OBJ/OBJ
COMMON /IFLAG2/IFLAG2
INTEGER*2 IFLAG2
COMMON /C2/C2
COMMON /PCONT/PCONT
INTEGER*2 PCONT
COMMON /PENFN3/PENFN3
INTEGER*2 PENFN3
COMMON /SHK2/SHK2
 INTEGER2 IFLAG2=0
POS(I)=CLAM2
XCHECK=WID(I)+POS(I)
IF(XCHECK.LT.XWID) GO TO 100
IFLAG2=1
WID(I)=XWID-POS(I)
100 CALL TEST1
SHK2=SHRINK
CALL OBJVAL
IF(PENFN2.EQ.1) GO TO 10
IF(PENFN3.EQ.1) GO TO 20
C2=OBJ+PENLTY*(SHRINK**2)
GO TO 30
10 C2=OBJ*PENLTY*(SHRINK+1.0)**2
GO TO 30
20 C2=OBJ*PENLTY*SHRINK
30 IF(IPCONT.10).NE.1) RETURN
WRITE(6,1)
1 FORMAT(' 'INTERMEDIATE RESULTS OF CPQS2')
WRITE(6,2) POS(I),WID(I),IFLAG2,XCHECK,WID1,C2,SHRINK
2 FORMAT(' 'X',F8.5,'XWID = ',F8.5,' OBJ C2 = ',F15.8,' SHRINK = ',F10.7)
RETURN
END
SUBROUTINE CYCHGT(I)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /CYHTS/PCONT
COMMON /SHRINK/SHK1
COMMON /SHK2/SHK2

XHGT=ZBOUND
U=XHGT
B=XL0W
REM1=HEIGHT(I)
CLAM1=0.332*(U—B)+B
CLAM2=0.618*(U—B)+B
CALL CHGT1(I)
CALL CHGT2(I)

DIFF=CLAM2-CLAM1

IF(PCONT(I).GT.0) GO TO 15
WRITE(6,1) INTERMEDIATE RESULTS OF CYCLIC COORDINATE FOR HEIGHT
WRITE(6,2) VARIABLE = ,14
WRITE(6,3) CLAM1,CLAM2
WRITE(6,4) CLAM1 = ,F15.8, CLAM2 = ,F15.8
WRITE(6,5) DIFFERENCE IN HEIGHTS = ,F15.8
WRITE(6,6) U,B
WRITE(6,7) CYHTS
STOP CRITERION FOR HEIGHT IN CYCLIC COORDINATE = ,F15 X.B

IF(DIFF.LT.CYHTS) GO TO 200
IF(C1.GT.C2) GO TO 100
U=CLAM2
CLAM2=CLAM1
B=B
C2=C1
SHK2=SHK1
CLAM1=0.332*(U—B)+B
CALL CHGT1(I)
GO TO 10

B=CLAM1
CLAM1=CLAM2
U=U
C1=C2
SHK1=SHK2
CLAM2=0.618*(U—B)+B
CALL CHGT2(I)
GO TO 10

IF(C1.LT.C2) GO TO 220
C1=C2
SHRINK=SHK2
HEIGHT(I)=CLAM2
GO TO 500
220 C1=C1
SHRINK=SHK1
HEIGHT(I)=CLAM1
500 RETURN

END

SUBROUTINE CHGT1(I)

COMMON /HEIGHT/HEIGHT(10)
COMMON /CLAM1/CLAM1
COMMON /SHRINK/SHRINK
COMMON /PENFN2/PENFN2
INTEGER*2 PENFN2
COMMON /PENLTY/PENLTY
COMMON /OBJ/OBJ
COMMON /CL/C1
COMMON /PCONT/PCONT(50)
INTEGER*2 PCONT
COMMON /PENFN3/PENFN3
INTEGER*2 PENFN3
COMMON /SHK1/SHK1
HEIGHT(I)=CLAM1
CALL TEST1
SHK1=SHRINK
CALL OBJVAL
IF(PENFN2.EQ.1)GO TO 10
IF(PENFN3.EQ.1)GO TO 20
C1=OBJ+PENLTY*(SHRINK**2)
GO TO 30
10 C1=OBJ+PENLTY*((SHRINK+1.0)**2)
GO TO 30
20 C1=OBJ+PENLTY*SHRINK
GO TO 30
30 IF(PCONT(I).NE.1)RETURN
WRITE(6,1)
1 FORMAT(' '*INTERMEDIATE RESULTS IN CHGT1*')
WRITE(6,2) C1,OBJ,SHRINK,CLAM1,HEIGHT(I),I
2 FORMAT(' '*C1 = ',F14.7,' OBJ = ',F14.7,' SHRINK = ',F10.5,' CLAM X1 = ',F14.7,' HEIGHT(I) = ',F8.5,' VARIABLE = ',I3)
RETURN
END

SUBROUTINE CHGT2(I)

COMMON /HEIGHT/HEIGHT(10)
COMMON /CLAM2/CLAM2
COMMON /SHRINK/SHRINK
COMMON /PENFN2/PENFN2
INTEGER*2 PENFN2
COMMON /PENLTY/PENLTY
COMMON /C2/C2
COMMON /OBJ/OBJ
COMMON /PCONT/PCONT(50)
INTEGER*2 PCONT
COMMON /PENFN3/PENFN3
INTEGER*2 PENFN3
COMMON /SHK2/SHK2
HEIGHT(I)=CLAM2
CALL TEST1
SHK2=SHRINK
CALL OBJVAL
IF(PENFN2.EQ.1)GO TO 10
IF(PENFN3.EQ.1)GO TO 20
C2=OBJ+PENLTY*(SHRINK**2)
GO TO 30
10 C2=OBJ+PENLTY*(SHRINK+1.0)**2
GO TO 30
20 C2=OBJ+PENLTY*SHRINK
30 IF (PCONT(+1).*NE.+1) RETURN
WRITE(6+1)
1 FORMAT(*,'INTERMEDIATE RESULTS OF CHNG2')
WRITE(6+2) HEIGHT(I),CLAM2,OBJ,C2,SHRINK,I
2 FORMAT(*,'HEIGHT = ',F8.5,' CLAM2 = ',F12.3,' OBJ = ',F15.8,' C2
X = ',F15.8,' SHRINK = ',F12.8,' VARIABLE ',I3)
RETURN
END
SUBROUTINE OUTCWO(I)
COMMON /C1/C1
COMMON /CYITER/CYITER
COMMON /SHRINK/SHRINK
WRITE(6.1) I
1 FORMAT('FINAL OUTPUT OF A CYCLE OF VARIABLE ',I2,' ON AN ITERAT
XION OF THE CYCLIC COORDINATE')
WRITE(6.2)
2 FORMAT('* -20X,'WIDTH VARIABLE')
CALL OUTVAR
WRITE(6.3) C1
3 FORMAT(' -20X,'HEIGHT VARIABLE')
CALL OUTVAR
WRITE(6.4) C1
4 FORMAT(' -20X,'HEIGHT VARIABLE')
RETURN
END
SUBROUTINE OUTCP(I)
COMMON /C1/C1
COMMON /CYITER/CYITER
COMMON /SHRINK/SHRINK
WRITE(6.1) I
1 FORMAT('FINAL OUTPUT OF A CYCLE OF VARIABLE ',I2,' ON AN ITERAT
XION OF THE CYCLIC COORDINATE')
WRITE(6.2)
2 FORMAT('* -20X,'WIDTH VARIABLE')
CALL OUTVAR
WRITE(6.3) C1
3 FORMAT(' -20X,'HEIGHT VARIABLE')
CALL OUTVAR
WRITE(6.4) C1
4 FORMAT(' -20X,'HEIGHT VARIABLE')
RETURN
END
SUBROUTINE OUTCH(I)
COMMON /C1/C1
COMMON /CYITER/CYITER
COMMON /SHRINK/SHRINK
WRITE(6.1) I
1 FORMAT('FINAL OUTPUT OF A CYCLE OF VARIABLE ',I2,' ON AN ITERAT
XION OF THE CYCLIC COORDINATE')
WRITE(6.2)
2 FORMAT('* -20X,'HEIGHT VARIABLE')
CALL OUTVAR
WRITE(6.3) C1
3 FORMAT(' -20X,'HEIGHT VARIABLE')
CALL OUTVAR
WRITE(6.4) C1
4 FORMAT(' -20X,'HEIGHT VARIABLE')
RETURN
END
WRITE(6,5) CYTER
5 FORMAT(' *20X,'CURRENT ITERATION MINIMUM VALUE = ',F15.8)
WRITE(6,4) SHRINK
4 FORMAT(' *20X,'SHRINKAGE VALUE = ',F15.8)
RETURN
END
SUBROUTINE STEEP
COMMON /GVAL/GVAL
COMMON /STPSTP/STPSTP
COMMON /PCONT/PCONT(50)
INTEGER*2 PCONT
INTEGER*2 MAXBND
IT=1
IF(PCONT(42).EQ.1) WRITE(6,1) IT
1 FORMAT(1X,'ITERATION NUMBER ',I3,' OF THE STEEPEST DESCENT ALGORITHM)
CALL GRAD
CALL STPDIR
IF(MAXBND.EQ.1) CALL 80UND2
IF(MAXBND.EQ.1) GO TO 2
CALL BOUND
2 CALL GOLDEN
IF(PCONT(22).EQ.1) CALL OUTGLD
10 CURRENT=GVAL
CALL STPSAV
CALL GRAD
CALL STPDIR
IF(PCONT(2).EQ.1) CALL OUTGRA
IF(PCONT(3).EQ.1) CALL OUTUIK
IF(MAXBND.EQ.1) CALL BOUND2
IF(MAXBND.EQ.1) GO TO 12
CALL BOUND
12 CALL GOLDEN
IF(PCONT(22).EQ.1) CALL OUTGLD
0 DIFF=CURRNT-GVAL
IF(DIFF.LT.0.0) CALL STPGET
IF(DIFF.LT.STPSTP) GO TO 20
IT=IT+1
IF(PCONT(42).EQ.1) WRITE(6,1) IT
100 CURRNT=GVAL
GO TO 10
20 IF(PCONT(42).EQ.1) WRITE(6,30)
30 FORMAT('OPTIMAL SOLUTION TO THE STEEPEST DESCENT ALGORITHM')
IF(PCONT(42).EQ.1) CALL OUTVAR
RETURN
END
SUBROUTINE STPSAV
COMMON /HEIGHT/HEIGHT(10)
COMMON /POS/POS(10)
COMMON /RSTPH/RSTPH(10)
COMMON /RSTPP/RSTPP(10)
COMMON /NOZERO/NOZERO
00 10 I=1,NOZERO
10 RSTPP(I)=POS(I)
10 RSTPH(I)=HEIGHT(I)
CONTINUE
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /RSTPP/RSTPP(10)
COMMON /RSTPH/RSTPH(10)
COMMON /RSTPW/RSTPW(10)
COMMON /NOZERO/NOZERO
DO 10 I=1,NOZERO
WID(I)=RSTPW(I)
POS(I)=RSTPP(I)
HEIGHT(I)=RSTPH(I)
10 CONTINUE
RETURN
END
SUBROUTINE STPGET
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /RSTPP/RSTPP(10)
COMMON /RSTPH/RSTPH(10)
COMMON /RSTPW/RSTPW(10)
COMMON /NOZERO/NOZERO
DO 10 I=1,NOZERO
WID(I)=RSTPW(I)
POS(I)=RSTPP(I)
HEIGHT(I)=RSTPH(I)
10 CONTINUE
RETURN
END
SUBROUTINE STPOIR
COMMON /PGRAO/PGRAO(10)
COMMON /HGRAD/HGRAD(10)
COMMON /WGRAD/WGRAD(10)
COMMON /NOZERO/NOZERO
COMMON /DIRECT/DIRECT(20)
00 I 1=1,20
DIRECT(I)=0.0
1 CONTINUE
1=
I=I+1
DO 10 I=1,12
DIRECT(I)=PGRAO(I)*(-1.0)
10 CONTINUE
DO 20 I=1,NOZERO
DIRECT(I)=HGRAD(I)*(-1.0)
20 CONTINUE
DO 30 I=1,NOZERO
DIRECT(I)=WGRAD(I)*(-1.0)
30 CONTINUE
RETURN
END
SUBROUTINE RON
COMMON /PGRAO/PGRAO(10)
COMMON /HGRAD/HGRAD(10)
COMMON /WGRAD/WGRAD(10)
PGRAO(1)=3.50
PGRAO(2)=3.00
HGRAD(1)=11.64
HGRAD(2)=12.40
WGRAD(1)=-89.8091431
WGRAD(2)=10.5625946
XRON=2.0
RETURN
END
SUBROUTINE TEST1
CALL TEST
RETURN
END
SUBROUTINE EQUATN
COMMON /SHRINK/SHRINK
304

1860. COMMON /WID/WID(10)
1861. COMMON /HEIGHT/HEIGHT(10)
1862. COMMON /POS/POS(10)
1863. COMMON /PENFN2/PENFN2
1864. INTEGER*2 PENFN2
1865. COMMON /SHK70L/SHK70L
1866. SHRINK=0.0
1867. SHRINK=SHRINK+(-0.20*WID(I)+(-0.1000*HEIGHT(I))+(1.0000*POS(I))
1868. SHRINK=SHRINK+(-0.15*WID(I)+(-0.21*HEIGHT(I))+(-0.002*POS(I))
1869. SHRINK=SHRINK+(-1.15*WID(I)+(-2.21*HEIGHT(I))+(-0.002*POS(I))
1870. SHRINK=SHRINK+(-1.15*WID(I)+(-2.21*HEIGHT(I))+(-0.002*POS(I))
1871. SHRINK=SHRINK+(-2.22*WID(I)+(-1.11*HEIGHT(I))+(-0.002*POS(I))
1872. SHRINK=SHRINK+(-2.22*WID(I)+(-1.11*HEIGHT(I))+(-0.002*POS(I))
1873. SHRINK=SHRINK+4.42069
1874. IF(SHRINK.LT.SHK70L) SHRINK=0.0
1875. IF(SHRINK.LT.SHK70L) SHRINK=-1.0
1876. WRITE(6,100) SHRINK
1877. 100 FORMAT(1X,'SHRINKAGE VALUE FROM EQUATION = ',F12.8)
1878. CALL OUTVAR
1879. RETURN
1880. END
1881. SUBROUTINE GLDNEG
1882. COMMON /WID/WID(10)
1883. COMMON /POS/POS(10)
1884. COMMON /HEIGHT/HEIGHT(10)
1885. COMMON /NOZERO/NOZERO
1886. DO 10 I=1,NOZERO
1887. IF(WID(I).LT.0.0) WID(I)=0.0
1888. IF(HEIGHT(I).LT.0.0) HEIGHT(I)=0.0
1889. IF(POS(I).LT.0.0) POS(I)=0.0
1890. 10 CONTINUE
1891. RETURN
1892. END
1893. SUBROUTINE GLDHGT
1894. COMMON /NOZERO/NOZERO
1895. COMMON /ZBOUND/ZBOUND(10)
1896. INTEGER*2 ZBOUND
1897. COMMON /HEIGHT/HEIGHT(10)
1898. DO 10 I=1,NOZERO
1899. X=ZBOUND(I)
1900. IF(HEIGHT(I).GT.X) HEIGHT(I)=ZBOUND(I)
1901. 10 CONTINUE
1902. RETURN
1903. END
1904. SUBROUTINE INPUT
1905. COMMON /PCONT/PCONT(50)
1906. INTEGER*2 PCONT
1907. CALL INDES
1908. CALL INALP
1909. CALL INOBJ
1910. CALL INTYP
1911. CALL OUTGEN
1912. RETURN
1913. END
1914. SUBROUTINE INDES
1915. COMMON /X/X(60,60)
1916. INTEGER*2 X
1917. COMMON /LENGTH/LENGTH
1918. INTEGER LENGTH
1919. COMMON /WIDTH/WIDTH
INTEGER WIDTH
COMMON /NUMREC/NUMREC
INTEGER NUMREC
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /RECWID/RECWID(20)
INTEGER RECWID
COMMON /ZCORNR/ZCORNR(10,4,2)
INTEGER ZCORNR
COMMON /NOZERO/NOZERO
INTEGER NOZERO
COMMON /ZTYPE/ZTYPE(10)
INTEGER ZTYPE
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /ZCON/ZCON(10)
INTEGER ZCON

READ(IN,100) NUMREC

FORMAT(IX,12)
DO 200 I=1,NUMREC
READ(IN,201) RECLEN(I),RECWID(I)
FORMAT(IX,12)
200 CONTINUE
READ(IN,300) NOZERO
FORMAT(IX,12)
DO 310 I=1,NOZERO
READ(IN,320) ZTYPE(I),ZBOUND(I),ZSIDE(I),ZREC(I),ZCON(I)
FORMAT(IX,5(12,IX))
320 CONTINUE
310 CONTINUE
READ(IN,400) LENGTH,WIDTH
FORMAT(IX,12)
DO 410 J=1,WIDTH
READ(IN,420) (X(I,J),I=1,LENGTH)
FORMAT(IX,60U)
410 CONTINUE
RETURN
END

SUBROUTINE INALP
COMMON /ALP/ALP(27)
LOGICAL*1 ALP
READ(5,10) ( ALP(I),I=1,27)
FORMAT(27A1)
RETURN
END

SUBROUTINE INOBJ
COMMON /NOZERO/NOZERO
INTEGER NOZERO
COMMON /COEFW/COEFW(10)
COMMON /CQEFP/CQEFP(10)
COMMON /COEFH/COEFH(10)
COMMON /CCEFWH/CCEFWH(10)
1980. COMMON /COEFZ/COEFZ(10)
1981. COMMON /COEFPH/COEFPH(10)
1982. COMMON /COEFW/COEFW(10)
1983. COMMON /COEFW2/COEFW2(10)
1984. COMMON /COEFH2/COEFH2(10)
1985. COMMON /COEFW2/COEFW2(10)
1986. DO 10 I=1,NOZERO
1987. READ(5,20) COEFZ(I),COEFW2(I),COEFH2(I),COEFP2(I)
1988. 20 FORMAT(4(F8.2))
1989. READ(5,30) COEFWH(I),COEFWP(I),COEFPH(I)
1990. 30 FORMAT(3(F8.2))
1991. READ(5,40) COEFH2(I),COEFPH2(I),COEFP2(I)
1992. 40 FORMAT(3(F8.2))
1993. 10 CONTINUE
1994. RETURN
1995. END
1996. SUBROUTINE INTYP
1997. COMMON /TYP/TYP(10,6)
1998. LOGICAL *1 TYP
1999. COMMON /NGZERO/NGZERO
2000. DO 10 I=1,NOZERO
2001. READ(5,20) TYP(I,J),J=1,6)
2002. 20 FORMAT(6A1)
2003. 10 CONTINUE
2004. RETURN
2005. END
2006. SUBROUTINE OUTGEN
2007. COMMON /X/X(160,60)
2008. INTEGER *2 X
2009. COMMON /LENGTH/LENGTH
2010. INTEGER LENGTH
2011. COMMON /WIDTH/WIDTH
2012. INTEGER WIDTH
2013. COMMON /NUMREC/NUMREC
2014. INTEGER NUMREC
2015. COMMON /RECLEN/RECLEN(20)
2016. INTEGER RECLEN
2017. COMMON /RECWID/RECWID(20)
2018. INTEGER RECWID
2019. COMMON /ZCORNR/ZCORNR(10,4,2)
2020. INTEGER ZCORNR
2021. COMMON /NOZERO/NOZERO
2022. INTEGER NOZERO
2023. COMMON /ZTYPE/ZTYPE(10)
2024. INTEGER ZTYPE
2025. COMMON /ZBOUND/ZBOUND(10)
2026. INTEGER ZBOUND
2027. COMMON /ZSIDE/ZSIDE(10)
2028. INTEGER ZSIDE
2029. COMMON /ZREG/ZREG(10)
2030. INTEGER ZREG
2031. COMMON /ZCON/ZCON(10)
2032. INTEGER ZCON
2033. COMMON /COEFW/COEFW(10)
2034. COMMON /COEFPH/COEFPH(10)
2035. COMMON /COEFP/COEFP(10)
2036. COMMON /COEFZ/COEFZ(10)
2037. COMMON /COEFWH/COEFWH(10)
2038. COMMON /COEFWP/COEFWP(10)
2039. COMMON /COEFPH/COEFPH(10)
COMMON /COEFW2/COEFW2(10)
COMMON /COEFH2/COEFH2(10)
COMMON /COEFP2/COEFP2(10)
COMMON /TYP/TYPI10,6)
LOGICAL * 1 TYP
WRITE(6,1)
 1 FORMAT('1',55X,'GENERAL INFORMATION')
WRITE(6,2)
 2 FORMAT('-','')
WRITE(6,3)
 3 FORMAT(' ','')
WRITE(6,10)
 10 FORMAT(' ','THE FOLLOWING INFORMATION IS IN REGARD TO THE GEOMETRY
XOF THE CASTING')
WRITE(6,11)
 11 FORMAT(' ','10X,'OVERALL FLASK DIMENSIONS')
WRITE(6,12) LENGTH
 12 FORMAT(' ','20X,'LENGTH OF FLASK = ',14,' INCHES')
WRITE(6,13) WIDTH
 13 FORMAT(' ','20X,'WIDTH OF FLASK = ',14,' INCHES')
WRITE(6,14)
 14 FORMAT(' ','10X,'INFORMATION ON THE MAKEUP OF THE CASTING GEOMETRY')
WRITE(6,15) NUMREC
 15 FORMAT(' ','20X,'NUMBER OF RECTANGLES = ',13)
WRITE(6,16)
 16 FORMAT(' ','20X,'INDIVIDUAL RECTANGLE INFORMATION FOLLOWS:')
DO 20 I=1,NUMREC
WRITE(6,21) I,RECLEN11).RECWIDII
 21 FORMAT(' ','30X,RECTANGLE ',13,' DIMENSION - LENGTH = ',15,' X WIDTH = ',15)
20 CONTINUE
WRITE(6,30)
 30 FORMAT(' ','47X,'GRAPHIC DISPLAY OF CASTING GEOMETRY')
WRITE(6,31)
 31 FORMAT(' ','10X,'CASTING GEOMETRY CODE')
WRITE(6,32)
 32 FORMAT(' ','20X,'0 - INDICATES EXTERNAL AIR BARRIER')
308

2081. WRITE(6,33)
2082. 33 FORMAT(' ',20X,'1 - INDICATES SAND MATERIAL')
2083. WRITE(6,34)
2084. 34 FORMAT(' ',20X,'2 - INDICATES CASTING')
2085. WRITE(6,35)
2086. 35 FORMAT(' ',20X,'3 - INDICATES CHILL MATERIAL')
2087. WRITE(6,36)
2088. 36 FORMAT(' ',20X,'4 - INDICATES RISER MATERIAL')
2089. WRITE(6,37)
2090. 37 FORMAT(' ',20X,'5 - INDICATES CASTING AREA IN CONTACT WITH SAND')
2091. WRITE(6,38)
2092. 38 FORMAT(' ',20X,'6 - INDICATES SAND AREA IN CONTACT WITH CASTING')
2093. WRITE(6,39)
2094. 39 FORMAT(' ',20X,'7 - INDICATES CASTING AREA IN CONTACT WITH COOLING')
2095. WRITE(6,40)
2096. 40 FORMAT(' ',20X,'8 - INDICATES SAND AREA IN CONTACT WITH COOLING')

2097. DO 40 I=1,LENGTH
2098. 41 FORMAT(' ',20X,'1011)
2099. 40 CONTINUE
2100. WRITE(6,50)
2101. 50 FORMAT(' ',20X,'ZERO-ONE VARIABLE GENERAL INFORMATION')
2102. WRITE(6,51) NOZERO
2103. 51 FORMAT(' ',20X,'THERE ARE ',12,' ZERO - ONE VARIABLES')
2104. WRITE(6,52)
2105. 52 FORMAT(' ',20X,'INFORMATION ON EACH ZERO - ONE VARIABLE FOLLOWS:')
2106. DO 100 I=1,NOZERO
2107. WRITE(6,101) I
2108. 101 FORMAT(' ',20X,'ZERO - ONE VARIABLE ',13)
2109. WRITE(6,102) ITYP(I,J),J=1,6)
2110. 102 FORMAT(' ',20X,'TYPE - ',6A1)
2111. WRITE(6,103)
2112. 103 FORMAT(' ',20X,'CORNER EXTREME POSITIONS')
2113. WRITE(6,104) (ZCORNRI(I,1,K),K=1,2)
2114. 104 FORMAT(' ',20X,'UPPER LEFT CORNER ROW = ',14,' COLUMN = ',14)
2115. WRITE(6,105) (ZCORNRI(I,2,K),K=1,2)
2116. 105 FORMAT(' ',20X,'UPPER RIGHT CORNER ROW = ',14,' COLUMN = ',14)
2117. WRITE(6,106) (ZCORNRI(I,3,K),K=1,2)
2118. 106 FORMAT(' ',20X,'LOWER LEFT CORNER ROW = ',14,' COLUMN = ',14)
2119. WRITE(6,107) (ZCORNRI(I,4,K),K=1,2)
2120. 107 FORMAT(' ',20X,'LOWER RIGHT CORNER ROW = ',14,' COLUMN = ',14)
2121. WRITE(6,110) ZBOUND(I)
2122. 110 FORMAT(' ',20X,'BOUND ON HEIGHT OR WIDTH DEPENDING ON CONFIGURATION')
2123. IF(ZSIDE(I).EQ.1 OR.ZSIDE(I).EQ.4) WRITE(6,112) RECLLEN(REC(I))
2124. IF(ZSIDE(I).EQ.2 OR.ZSIDE(I).EQ.3) WRITE(6,112) REC(ZREC(I))
2125. 112 FORMAT(' ',20X,'BOUND ON CASTING CONTACT LENGTH = ',14)
2126. WRITE(6,120) ZSIDE(I)
2127. 120 FORMAT(' ',20X,'SIDE OF ZERO - ONE VARIABLE CONTACTING SIDE OF RECTANGLE - ',13)
2128. WRITE(6,121) ZREC(I)
2129. 121 FORMAT(' ',20X,'RECTANGLE NUMBER WHICH ZERO - ONE VARIABLE CONTACTS')
2130. XS - ',13)
2131. WRITE(6,122) ZCON(I)
2132. 122 FORMAT(' ',20X,'SIDE OF RECTANGLE WHICH ZERO - ONE VARIABLE CONTACTS')
309

2140. XTS = 13
2141. 100 CONTINUE
2142. WRITE(6,200)
2143. 200 FORMAT('1',44X,'OBJECTIVE FUNCTION COEFFICIENT INFORMATION')
2144. WRITE(6,2)
2145. WRITE(6,201)
2146. 201 FORMAT('1',16X,'LINEAR TERMS',14X,'INTERACTION TERMS',6X,8
2147. XX,'SQUARED TERMS')
2148. WRITE(6,202)
2149. 202 FORMAT('1',16X,'VARIABLE',2X,'TYPE',2X,'WIDTH',2X,'HEIGHT',2X,'POSITHON',2X,'WIDTH',2X,'HEIGHT',2X,'POSITHON',2X)
2150. DO 210 I=1,NOZERO
2151. WRITE(6,223) I,TYP,I,J=1,6,COEFLJ),COEFWJ),COEPJ),COEFWHJ),COEFWPJ),
2152. 223 FORMAT('1',3X,I2,3X,IX,6A1,IX,1012X,FT.2,IX))
2153. 210 CONTINUE
2154. RETURN
2155. END
2156. SUBROUTINE START
2157. COMMON /PCONT/PCONT(50)
2158. COMMON /EIGCHK/EIGCHK
2159. COMMON /ISENS/ISENS
2160. COMMON /XTIME/XTIME
2161. COMMON /STDENS/STDENS
2162. COMMON /SADENS/SADENS
2163. COMMON /FREEZE/FREEZE
2164. COMMON /ALPHA/ALPHA
2165. COMMON /SANDW/SANDW
2166. COMMON /SANDL/SANDL
2167. COMMON /TSAND/TSAND
2168. COMMON /TAIR/TAIR
2169. COMMON /TSTEEL/TSTEEL
2170. COMMON /TINTFM/TINTFM
2171. COMMON /TINTFS/TINTFS
2172. COMMON /TCHILL/TCHILL
2173. COMMON /TRISER/TRISER
2174. COMMON /I1STP/I1STP
2175. COMMON /I2STP/I2STP
2176. COMMON /I3STP/I3STP
2177. COMMON /I4STP/I4STP
2178. COMMON /I5STP/I5STP
2179. COMMON /I6STP/I6STP
2180. COMMON /P1STP/P1STP
2181. COMMON /P2STP/P2STP
2182. COMMON /P3STP/P3STP
2183. COMMON /P4STP/P4STP
2184. COMMON /P5STP/P5STP
2185. COMMON /P6STP/P6STP
2186. COMMON /P7STP/P7STP
2187. COMMON /P8STP/P8STP
2188. COMMON /P9STP/P9STP
2189. COMMON /P10STP/P10STP
2190. COMMON /P11STP/P11STP
2191. COMMON /P12STP/P12STP
2192. COMMON /P13STP/P13STP
2193. COMMON /P14STP/P14STP
2194. COMMON /P15STP/P15STP
2195. COMMON /P16STP/P16STP
2196. COMMON /P17STP/P17STP
2197. COMMON /P18STP/P18STP
2198. COMMON /P19STP/P19STP
2199. COMMON /P20STP/P20STP

2149. WRITE(6,201)
2150. WRITE(6,202)
2151. WRITE(6,203)
2152. WRITE(6,204)
2153. WRITE(6,205)
2154. WRITE(6,206)
2155. WRITE(6,207)
2156. WRITE(6,208)
2157. WRITE(6,209)
2158. WRITE(6,210)
2159. WRITE(6,211)
2160. WRITE(6,212)
2161. WRITE(6,213)
2162. WRITE(6,214)
2163. WRITE(6,215)
2164. WRITE(6,216)
2165. WRITE(6,217)
2166. WRITE(6,218)
2167. WRITE(6,219)
2168. WRITE(6,220)
2169. WRITE(6,221)
2170. WRITE(6,222)
2171. WRITE(6,223)
2172. WRITE(6,224)
2173. WRITE(6,225)
2174. WRITE(6,226)
2175. WRITE(6,227)
2176. WRITE(6,228)
2177. WRITE(6,229)
2178. WRITE(6,230)
2179. WRITE(6,231)
2180. WRITE(6,232)
2181. WRITE(6,233)
2182. WRITE(6,234)
2183. WRITE(6,235)
2184. WRITE(6,236)
2185. WRITE(6,237)
2186. WRITE(6,238)
2187. WRITE(6,239)
2188. WRITE(6,240)
2189. WRITE(6,241)
2190. WRITE(6,242)
2191. WRITE(6,243)
2192. WRITE(6,244)
2193. WRITE(6,245)
2194. WRITE(6,246)
2195. WRITE(6,247)
2196. WRITE(6,248)
2197. WRITE(6,249)
2198. WRITE(6,250)
2199. WRITE(6,251)
COMMON /I3STP/I3STP
COMMON /I4STP/I4STP
COMMON /I5STP/I5STP
COMMON /SHKTCL/SHKTOL
COMMON /EIGCHK/EIGCHK
INTEGER * 2 EIGCHK
COMMON /COMSRI/COMSRI
INTEGER * 2 COMSRI
COMMON /COMSR2/COMSR2
INTEGER * 2 COMSR2
COMMON /OFPSSTP/OFPSSTP
COMMON /DGSTCP/DGSTOP
COMMON /DGOSTP/DGQSTP
COMMON /TINTGD/TINTGD
INTEGER * 2 TINTGD
COMMON /PENINT/PENINT
COMMON /PENMAX/PENMAX
COMMON /PENMLT/PENMLT
COMMON /DFCTEV/DFCTEV
INTEGER * 2 DFCTEV
COMMON /DGBND1/DGBND1
INTEGER * 2 DGBND1
COMMON /DGND2/DGBND2
INTEGER * 2 DGBND2
COMMON /SOLIT/SOLIT
INTEGER SOLIT
COMMON /IPRT/IPRT
COMMON /NEGCHK/NEGCHK
INTEGER * 2 NEGCHK
COMMON /DULTOL/DULTOL
COMMON /BENSTP/BENSTP
ISENS=3
XTIME=.00002778
XDIM=(1.0/ISENS)/12.0
STDENS=500.0
SADENS=105.0
FREEZE=2770.0
ALPHA=0.08
SANDL=2
TSAND=80.0
TAIR=50.0
TSTEEL=2800.0
TINF=2385.0
TINFS=2385.0
TCHILL=80.0
TRISER=2800.0
I1STP=20
I2STP=20
I3STP=30
I4STP=40
I5STP=500
SHKTCL=0.05
EIGCHK=1
COMSP1=1
CCMSR2=1
OFPSSTP=50.0
DGSTOP=1.0E-4
DOUSTP=5.0
TINTG0=1
PENINT=100.0
PENMAX=1.0E12
PENMT=10.0
DFCTEV=0
DGBND1=0
DGBND2=1
SOLIT=1000
IPRT=50
NEGCHK=1
HGTCHK=1
OUTTOL=0.10
BENSTP=20.0
RETURN

END

SUBROUTINE RANGE
COMMON /SENS/
SENS1=1.0
SENS2=SENS1/2.0
R(I)=SENS2
DO 300 I=1,ISENS
300 CONTINUE
RETURN
END

SUBROUTINE PCONTL
COMMON /PCONT/
INTEGER * 2 PCONT
DO 10 I=1,50
PCONT(I)=1
10 CONTINUE
C PCONT1 - CONTROLS PRINTING OF A GENERAL OUTPUT ROUTINE
C COEFFICIENTS, VARIABLES, ETC. - OUTGEN
PCONT(1)=1
C PCONT2 - CONTROLS PRINTING OF A LIST OF GENERAL CONSTANTS USED
C IN ALL SECTIONS OF THE PROGRAM - OUTCST
PCONT(2)=1
C PCONT3 - CONTROLS PRINTING OF BREAK POINTS FOR THE ROUND-OFF
C IN THE GEOMETRY SET-UP SECTION OF THE PROGRAM - OUTRND
PCONT(3)=1
C PCONT4 - CONTROLS PRINTING OF A LISTING OF THE PRINT CONTROL
C VALUES - OUTPCT
PCONT(4)=1
C PCONT5 - CONTROLS PRINTING OF THE OBJECTIVE FUNCTION IN A
C MATRIX FORM - OUTOMT
PCONT(5)=1
C PCONT6 - CONTROLS PRINTING OF THE MEANING OF THE TEMPERATURE
C DISTRIBUTION CODE - OUTTDD
PCONT(6)=1
C PCONT7 - CONTROLS PRINTING OF ALL POSSIBLE COMBINATIONS OF THE
C ZERO-ONE VARIABLES - OUTCM3
312.

PCCNT(7)=1
2321.
C
2322.
C PCONT8 - CONTROLS PRINTING OF ALL COMBINATIONS WITH AT LEAST
2323.
C ONE RISER (ALL CHILL COMBINATIONS) - OUTCF1
2324.
PCONT(8)=1
2325.
C
2326.
C PCONT9 - CONTROLS PRINTING OF ALL COMBINATIONS OF Zero-One
2327.
C VARIABLES AFTER THE FIRST SHRINKAGE RULE APPLIED - OUTCF2
2328.
PCONT(9)=1
2329.
C
2330.
C PCONT10 - CONTROLS THE PRINTING OF ALL COMBINATIONS OF Zero-One
2331.
C VARIABLES AFTER THE SECOND SHRINKAGE RULE APPLIED - OUTCF3
2332.
PCONT(10)=1
2333.
PCONT(20)=0
2334.
PCONT(19)=0
2335.
DO 63 I=1,50
2335.1 PCONT(I)=0
2335.2 63 CONTINUE
2335.3 PCONT(31)=1
2335.4 PCONT(32)=1
2335.5 PCONT(17)=1
2336.
RETURN
2337.
END
2338.
SUBROUTINE ZERO
2339. FCON=0.0
2340. FCON=0.0
2341.
RETURN
2342.
END
2343.
SUBROUTINE OBJSET
2344. COMMON /CQEFW/COEFWI(10)
2345. COMMON /CQEFH/COEFP(10)
2346. COMMON /CQEF/COEFP2(10)
2347. CCMMON /CCEFWP/CCEFWP(10)
2348. CCMMON /CCEFHP/CCEFHP(10)
2349. CCMMON /CCEF/COEFH(10)
2350. CCMMON /CCEF2/COEFH2(10)
2351. CCMMON /CCEF/COEF2(10)
2352. CCMMON /CCEF2/COEF2(10)
2353. CCMMON /CCEF/COEF(18,13)
2354. CCMMON /COEF/NOZERO
2355. CCMMON /ZVAL/ZVAL(10)
2356.
INTEGER = 2 ZVAL
2357. DO 1 I=1,18
2358. DO 2 J=1,18
2359. COEF(I,J)=0.0
2360.
2 CONTINUE
2361.
1 CONTINUE
2362.
DO 10 I=1,NOZERO
2363. J1=(I-1)*3+1
2364. J2=J1+1
2365. J3=J2+1
2366. I1=(I-1)*3+1
2367. I2=I1+1
2368. I3=I2+1
2369. COEF(I1,J1)=COEFW(I1)
2370. COEF(I2,J2)=COEFP2(I2)
2371. COEF(I3,J3)=COEFP2(I3)
2372. COEF(I1,J2)=COEFWH(I1)/2.0
2373. COEF(I2,J1)=COEFWH(I1)/2.0
SUBROUTINE EIGEN
COMMON /COEF/COEF(18,18)
DIMENSION A(18,18),D(18),Z(18,18),WK(400)
INTEGER N,JOBN,IZIER
N=18
IZ=18
JOBN=20
DO 10 I=1,18
DO 20 J=1,18
A(I,J)=COEF(I,J)
20 CONTINUE
10 CONTINUE
CALL EIGNSA(N,JOBN,D,Z,IZ,WK,IER)
WRITE(6,1)
1  FORMAT(•-','EIGEN VALUES FOR THE QUADRATIC PORTION OF THE OBJECTIVE FUNCTION ARE COMPUTED')
WRITE(6,2)
2  FORMAT(•  , 'THE EIGEN VALUES ARE AS LISTED BELOW')
DO 30 I=1,18
30 FORMAT( •  •, ' EIGEN VALUE NUMBER',13,' = ',F15.8)
30 CONTINUE
DO 100 I=1,18
IF(D(I).LT.0.0) GO TO 150
100 CONTINUE
WRITE(6,200)
200 FORMAT( •  'AS SHOWN ABOVE ALL EIGEN VALUES ARE POSITIVE - T
XHUS THE QUADRATIC MATRIX IS POSITIVE DEFINITE')
WRITE(6,201)
201 FORMAT( ' ', ' POSITIVE DEFINITE IMPLIES CONVEXITY')
GO TO 500
150 WRITE(6,301)
301 FORMAT( 'AS SHOWN ABOVE AT LEAST ONE OF THE EIGEN VALUES
X IS NEGATIVE WHICH ELIMINATES POSITIVE DEFINITENESS')
WRITE(6,302)
302 FORMAT( ' ', ' THIS IMPLIES LACK OF CONVEXITY')
500 RETURN
SUBROUTINE TDIST
COMMON /LTEMP/LTEMP(30)
COMMON /I1STP/I1STP
COMMON /I2STP/I2STP
COMMON /I3STP/I3STP
COMMON /I4STP/I4STP
COMMON /I5STP/I5STP
COMMON /TSTEEL/TSTEEL
LTEMP(I)=TSTEEL
DO 2 I=1,30
1 CONTINUE
DO 2 I=1,11
2434. LTEMP(I) = LTEMP(I-1) - 12STP
2435. 2 CONTINUE
2436. DO 3 I = 12, 16
2437. LTEMP(I) = LTEMP(I-1) - 13STP
2438. 3 CONTINUE
2439. DO 4 I = 17, 21
2440. LTEMP(I) = LTEMP(I-1) - 14STP
2441. 4 CONTINUE
2442. DO 5 I = 22, 26
2443. LTEMP(I) = LTEMP(I-1) - 15STP
2444. 5 CONTINUE
2445. LTEMP(27) = 0
2446. RETURN
2447. END

SUBROUTINE OUTCNT
2448. COMMON /ISENS/ ISENS
2449. COMMON /XTIME/ XTIME
2450. COMMON /XDIM/ XDIM
2451. COMMON /STDENS/ STDENS
2452. COMMON /SADENS/ SADENS
2453. COMMON /FREEZE/ FREEZE
2454. COMMON /ALPHA/ ALPHA
2455. COMMON /SANDW/ SANDW
2456. INTEGER SANDW
2457. COMMON /SANDL/ SANDL
2458. INTEGER SANDL
2459. COMMON /TSAND/ TSAND
2460. COMMON /TAIR/ TAIR
2461. COMMON /TSADW/ TSADW
2462. COMMON /TSADL/ TSADL
2463. COMMON /TSTEEL/ TSTEEL
2464. COMMON /TINTFM/ TINTFM
2465. COMMON /TINTFS/ TINTFS
2466. COMMON /TRISER/ TRISER
2467. COMMON /I1STP/ I1STP
2468. COMMON /I2STP/ I2STP
2469. COMMON /I3STP/ I3STP
2470. COMMON /I4STP/ I4STP
2471. COMMON /I5STP/ I5STP
2472. COMMON /SHKTOL/ SHKTOL
2473. COMMON /EIGCHK/ EIGCHK
2474. INTEGER * 2 EIGCHK
2475. COMMON /COMSR1/ COMSR1
2476. INTEGER * 2 COMSR1
2477. COMMON /COMSR2/ COMSR2
2478. INTEGER * 2 COMSR2
2479. WRITE(6,1)
2480. 1 FORMAT(*',15X,'GENERAL CONSTANT AND PROGRAM CONTROL INFORMATION')
2481. WRITE(6,2)
2482. 2 FORMAT('THE FOLLOWING CONSTANTS ARE USED IN THE SOLIDIFICATION SECTION OF THE PROGRAM, THEIR VALUES ARE:')
2483. WRITE(6,3)
2484. 3 FORMAT('ISENS - NUMBER OF NODES PER INCJ = ',14)
2485. WRITE(6,4) ISENS
2486. 4 FORMAT('XTIME - TIME BETWEEN EACH SOLIDIFICATION ITERATION = ',F15.10)
2487. WRITE(6,5) XTIME
2488. 5 FORMAT('XDIM - DISTANCE BETWEEN EACH NODE = ',F15.10)
2489. WRITE(6,6) XDIM
*STDEN$ - STEEL DENSITY (POUNDS PER CUBIC FOOT) = X'F15.8)
*SADENS - SAND DENSITY (POUNDS PER CUBIC FOOT) = X'F15.8)
*FREEZE - FREEZING TEMPERATURE OF STEEL (DEGREE F) = X'F10.3)
*ALPHA - SHRINKAGE CONSTANT (PERCENTAGE) = X'F10.5)
*SANDW - BARRIER OF SAND ON TOP AND BOTTOM OF CASTING (INCHES) = X'F15)
*SANDL - BARRIER OF SAND ON BOTH SIDES OF CASTING (INCHES) = X'F15)
*THE FOLLOWING ARE INITIAL TEMPERATURES (ALL IN DEGREES F):
*TAIR - TEMPERATURE OF SURROUNDING AIR = X'F15.8)
*TSAND - TEMPERATURE OF THE SAND = X'F15.8)
*TSTEE1 - TEMPERATURE OF THE STEEL = X'F15.8)
*TRISER - TEMPERATURE OF THE RISER = X'F15.8)
*TCHILL - TEMPERATURE OF THE CHILL = X'F15.8)
*X INTERFACE = X'F15.5)
*SHKTOL - SHRINKAGE TOLERANCE TO BE CONSIDERED SHRINKAGE FREE (CUBIC INCHES) = X'F15.8)
*THE FOLLOWING VARIABLES DICTATE THE CONTROL OF THE PROGRAM:
*EIGCHK

315
SUBROUTINE OUTRND

COMMON /ISENS/ISENS
COMMON /R/R(5)

WRITE(6,1)
1 FORMAT(•,4X,'THE FOLLOWING INFORMATION GIVES NODE REPRESENTATION S
XZE')

WRITE(6,2) ISENS
2 FORMAT('.5X,ISENS IT IV ITY INODES PER INCH) = ',15)
X=1.0/ISENS/12.0
WRITE(6,3) X
3 FORMAT('.5X,Distance BETWEEN NODES = ',F.12.7)
WRITE(6,4)
4 FORMAT(•,5X,'ROUND - OFFS OF VARIABLE BETWEEN NODES ARE GIVEN')
DO 10 I=1,ISENS
10 WRITE(6,11) I,R(I)
CONTINUE
RETURN

SUBROUTINE OUTPCT

COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT

WRITE(6,100)
100 FORMAT(•,55X,'PRINT CONTROL VALUES')
WRITE(6,101)
101 FORMAT(•,4X,1)
WRITE(6,1)
1 FORMAT('.5X,'PCONT1 = ',12,' CONTROLS PRINTING OF OVERALL GENERAL
XINPUT - OUTGEN')
WRITE(6,2) PCONT(1)
2 FORMAT('.5X,'PCONT2 = ',12,' CONTROLS PRINTING OF THE LIST OF CON
XTANTS USED IN THE PROGRAM - OUTGEN')
WRITE(6,3) PCONT(2)
3 FORMAT('.5X,'PCONT3 = ',12,' CONTROLS THE PRINTING OF THE BREAK PC
XINTS FOR VARIABLE ROUND-OFF - OUTRND')
WRITE(6,4) PCONT(3)
4 FORMAT('.5X,'PCONT4 = ',12,' CONTROLS PRINTING OF PRINT CONTROL VA
SUBROUTINE OUTOMT

COMMON /COEF/COEF(18,18)
COMMON /NOZERO/NOZERO
COMMON /COEFW/COEFW(10)
COMMON /COEFH/COEFH(10)
COMMON /COEFP/COEFP(10)

I1=1
I2=NOZERO*3
J1=1
J2=NOZERO*3
IP=NOZERO*3

WRITE(6,1)
FORMAT(1',46X,'THE OBJECTIVE FUNCTION IN MATRIX FORM')
WRITE(6,100)
FORMAT(1',LINEAR PORTION')
WRITE(6,101)
FORMAT(1',COEFW')
WRITE(6,102)
FORMAT(1',COEFH')
WRITE(6,103)
FORMAT(1',COEFP')

120 CONTINUE
WRITE(6,2)
FORMAT(1',QUADRATIC PORTION')
WRITE(6,20)
FORMAT(1',DIMENSION MATRIX')
WRITE(6,3)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,4)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,5)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,6)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,7)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,8)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,9)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,10)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,11)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,12)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,13)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,14)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,15)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,16)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,17)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,18)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,19)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,20)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,21)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,22)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,23)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,24)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,25)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,26)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,27)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,28)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,29)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,30)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,31)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,32)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,33)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,34)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,35)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,36)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,37)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,38)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,39)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,40)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,41)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,42)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,43)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,44)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,45)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,46)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,47)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,48)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,49)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,50)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,51)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,52)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,53)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,54)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,55)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,56)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,57)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,58)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,59)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,60)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,61)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,62)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,63)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,64)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,65)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,66)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,67)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,68)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,69)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'
WRITE(6,70)
FORMAT(1',XOS, WID, HGT', POS, WID, HGT', POS, WID, HGT', P'}
WRITE(6,5)
5 FORMAT(' ',i,i,i)
DO 10 I=1,NOZERO
11=I-1*3+1
I2=I+1
I3=I+2
WRITE(6,11) (COEF(I1,J),J=1,18)
11 FORMAT(' ',WID,,2X,18F6.1)
WRITE(6,12) I1(COEF(I2,J),J=1,18)
12 FORMAT(' ',HGT,,11,IX,18F6.1)
WRITE(6,13) (COEF(I3,J),J=1,13)
13 FORMAT(' ',PGS,,2X,18F6.1)
10 CONTINUE
RETURN
END
SUBROUTINE OUTTD
COMMON /ALP/ALPI27)
LOGICAL*1 ALP
COMMON /LTEMP/LTEMPt30)
WRITE(6,11)
11 FORMAT(I6,2X,'THE FOLLOWING IS THE TEMPERATURE REPRESENTATION OF X THE SOLIDIFICATION CODE')
WRITE(6,2)
2 FORMAT(I6,2X,15X,ALP,3X,'IMPLIES TEMPERATURE RANGE OF ',15,' TO ',15)
10 CONTINUE
RETURN
END
SUBROUTINE COMBO
COMMON /PCONT/PCONT1501
INTEGER*2 PCONT
COMMON /C0MSR1/C0MSR1
INTEGER*2 C0MSR1
COMMON /C0MSR2/C0MSR2
INTEGER*2 C0MSR2
CALL SETCMB
IF(PCONT17).EQ.1 CALL OUTCM1
CALL CMDFE1
IF(PCONT18).EQ.1 CALL OUTF1
CALL CMFBE1
IF(C0MSR1.EQ.1) CALL CMFBE2
IF(C0MSR1.EQ.1.AND.PCONT19).EQ.1) CALL OUTF2
IF(C0MSR2.EQ.1) CALL CMFBE3
IF(C0MSR2.EQ.1.AND.PCONT19).EQ.1) CALL OUTF3
10 CONTINUE
RETURN
END
SUBROUTINE SETCMB
COMMON /COMB/COMBUOO ,6)
INTEGER*2 COMB
COMMON /NQZERO/NQZERO
COMMON /NUMCMB/NUMCMB
DO 21=1,100
DO 3 J=1,6
COMBI,(J)=0
3 CONTINUE
10 CONTINUE
ICT=NOZERO
2721. IF(NOZERO.LT.2) GO TO 1000
2722. INUM=NOZERO-1
2723. DO 100 I=1,INUM
2724. JNUM=NOZERO-1
2725. DO 110 J=1,JNUM
2726. ICT=ICT+1
2727. J1=I+J
2728. COMBIICT,I) )=1
2729. COMBIICT,J1)=1
2730. 110 CONTINUE
2731. 100 CONTINUE
2732. IF!NOZERO.LT.3) GO TO 1000
2733. INUM=NOZERO-2
2734. DO 200 I=1,INUM
2735. JS=INUM-I+1
2736. DO 210 J=1,JS
2737. JNUM=JS-J+1
2738. DO 220 K=1,JNUM
2739. ICT=ICT+1
2740. J1=I+J
2741. K1=J1+K
2742. COMBIICT,I) )=1
2743. COMBIICT,J1)=1
2744. COMBIICT,K1)=1
2745. 220 CONTINUE
2746. 210 CONTINUE
2747. 200 CONTINUE
2748. IF(NOZERO.EQ.4) GO TO 700
2749. IF(NOZERO.EQ.5) GO TO 660
2750. IF(NOZERO.LT.4) GO TO 1000
2751. IR=ICT
2752. INUM=NOZERO-1
2753. DO 300 I=1,INUM
2754. JNUM=NOZERO-1
2755. DO 310 J=1,JNUM
2756. ICT=ICT+1
2757. J1=I+J
2758. COMBIICT,I) )=1
2759. COMBIICT,J1)=1
2760. 310 CONTINUE
2761. 300 CONTINUE
2762. IS=ICT
2763. IR=IR+1
2764. DO 320 I=IR,IS
2765. DO 330 J=1,5
2766. IF(COMBI,J).EQ.0) GO TO 325
2767. COMBI,J)=0
2768. GO TO 330
2769. 325 COMBI,J)=1
2770. 330 CONTINUE
2771. 320 CONTINUE
2772. IR=ICT
2773. DO 400 I=1,NOZERO
2774. ICT=ICT+1
2775. COMBIICT,I) )=1
2776. 400 CONTINUE
2777. IS=ICT
2778. IR=IR+1
2779. DO 420 I=IR,IS
DO 430 J=1,6
2781. IF(COMB(I,J).EQ.0) GO TO 425
2782. COMB(I,J)=0
2783. GO TO 430
2784. 425 COMB(I,J)=1
2785. 430 CONTINUE
2786. 420 CONTINUE
2787. ICT=ICT+1
2788. DO 510 J=1,6
2789. COMB(ICT,J)=1
2790. 510 CONTINUE
2791. GO TO 1000
2792. 700 ICT=ICT+1
2793. DO 710 I=1,4
2794. COMB(ICT,I)=1
2795. 710 CONTINUE
2796. GO TO 1000
2797. 300 IS=ICT
2798. DO 810 I=1,NOZERO
2799. ICT=ICT+1
2800. COMB(ICT,I)=1
2801. 810 CONTINUE
2802. IR=ICT
2803. IS=IS+1
2804. DO 830 I=IS,IR
2805. DO 832 J=1,5
2806. IF(COMB(I,J).EQ.0) GO TO 835
2807. COMB(I,J)=0
2808. GO TO 832
2809. 835 COMB(I,J)=1
2810. 832 CONTINUE
2811. 830 CONTINUE
2812. ICT=ICT+1
2813. DO 840 I=1,5
2814. COMB(ICT,I)=1
2815. 840 CONTINUE
2816. 1000 NUMCMB=ICT
2817. RETURN
2818. END
2819. SUBROUTINE CMBFE1
2820. COMMON /NOZERO/NOZERO
2821. COMMON /COMB/COMB(100,6)
2822. INTEGER*2 COMB
2823. COMMON /NUMCMB/NUMCMB
2824. COMMON /COMUSE/COMUSE(100)
2825. INTEGER*2 COMUSE
2826. COMMON /ZTYPE/ZTYPE(10)
2827. INTEGER ZTYPE
2828. C COMUSE = 1 IMPLIES FEASIBLE
2829. C COMUSE = 0 IMPLIES INFEASIBLE
2830. C CHECK FOR ALL VARIABLES OF A COMBINATION BEING CHILLS
2831. DO 10 I=1,NUMCMB
2832. USE=1
2833. DO 20 J=1,NOZERO
2834. IF(COMB(I,J).EQ.0) GO TO 20
2835. IF(ZTYPE(J).EQ.4) GO TO 30
2836. 20 CONTINUE
2837. COMUSE(I)=0
2838. GO TO 10
2839. 30 COMUSE(I)=1
SUBROUTINE CMBFE2
COMMON /SHRINK/SHRINK
COMMON /NOZERO/NOZERO
COMMON /ZTYPE/ZTYPE10)
INTEGER ZTYPE
COMMON /ZVAL/ZVAL(10)
INTEGER ZVAL
COMMON /SHKTOL/SHKTOL
COMMON /COMB/COMB(100,6)
INTEGER COMB
COMMON /NUMCMB/NUMCMB
COMMON /COMUSE/COMUSE1100)
INTEGER COMUSE
DO 111 1=1,NOZERO
IF(ZTYPE(I).EQ.3) GO TO 100
ZVAL(I)=I
DO 101 J=1,NOZERO
IF(ZTYPE(I).NE.3) ZVAL(I)=1
101 CONTINUE
CALL MAXVAL
CALL TEST
IF(SHRINK.LT.SHKTOL) GO TO 100
ICT=I
DO 200 K=1,NUMCMB
IF(COMB(K,ICT).NE.1) GO TO 200
DO 210 L=1,NOZERO
IF(ZTYPE(L).EQ.3) GO TO 210
IF(COMB(K,L).EQ.0) GO TO 210
IF(L.EQ.ICT) GO TO 210
210 CONTINUE
COMUSEIK)=0
200 CONTINUE
100 10 1=1,NOZERO
ZVAL(I)=0
10 CONTINUE
RETURN
END
SUBROUTINE CLERC8
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /NOZERO/NOZERO
DO 10 I=1,NOZERO
WID(I)=0.0
POS(I)=0.0
HEIGHT(I)=0.0
10 CONTINUE
RETURN
END
SUBROUTINE CMBFE3
COMMON /SHRINK/SHRINK
COMMON /NOZERO/NOZERO
COMMON /ZTYPE/ZTYPE10)
INTEGER ZTYPE
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
COMMON /SHKTOL/SHKTOL
COMMON /COMB/COMB(100,6)
INTEGER * 2 COMB
COMMON /NUMCMB/NUMCMB
COMMON /COMUSE/COMUSE(100)
INTEGER * 2 COMUSE
COMMON /ITRK/ITRK(10)
INTEGER * 2 ITRK
DO 1 I=1,10
ITRK(1)=0
1 CONTINUE
IRCT=0
DO 10 I=1,NOZERO
IF(ZTYPE(I).EQ.3) GO TO 10
IRCT=IRCT+1
ITRK(IRCT)=I
10 CONTINUE
ICMB=IRCT-1
DO 100 I=1,ICMB
J1=J1+1
J2=IRCT
DO 110 J=J1,J2
ZVAL(ITRK(I))=1
ZVAL(ITRK(J))=1
110 CONTINUE
CALL MAXVAL
CALL TEST
IF(SHRINK.LT.SHKTOL) GO TO 150
150 CONTINUE
U = 1, NOZERO
ZVAL(1)=0
160 CONTINUE
100 CONTINUE
RETURN
END
SUBROUTINE OUTCMB
COMMON /NUMCMB/NUMCMB
COMMON /COMB/COMB(100,6)
INTEGER * 2 COMB
COMMON /NOZERO/NOZERO
COMMON /TYP/TYP(10,6)
LOGICAL * 1 TYP
WRITE(6,1)
1 FORMAT(*1',28X,'COMBINATIONS OF ZERO-ONE VARIABLES ALL POSSIBILITIES
XES ARE LISTED BELOW*)
WRITE(6,2)
BEGIN F77

2 FORMAT('VARIA\'BLE TYPES')
2 WRITE(6,10) (TY\(P(I,J),J=1,6),I=1,NOZERO)
10 FORMAT('TYPE',60A1)
DO 100 I=1,NUMCMB
WRITE(6,101) (COMB(I,J),J=1,NOZERO)
101 FORMAT(' ',10(I3,3X))
100 CONTINUE
RETURN
END
SUBROUTINE QUTCF1
COMMON /NUMCMB/NUMCMB
COMMON /COMB/COMB(100,6)
INTEGER*2 COMB
COMMON /COHUSE/COMUSE(100)
INTEGER*2 COMUSE
COMMON /NOZERO/NOZERO
COMMON /TYP/TYP(10,6)
LOGICAL * 1 TYP
WRITE(6,1)
1 FORMAT('THE FOLLOWING ARE ALL POSSIBLE COMBINATIONS OF ZE
XERO-ONE VARIABLES WITH AT LEAST ONE RISER')
WRITE(6,2)
2 FORMAT('YPES OF VARIABLES')
WRITE(6,3) (TYP(I,J),J=1,6),I=1,NOZERO)
3 FORMAT(' ',60A1)
DO ICO 1=1,NUMCMB
IF(COMUSE(I).EQ.0) GO TO 100
WRITE(6,101) (COMB(I,J),J=1,NOZERO)
101 FORMAT(' ',10(I3,3X))
100 CONTINUE
RETURN
END
SUBROUTINE QUTCF2
COMMON /COMB/COMB(100,6)
INTEGER*2 COMB
COMMON /NUMCMB/NUMCMB
COMMON /ZTYPE/ZTYPE(10)
INTEGER ZTYPE
COMMON /NOZERO/NOZERO
COMMON /CCMUSE/COMUSE(100)
INTEGER*2 COMUSE
COMMON /TYP/TYP(10,6)
LOGICAL * 1 TYP
WRITE(6,1)
1 FORMAT('ALL POSSIBLE COMBINATIONS OF ZERO-ONE VARIABLES AF
XTER SOLIDIFICATION RULE NUMBER ONE APPLICATION')
WRITE(6,2)
2 FORMAT('YPES OF VARIABLES')
WRITE(6,3) (TYP(I,J),J=1,6),I=1,NOZERO)
3 FORMAT(' ',60A1)
DO ICO 1=1,NUMCMB
IF(COMUSE(I).EQ.0) GO TO 100
WRITE(6,101) (COMB(I,J),J=1,NOZERO)
101 FORMAT(' ',10(I3,3X))
100 CONTINUE
RETURN
END
SUBROUTINE QUTCF3
COMMON /COMB/COMB(100,6)
INTEGER*2 COMB
END F77
COMMON /NUMCMB/NUMCMB
COMMON /ZTYPE/ZTYPE(10)
INTEGER ZTYPE
COMMON /NOZERO/NOZERO
COMMON /COMUSE/COMUSE(100)
INTEGER * 2 COMUSE
COMMON /TYP/TYP(10,6)
LOGICAL * 1 TYP
WRITE(6,1)
1 FORMAT('1'X,'ALL POSSIBLE COMBINATIONS OF ZERO-ONE VARIABLES AF
XER SOLIDIFICATION RULE NUMBER TWO APPLICATION')
WRITE(6,2)
2 FORMAT('1'X,'VARIABLE TYPES')
WRITE(6,3) ((TYP(I,J),J=1,NOZERO),I=1,NOZERO)
3 FORMAT('1'X,'60AL')
IF (COMUSE(I).EQ.0) GO TO 100
WRITE(6,101) (COMB(I,J),J=1,NOZERO)
101 FORMAT('1',10(13.3X))
100 CONTINUE
RETURN
END
SUBROUTINE TEQUAN
COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT
CALL TSGRAD
IF (PCONT(10).EQ.1) CALL OUTTSG
CALL TINTS
IF (PCONT(11).EQ.1) CALL OUTTIS
RETURN
END
SUBROUTINE TSGRAD
COMMON /TINTGD/TINTGD
INTEGER * 2 TINTGD
COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT
COMMON /TSGDW/TSGDW(10)
COMMON /TSGDH/TSGDH(10)
COMMON /TSGDP/TSGDP(10)
COMMON /NOZERO/NOZERO
COMMON /ZVAL/ZVAL(10)
COMMON /SHRINK/SHRINK
COMMON /SHKOMP/SHKOMP
COMMON /VAL/VAL
IF (TINTGD.EQ.0) GO TO 100
DO 1 I=1,10
TSGDW(I)=0.0
TSGDH(I)=0.0
TSGDP(I)=0.0
1 CONTINUE
DO 2 I=1,NOZERO
ZVAL(I)=1
2 CONTINUE
CALL AVGVAL
CALL TEST
SHKOMP=SHRINK
DO 10 I=1,NOZERO
CALL TSGRAW(I)
TSGDW(I)=VAL
CALL TSGRAH(I)
TSGDH(I)=VAL
CALL TSGRAP(I)
TSGDP(I)=VAL
10 CONTINUE
100 RETURN
SUBROUTINE TSGRAH(I)
COMMON /WID/WID(I)
COMMON /POS/POS(I)
COMMON /SHKCMP/SHKCMP
COMMON /SHRINK/SHRINK
COMMON /VAL/VAL
COMMON /TINTGF/TINTGF
INTEGER * 2 TINTGF
COMMON /RECLEN/RECLEN(I)
INTEGER RECLEN
COMMON /RECWID/RECWID(I)
INTEGER RECWID
COMMON /ZSIDE/ZSIDE(I)
INTEGER ZSIDE
COMMON /ZREC/ZREC(I)
INTEGER ZREC
REMB1=WID(I)
REMB2=POS(I)
SLACK1=(RECWID(ZREC(I))-WID(I)-POS(I))
SLACK2=SLACK1+POS(I)
IF(ZSIDE(I)=1 OR ZSIDE(I)=4) SLACK1=(RECLEN(ZREC(I))-WID(I)-
XPOS(I))
IF(SLACK1<1.0) GO TO 50
WID(I)=WID(I)+1.0
CALL TEST
TOP=SHKCMP-SHRINK
BOT=REMB1-WID(I)
VAL=TOP/BOT
GO TO 100
50 SLACK2=SLACK1+POS(I)
IF(SLACK2=1.0) GO TO 70
WID(I)=WID(I)+1.0
POS(I)=POS(I)-(1.0-SLACK1)
IF(POS(I)<0.0) POS(I)=0.0
CALL TEST
3121. TOP=SHKCMP-SHRINK
3122. BOT=REMB1-WID(I)
3123. VAL=TOP/BOT
3124. GO TO 100
3125. 70 WID(I)=SLACK1+WID(I)+POS(I)
3126. POS(I)=0.0
3127. CALL TEST
3128. TOP=SHKCMP-SHRINK
3129. BOT=REMB1+WID(I)
3130. VAL=TOP/BOT
3131. 100 IF(TINTGF.EQ.0) GO TO 300
3132. WID(I)=REMB1
3133. POS(I)=REMB2
3134. IF(SLACK1.LT.2.0) GO TO 150
3135. WID(I)=WID(I)+2.0
3136. CALL TEST
3137. TOP=SHKCMP-SHRINK
3138. BOT=REMB1+WID(I)
3139. TSGDW2=TOP/BOT
3140. GO TO 200
3141. 150 SLACK2=SLACK1+POS(I)
3142. IF(SLACK2.LT.2.0) GO TO 170
3143. WID(I)=WID(I)+2.0
3144. POS(I)=POS(I)-(2.0-SLACK1)
3145. IF(POS(I).LT.0.0) POS(I)=0.0
3146. CALL TEST
3147. TOP=SHKCMP-SHRINK
3148. BOT=REMB1+WID(I)
3149. TSGDW2=TOP/BOT
3150. GO TO 200
3151. 170 WID(I)=SLACK1+WID(I)+POS(I)
3152. POS(I)=0.0
3153. CALL TEST
3154. TOP=SHKCMP-SHRINK
3155. BOT=REMB1+WID(I)
3156. TSGDW2=TOP/BOT.
3157. 200 IF(TSGDW2.LT.VAL) VAL=TSGDW2
3158. 300 WID(I)=REMB1
3159. POS(I)=REMB2
3160. RETURN
3161. END
3162. SUBROUTINE TSGRAPCI)
3163. COMMON /POS/POS10)
3164. COMMON /WID/WID10)
3165. COMMON /SHKCMP/SHKCMP
3166. COMMON /SHRINK/SHRINK
3167. COMMON /VAL/VAL
3168. COMMON /TINTGF/TINTGF
3169. INTEGER * 2 TINTGF
3170. COMMON /RECLEN/RECLEN(20)
3171. INTEGER RECLEN
3172. COMMON /RECWD/RECWD(20)
3173. INTEGER RECWD
3174. COMMON /ZSIDE/ZSIDE(10)
3175. INTEGER ZSIDE
3176. COMMON /ZREC/ZREC(10)
3177. INTEGER ZREC
3178. REMB1=POS(I)
3179. REMB2=WID(I)
3180. SLACK1=(RECWIDTH(REC(I))−WID(I)−POS(I))
3181. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) SLACK1=(RECLEN(REC(I))−WID(I)−
3182. XPOS(I))
3183. IF(SLACK1.LT.1.0) GO TO 50
3184. POS(I)=POS(I)+1.0
3185. CALL TEST
3186. TQP=SHKCMP−SHRINK
3187. BOT=REMB1−POS(I)
3188. VAL=TOP/BOT
3189. GO TO 100
3190. 50 SLACK2=SLACK1+WID(I)
3191. IF(SLACK2.LT.1.0) GO TO 70
3192. POS(I)=POS(I)+1.0
3193. WID(I)=WID(I)−(1.0−SLACK1)
3194. IF(WID(I).LT.0.0) WID(I)=0.0
3195. CALL TEST
3196. TQP=SHKCMP−SHRINK
3197. BOT=REMB1−POS(I)
3198. VAL=TOP/BOT
3199. GO TO 200
3200. 70 POS(I)=SLACK1+WID(I)+POS(I)
3201. WID(I)=0.0
3202. CALL TEST
3203. TQP=SHKCMP−SHRINK
3204. BOT=REMB1−POS(I)
3205. VAL=TOP/BOT
3206. 100 IF(INTGF.EQ.0) GO TO 300
3207. POS(I)=REMB1
3208. WID(I)=REMB2
3209. IF(SLACK2.LT.2.0) GO TO 150
3210. POS(I)=POS(I)+2.0
3211. CALL TEST
3212. TQP=SHKCMP−SHRINK
3213. BOT=REMB1−POS(I)
3214. TSGDP2=TOP/BOT
3215. GO TO 200
3216. 150 SLACK2=SLACK1+WID(I)
3217. IF(SLACK2.LT.2.0) GO TO 170
3218. POS(I)=POS(I)+2.0
3219. WID(I)=WID(I)−(2.0−SLACK1)
3220. IF(WID(I).LT.0.0) WID(I)=0.0
3221. CALL TEST
3222. TQP=SHKCMP−SHRINK
3223. BOT=REMB1−POS(I)
3224. TSGDP2=TOP/BOT
3225. GO TO 200
3226. 170 POS(I)=SLACK1+WID(I)+POS(I)
3227. WID(I)=0.0
3228. CALL TEST
3229. TQP=SHKCMP−SHRINK
3230. BOT=REMB1−POS(I)
3231. TSGDP2=TOP/BOT
3232. 200 IF(TSGDP2.LT.VAL) VAL=TSGDP2
3233. 300 POS(I)=REMB1
3234. WID(I)=REMB2
3235. RETURN
3236. END
3237. SUBROUTINE TSGRAH(I)
3238. COMMON /HEIGHT/HEIGHT(10)
3239. COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /VAL/VAL
COMMON /TINTGF/TINTGF
INTEGER * 2 TINTGF
COMMON /SHRINK/SHRINK
COMMON /SHKCMP/SHKCMP
INTEGER * 2 TINTGF
EXTERNAL IFLAG
COMMON /SHRINK/SHRINK
COMMON /SHKCMP/SHKCMP
IFLAG=0
REMB1=HEIGHT(I)
SLACK1=ZBOUND(I)-HEIGHT(I)
IF(SLACK1.LT.1.0) GO TO 50
HEIGHT(I)=HEIGHT(I)+1.0
CALL TEST
TOp=SHKCMP-SHRINK
BOT=REMB1-HEIGHT(I)
VAL=TOP/BOT
GO TO 100
50 HEIGHT(I)=ZBOUND(I)
CALL TEST
TOP=SHKCMP-SHRINK
BOT=REMB1-HEIGHT(I)
VAL=TOP/BOT
GO TO 200
100 IF(TINTGF.EQ.0) GO TO 300
IF(SLACK1.LT.2.0) GO TO 150
HEIGHT(I)=HEIGHT(I)+2.0
CALL TEST
TOP=SHKCMP-SHRINK
BOT=REMB1-HEIGHT(I)
TSGDH2=TOP/BOT
GO TO 200
150 IF(SLACK1.LT.1.0) IFLAG=1
HEIGHT(I)=ZBOUND(I)
CALL TEST
TOP=SHKCMP-SHRINK
BOT=REMB1-HEIGHT(I)
TSGDH2=TOP/BOT
GO TO 200
200 IF(TSGDH2.LT.VAL) VAL=TSGDH2
300 HEIGHT(I)=REMB1
RETURN
END
SUBROUTINE OUTTSG
COMMON /TSGDW/TSGDW(10)
COMMON /TSGDH/TSGDH(10)
COMMON /TSGDP/TSGDP(10)
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
COMMON /TINTGD/TINTGD
INTEGER * 2 TINTGD
COMMON /TYP/TYP(10,6)
LOGICAL * 1 TYP
WRITE(6,1)
1 FORMAT(1X,1H1,RESULTS OF INITIAL SHRINKAGE GRADIENT EVALUATION')
WRITE(6,2)
2 FORMAT(1X,1H2,IN THIS TEST ALL ZERO-ONE VARIABLES HAD A VALUE OF ONE
X, THE CORRESPONDING WIDTH, HEIGHT AND POSITION WERE SET AT')
WRITE(6,3)
3 FORMAT(1X,1H3,THEIR AVERAGE VALUES. THE AVERAGES ARE SET WITH RESPE
XCT TO THEIR OWN INDIVIDUAL BOUNDS. EACH VARIABLE WAS VARIED')
WRITE(6,4)
4 FORMAT(1X,1H4,SLIGHTLY TO NUMERICALLY DETERMINE ALL PARTIAL DERIVATI
XVES FOR EACH INDEPENDENT VARIABLE

IF(ITINTGD.EQ.1) WRITE(6,10)
10 FORMAT(0,10X,'IN THIS CASE, A MORE EXTENSIVE GRADIENT CALCULATION')
WRITE(6,20)
20 FORMAT(0,'THE RESULTS FOLLOW')
WRITE(6,100)
100 FORMAT(0,25X,14X,'SHRINKAGE GRADIENTS')
WRITE(6,101)
101 FORMAT(0,'VARIABLE NUMBER',3X,'TYPE',3X,5X,'WIDTH ',5X,5X,'HEIGHT ')
WRITE(6,102)
102 FORMAT(0,'POSITION',4X)
DO 200 I=1,NOZERO
WRITE(6,201)
201 FORMAT(0,6X,12,7X,2X,6A1,2X,13X,F12.3))
200 CONTINUE
RETURN
END

SUBROUTINE TINTSG
COMMON /ISENS/ISENS
COMMON /GET/GET
COMMON /R/R(5)
COMMON /ISEND/ISEND
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /ZVAL/ZVAL(10)
INTEGER ZVAL
COMMON /NOZERO/NOZERO
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /RECWID/RECWID(20)
INTEGER RECWID
COMMON /POS/POS(10)
COMMON /WID/WID(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZTYPE/ZTYPE(10)
INTEGER ZTYPE
COMMON /SHRINK/SHRINK
COMMON /WRSTSK/WRSTSK
DO 10 I=1,100
X=I
GET=0.1*(X)
CALL ROUND
IF(ISEND.EQ.1) X1=GET
IF(ISEND.EQ.2) X2=GET
IF(ISEND.EQ.3) X3=GET
IF(ISEND.EQ.3) GO TO 20
10 CONTINUE
20 DO 90 I=1,10
ZVAL(I)=0.0
WID(I)=0.0
HEIGHT(I)=0.0
POS(I)=0.0
90 CONTINUE
DO 100 I=1,NOZERO
IF(ZTYPE(I).EQ.3) GO TO 100
ZVAL(I)=1
3360. \[ \text{WID}(i) = x3 \]
3361. \[ \text{HEIGHT}(i) = x2 \]
3362. \[ x = \text{WID}(z) \cdot \text{RECW}(i) \]
3363. \[ \text{IF}(\text{ZSIDE}(i) = \text{EQ.} 1.0 \text{ OR } \text{ZSIDE}(i) = \text{EQ.} 4) x = \text{RECl}(z) \]
3364. \[ \text{POS}(i) = (x - x3) / 2.0 \]
3365. 100 \text{ CONTINUE}
3366. \text{CALL TEST}
3367. \text{WRSTSK} = \text{SHRINK}
3368. \text{RETURN}
3369. \text{END}
3370. \text{SUBROUTINE OUTTIS}
3371. \text{COMMON} / \text{WRSTSK} / \text{WRSTSK}
3372. \text{WRITE}(6,11)
3373. 11 \text{ FORMAT(}-*, ',' )
3374. \text{WRITE}(6,2)
3375. 2 \text{ FORMAT(}-*, 'THE SHRINKAGE EQUATION DEVELOPED FROM INTEGRATING EACH}
3376. 2 \text{ X INDEPENDENT VARIABLES PARTIAL DERIVATIVE ABOVE AND ADDING THE')
3377. \text{WRITE}(6,3)
3378. 3 \text{ FORMAT(}-*, 'CONSTANT TERM DEVELOPED PREVIOUSLY')
3379. \text{WRITE}(6,4) \text{WRSTSK}
3380. 4 \text{ FORMAT(}-*, 'LUX, 'CONST ANT TERM = *F15.3, ' CUBIC INCHES')}
3381. \text{RETURN}
3382. \text{END}
3383. \text{SUBROUTINE TEQVAL}
3384. \text{COMMON} / \text{NOZERO} / \text{NOZERO}
3385. \text{COMMON} / zVAL/zVAL(10)
3386. \text{INTEGER} \* 2 zVAL
3387. \text{COMMON} / \text{POS}/\text{POS}(10)
3388. \text{COMMON} / \text{WID}/\text{WID}(10)
3389. \text{COMMON} / \text{HEIGHT}/\text{HEIGHT}(10)
3390. \text{COMMON} / \text{WRSTSK} / \text{WRSTSK}
3391. \text{COMMON} / \text{SHKTL}/\text{SHKTL}
3392. \text{COMMON} / \text{TSDW}/\text{TSDW}(10)
3393. \text{COMMON} / \text{TSGD}/\text{TSGD}(10)
3394. \text{COMMON} / \text{TSGP}/\text{TSGP}(10)
3395. \text{COMMON} / \text{SHK}/\text{SHK}(10)
3396. \text{COMMON} / \text{DSHKVL}/\text{DSHKVL}
3397. \text{SHRINK} = \text{WRSTSK}
3398. \text{DO} 10 \text{I} = 1, \text{NOZERO}
3399. \text{IF}(\text{ZSIDE}(i) = \text{EQ.} 0) \text{GO TO} 10
3400. \text{SHRINK} = \text{SHRINK} + \text{WID}(i) \cdot \text{TSDW}(i)
3401. \text{SHRINK} = \text{SHRINK} + \text{HEIGHT}(i) \cdot \text{TSGD}(i)
3402. \text{SHRINK} = \text{SHRINK} + \text{POS}(i) \cdot \text{TSGP}(i)
3403. 10 \text{ CONTINUE}
3404. \text{DSHKVL} = \text{SHRINK}
3405. \text{IF}(\text{SHRINK} \lt 1. \text{SHKTL}) \text{SHRINK} = 0.0
3406. \text{RETURN}
3407. \text{END}
3408. \text{SUBROUTINE OUDDUAL}
3409. \text{WRITE}(6,1)
3410. 1 \text{ FORMAT(}-*, 'DUAL RESULTS')
3411. \text{RETURN}
3412. \text{END}
3413. \text{SUBROUTINE FIRST}
3414. \text{COMMON} / \text{ITBEND} / \text{ITBEND}
3415. \text{COMMON} / \text{DFUNC2} / \text{DFUNC2}
3416. \text{COMMON} / \text{GENINF} / \text{GENINF}
3417. \text{INTEGER} \* 2 \text{ BENINF}
3418. \text{COMMON} / \text{UPBND} / \text{UPBND}
3419. \text{COMMON} / \text{IDFLAG} / \text{IDFLAG}
COMMON /BNBEST/BNBEST
INTEGER BNBEST
COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT
BENINF=0
CALL DFVIT
UPBND=1.0E50
CALL DFV
IF(IFLAG.EQ.5) BENINF=1
IF(IFLAG.EQ.5) RETURN
CALL DUAL
IF(PCONT(35).EQ.1) CALL OUDUAL
UPBND=DFUNC2
BNBEST=1
ITBEND=1
CALL BEND1
RETURN
SUBROUTINE BEND1
COMMON /DEXIS/DEXIS(10)
COMMON /DFUNC2/DFUNC2
COMMON /COEFZ/COEFZ(10)
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
COMMON /NOZERO/NOZERO
COMMON /FOFX/FOFX
COMMON /ZMULT/ZMULT(10)
FOFX=0.0
F0FX=DFUNC2
DO 10 I=1,NOZERO
IF(ZVAL(I).EQ.0) GO TO 10
10 CONTINUE
DO 15 I=1,10
ZMULT(I)=0.0
15 CONTINUE
DO 20 I=1,NOZERO
ZMULT(I)=COEFZ(I)-(1000.0*DEXIS(I))
20 CONTINUE
CALL STOBEN
CALL STORES
RETURN
END
SUBROUTINE STOBEN
COMMON /ITBEND/ITBEND
COMMON /FOFX/FOFX
COMMON /ZMULT/ZMULT(10)
COMMON /NOZERO/NOZERO
COMMON /BSTMLT/BSTMLT(65,10)
COMMON /BSTFX/BSTFX(65)
I=ITBEND
DO 10 J=1,10
BSTMLT(I,J)=0.0
10 CONTINUE
BSTFX(I)=0.0
DO 20 I=1,NOZERO
BSTMLT(ITBEND,I)=ZMULT(I)
20 CONTINUE
BSTFX(ITBEND)=FCFX
RETURN
ENDD
SUBROUTINE STORES
COMMON /ITBEND/ITBEND
COMMON /DFUNC2/DFUNC2
COMMON /CURSR2/CURSR2
COMMON /WID/WID(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /POS/POS(10)
COMMON /NOZERO/NOZERO
COMMON /ZVAL/ZVAL(10)
COMMON /ZVAL/ZVAL(10)
INTEGER * ZVAL
COMMON /BSTZVL/BSTZVL(65,10)
INTEGER BSTZVL
COMMON /BSTWID/BSTWID(65,10)
COMMON /BSTHT/BSTHT(65,10)
COMMON /BSTPOS/BSTPOS(65,10)
COMMON /BSTFTN/BSTFTN(65)
COMMON /BSTSHK/BSTSHK(65)
I=ITBEND
DO 10 J=1,10
BSTZVL(I,J)=0
BSTWID(I,J)=0.0
BSTHT(I,J)=0.0
BSTPOS(I,J)=0.0
10 CONTINUE
BSTFTN(I)=0.0
BSTSHK(I)=0.0
00 20 J=1,NOZERO
BSTZVL(I,J) = ZVAL(J)
BSTWID(I,J)=WID(J)
BSTHT(I,J)=HEIGHT(J)
BSTPOS(I,J)=POS(J)
20 CONTINUE
BSTFTN(I)=DFUNC2
BSTSHK(I)=CURSR2
RETURN
END
SUBROUTINE BENDER
COMMON /BENFLG/BENFLG
INTEGER * 2 BENFLG
COMMON /IDFLAG/IDFLAG
COMMON /DFUNC2/DFUNC2
COMMON /UPBND/UPBND
COMMON /ITBEND/ITBEND
COMMON /BNBEST/BNBEST
INTEGER BNBEST
COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT
BENFLG=0
00 10 CALL OBCST
IF(PCONT(31).EQ.1) CALL OBCST
CALL OPTCHK
IF(PCONT(32).EQ.1) CALL OBOCK
IF(BENFLG.EQ.1) GO TO 20
CALL SETZ
IF(PCONT(33).EQ.1) CALL OBNWZ
CALL DFP
IF(IDFLAG.EQ.5) CALL FATHOM
IF(IDFLAG.EQ.5.AND.PCONT(34).EQ.1) CALL OUTDF
IF(IDFLAG.EQ.5) GO TO 10
CALL OUAL
IF(IPCNT(35).EQ.1) CALL OUAL
ITBEND=ITBEND+1
IF(IFUNC2.LE.UPBND) UPBND=IFUNC2
IF(IFUNC2.LE.UPBND) BNBEST=ITBEND
IF(IPCNT(36).EQ.1) CALL OUALUPD
CALL BEND1
GO TO 10
20 RETURN
END

SUBROUTINE BENCST
COMMON /NOZERO/NOZERO
COMMON /ZVAL/ZVAL
INTEGER * 2 ZVAL
COMMON /COMB/COMB
INTEGER * 2 COMB
COMMON /CUMUSE/CUMUS
INTEGER * 2 COMUSE
COMMON /NUMCMB/NUMCMB
COMMON /YNOT/YNOT
COMMON /ITBEND/ITBEND
COMMON /BSTMLT/BSTMLT
COMMON /BSTFX/BSTFX
DIMENSION WORK(65)
DO 10 I=1,NUMCMB
IF(CUMUSE(I).EQ.0) GO TO 10
XMAX=-1.0E50
DO 20 J=1,ITBEND
WORK(J)=BSTFX(J)
DO 30 K=1,NOZERO
WORK(J)=WORK(J)+(BSTMLT(J,K)*COMB(K))
30 CONTINUE
20 CONTINUE
IF(WORK(I).GT.XMAX) XMAX=WORK(J)
IF(WORK(I).LE.XMIN) XMIN=WORK(I)
DO 100 I=1,NUMCMB
IF(CUMUSE(I).EQ.0) GO TO 100
IF(BSTVL(I).LE.XMIN) XMIN=BSTVL(I)
IF(BSTVL(I).LE.XMIN) NUMCMS=I
YNOT=XMIN
RETURN
END

SUBROUTINE OUBCST
COMMON /BSTMLT/BSTMLT
COMMON /BSTFX/BSTFX
COMMON /BSTVL/BSTVL
COMMON /NUMCMB/NUMCMB
COMMON /NOZERO/NOZERO
COMMON /ITBEND/ITBEND
COMMON /YNOT/YNOT
COMMON /COMB/COMB
3600. INTEGER * 2 COMB
3601. COMMON /COMUSE/COMUSE(100)
3602. INTEGER * 2 ZVAL
3603. COMMON /ZVAL/ZVAL(10)
3604. INTEGER * 2 ZVAL
3605. COMMON /TYP/TYP(10,6)
3606. LOGICAL * 1 TYP
3607. COMMON /BPRNT/BPRNT(100)
3608. COMMON /NEWCMB/NEWCMB
3609. WRITE(6,1) 1 FORMAT(*'BENDER MASTER PROBLEM RESULTS')
3610. WRITE(6,2) ITBEND 2 FORMAT(*'CURRENT ITERATION - *',141
3611. WRITE(6,3) 3 FORMAT(*'BELOW ARE LISTED THE BENDER CONSTRAINTS GENERATED THUS
3612. X FAR *)
3613. WRITE(6,4) 4 FORMAT(*'-*',10X,'ITERATION FIX,Z*)
3614. WRITE(6,5) 5 FORMAT(*',10X,12X,10!2X,6A1,2X))
3615. WRITE(6,6) 6 FORMAT(*',10X,5X,6I13,3X,5X,12X,5X,10!13,3X,5X)
3616. DO 10 1=1,ITBEND
3617. WRITE(6,11) I,BSTFX1I),BSTMLTIJ),J=1,NOZERO)
3618. WRITE(6,6) 11 FORMAT(*',15,5X,E12.5,10!E10.3)
3619. WRITE(6,6) 10 CONTINUE
3620. WRITE(6,6)
3621. WRITE(6,100)
3622. WRITE(6,101)
3623. WRITE(6,102) 101 FORMAT(*',14,2X,5X,COMBINATIONS',12X,5X,'LAST CONSTRAINT
3624. XVALUE',5X,'MAXIMUM CONSTRAINT VALUE*)
3625. WRITE(6,6) 102 FORMAT(*',6X,5X,36A1)
3626. WRITE(6,6) 10 CONTINUE
3627. WRITE(6,6)
3628. WRITE(6,100)
3629. WRITE(6,6)
3630. WRITE(6,6)
3631. WRITE(6,6)
3632. WRITE(6,6)
3633. WRITE(6,6)
3634. WRITE(6,6)
3635. WRITE(6,6)
3636. WRITE(6,6)
3637. IF (COMUSE(I).EQ.0) GO TO 110
3638. WRITE(6,111) I,COMBIJ),J=1,NOZERO),BPRNT(I),ECSTVL(I)
3639. WRITE(6,111) 111 FORMAT(*',14,2X,5X,6(13,3X),5X,4X,E13.6,4X,5X,5X,E13.6)
L. 110 CONTINUE
3642. WRITE(6,6)
3643. WRITE(6,200)
3644. 200 FORMAT("-",'THE OPTIMUM SOLUTION TO THE MASTER PROBLEM IS")
3645. WRITE(6,6)
3646. WRITE(6,201) YNOT
3647. 201 FORMAT("",10X,'MINIMUM VALUE OF Enumeration - ',E15.7)
3648. WRITE(6,202) NEWCMB
3649. 202 FORMAT("",10X,'COMBINATION NUMBER - ',I4)
3650. RETURN
3651. END
3652. SUBROUTINE OPTCHK
3653. COMMON /YNOT/YNOT
3654. COMMON /UPBND/UPBND
3655. COMMON /BENSTP/BENSTP
3656. COMMON /BENFLG/BENFLG
3657. INTEGER * 2 BENFLG
3658. DIFF=UPBND-YNOT
3659. IF (DIFF.LE.BENSTP) BENFLG=1
3660. RETURN
3661. END
3662. SUBROUTINE OUBOCK
3663. COMMON /YNOT/YNOT
3664. COMMON /UPBND/UPBND
3665. COMMON /BENSTP/BENSTP
3666. COMMON /BENFLG/BENFLG
3667. INTEGER * 2 BENFLG
3668. COMMON /ITBEND/ITBEND
3669. WRITE(6,1)
3670. 1 FORMAT("",38X,'RESULTS OF MASTER PROBLEM OPTIMIZATION CHECK')
3671. WRITE(6,2)
3672. 2 FORMAT("",*)
3673. WRITE(6,3) BENSTP
3674. 3 FORMAT("",10X,'TOLERANCE PARAMETER = ',E15.8)
3675. WRITE(6,4) UPBND
3676. 4 FORMAT("",10X,'CURRENT UPPER BOUND = ',E12.5)
3677. WRITE(6,5) YNOT
3678. 5 FORMAT("",10X,'CURRENT VALUE FROM MASTER PROBLEM = ',E12.5)
3679. DIFF=UPBND-YNOT
3680. WRITE(6,6) DIFF
3681. 6 FORMAT("",10X,'DIFFERENCE = ',E12.5)
3682. RETURN
3683. END
3684. SUBROUTINE SETZ
3685. COMMON /NEWCMB/NEWCMB
3686. COMMON /NOZERC/NOZERC
3687. COMMON /ZVAL/ZVAL(10)
3688. INTEGER* 2 ZVAL
3689. COMMON /COMB/COMB(100,6)
3690. INTEGER* 2 COMB
3691. COMMON /COMUSE/COMUSE(100)
3692. INTEGER* 2 COMUSE
3693. DO 100 I=1,NOZERC
3694. IF (COMB(NEWCMB,I).EQ.0) ZVAL(I)=0
3695. IF (COMB(NEWCMB,I).EQ.1) ZVAL(I)=1
3696. 100 CONTINUE
3697. RETURN
3698. END
3699. SUBROUTINE OUBNWZ
3701. COMMON /NEWCMB/NEWCMB
3702. COMMON /ZVAL/ZVAL(10)
3703. INTEGER * 2 ZVAL
3704. COMMON /NOZERO/NOZERO
3705. COMMON /TYP/TYP(10,6)
3706. LOGICAL * 1 TYP
3707. WRITE(6,1)
3708. 1 FORMAT('**','NEW BENDER COMBINATION TO BE USED TO DEVELOP SUBPROB.
3709. XEH**')
3710. WRITE(6,2)
3711. 2 FORMAT('**','**
3712. DO 10 I=1,NOZERO
3713. IF(ZVAL(I).EQ.0) WRITE(6,12) I,(TYP(I,J),J=1,6)
3714. IF(ZVAL(I).EQ.1) WRITE(6,11) I,(TYP(I,J),J=1,6)
3715. 11 FORMAT(*','10X,'VARIABLE **,**(I3;2X,6A1,2X,'IS IN SOLUTION SET')
3716. 12 FORMAT(*','10X,'VARIABLE **,**(I3;2X,6A1,2X,'IS NOT IN SOLUTION SET')
3717. 10 CONTINUE
3718. RETURN
3719. END
3720. SUBROUTINE FATHOM
3721. COMMON /COMB/COMB(100,6)
3722. INTEGER * 2 COMB
3723. COMMON /COMUSE/COMUSE(100)
3724. INTEGER * 2 COMUSE
3725. COMMON /NUMCMB/NUMCMB
3726. COMMON /ZVAL/ZVAL(10)
3727. INTEGER * 2 ZVAL
3728. COMMON /NOZERO/NOZERO
3729. COMMON /TYP/TYP(10,6)
3730. INTEGER ZTYPE
3731. COMMON /NEWCMB/NEWCMB
3732. COMMON /IFATH/IFATH(100)
3733. DIMENSION ICOMP(100)
3734. COMUSE(NEWCMB)=0
3735. DO 1 I=1,100
3736. IFATH(I)=0
3737. 1 CONTINUE
3738. DO 10 I=1,10
3739. ICOMP(I)=0
3740. 10 CONTINUE
3741. DO 20 I=1,NOZERO
3742. IF(ZVAL(I).EQ.1) GO TO 20
3743. ICOMP(I)=1
3744. 20 CONTINUE
3745. DO 100 I=1,NUMCMB
3746. IF(COMUSE(I).EQ.0) GO TO 100
3747. DO 110 J=1,NOZERO
3748. IF(COMB(I,J).EQ.1.AND.COMB(I,J).EQ.ICOMP(J)) GO TO 100
3749. 110 CONTINUE
3750. IFATH(I)=1
3751. COMUSE(I)=0
3752. 100 CONTINUE
3753. RETURN
3754. END
3755. SUBROUTINE OUTDFM
3756. COMMON /NOZERC/NOZERO
3757. COMMON /COMB/COMB(100,6)
3758. INTEGER * 2 COMB
3759. COMMON /TYP/TYP(10,6)
3760. LOGICAL * 1 TYP
COMMON /IFATH/, IFATH(100)
INTEGER * 2 IFATH
COMMON /NEWCMB/, NEWCMB
COMMON /ITBEND/, ITBEND
WRITE(6,1)
1 FORMAT(*',43X,'RESULTS OF INFEASIBILITY FATHOMING')
WRITE(6,2)
2 FORMAT(*',3X)
WRITE(6,3)
3 FORMAT(*',50X,'FATHOMED CONSTRAINTS')
WRITE(6,4) ((TYPI(1,J), J=1,6), I=1,NOZERO)
4 FORMAT(*',42X,36A1)
WRITE(6,2)
WRITE(6,6) (COMBINEWCMB, J=1, NOZERO)
6 FORMAT(*',20X,'CURRENT COMBINATION* ,3X,6A1,3X,13)
0010 1=1, NUMCMB
IF(IIFATH(1)=0) Go TO 10
IF(1.EQ.NEWCMB) Go TO 10
WRITE10,11) (COMBINEWCMB, J=1,NOZERO)
11 FORMAT(*',20X,'COMBINATION NUMBER* ,13,1X,6(I,13,3X))
10 CONTINUE
RETURN
END
SUBROUTINE OUBUPD
COMMON /UPBND/, UPBND
COMMON /DFUNC2/, DFUNC2
COMMON /ITBEND/, ITBEND
COMMON /BNBEST/, BNBEST
INTEGER 3NBEST
WRITE(6,1)
1 FORMAT(*',43X,'MASTER PROGRAM UPDATE INFORMATION')
WRITE(6,2)
2 FORMAT(*',3X)
WRITE(6,3) UPBND
3 FORMAT(*',10X,'UPPER Bound = ', E12.5)
WRITE(6,4) DFUNC2
4 FORMAT(*',10X,'SUBPROBLEM SOLUTION = ', E12.5)
WRITE(6,5) ITBEND
5 FORMAT(*',10X,'BENDER Iteration Number = ', 13)
WRITE(6,6) BNBEST
6 FORMAT(*',10X,'INDEX OF CURRENT BEST SOLUTION = ', 12)
RETURN
END
SUBROUTINE TEST
COMMON /SOLIT/, SOLIT
INTEGER SOLIT
COMMON /IPRT/, IPRT
COMMON /PCNT/, PCNT
INTEGER * 2 PCNT
COMMON /SFLAG/, SFLAG
INTEGER * 2 SFLAG
IT = 1
IF(I.EQ.1) Call EQUTIN
IF(I.EQ.1) RETURN
CALL ADJUST
CALL CENTER
CALL INTFAC
CALL INITAL
CALL SLDKEY
SFLAG = 1
ITERT=0

IF(PCONT(24).EQ.1) CALL OUTCOD
IF(PCONT(25).EQ.1) CALL OUTKEY
ISET=0
IF(PCONT(26).EQ.1) CALL OUTTMP(ITERT)
IF(PCONT(27).EQ.1) CALL OUTINT(ITERT)

CALL EQUATN
IF(SFLAG.EQ.1) RETURN
GO TO 10
ITERT=1,SOLIT
CALL SIMLAT
CALL SOLKEY
IF(SFLAG.EQ.2) GO TO 30
IF(ITERT.EQ.ISET) GO TO 20
GO TO 10
IF(PCONT(26).EQ.1) CALL OUTTMP(ITERT)
IF(PCONT(27).EQ.1) CALL OUTINT(ITERT)
ISET=ISET+IPRT
CONTINUE
30 CALL SHKSIZ
IF(PCONT(27).EQ.1) CALL OUTSLD(ITERT)
RETURN
END

SUBROUTINE ADJUST
COMMON /T/T(30,80)
INTEGER*2 T
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /ZCORNR/ZCORNR(10,4,2)
INTEGER ZCORNR
COMMON /ZVAL/ZVAL(10)
INTEGER*2 ZVAL
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /WID/WID(10)
INTEGER WIDTH
COMMON /LENGTH/LENGTH
INTEGER LENGTH
COMMON /NOZERO/NOZERO
INTEGER NOZERO
COMMON /IPOS/IPOS(10)
COMMON /IHGT/IHGT(10)
COMMON /IWID/IWID(10)
COMMON /SENSL/SENSL
INTEGER SENSL
COMMON /SENSW/SENSW
INTEGER SENSW
COMMON /KEY/KEY(10,2)
COMMON /ISENS/ISENS
DO 1 I=1,NGZERG
IP0Sm=P0SlI)
IHGT(I)=HEIGHT(I)
IWIO(I)=WIO(I)
CONTINUE
DO 5 J=1,30
T(I,J)=0
CONTINUE
5 CONTINUE
3882. DO 10 I=1,WIDTH
3883. DO 20 J=1,LENGTH
3884. T(I,J)=X(I,J)
3885. 20 CONTINUE
3886. 10 CONTINUE
3887. DO 30 I=1,NOZERO
3888. IF(ZVAL(I).EQ.1) GO TO 30
3889. I1=ZCORNK(I,1,1)
3890. I2=ZCORNK(I,3,1)
3891. J1=ZCORNK(I,1,2)
3892. J2=ZCORNK(I,2,2)
3893. DO 40 IS=I1,I2
3894. DO 50 JS=J1,J2
3895. T(IS,JS)=1
3896. 50 CONTINUE
3897. 40 CONTINUE
3898. 30 CONTINUE
3899. C APPLY SENSITIVITY
3900. SENSL=LENGTH-2*SENS+2
3901. SENSW=WIDTH-2*SENS+2
3902. DO 60 I=1,SENSW
3903. T(I,1)=0
3904. T(I,SENSL)=0
3905. 60 CONTINUE
3906. DO 65 J=1,SENSL
3907. T(1,J)=0
3908. T(SENSW,J)=0
3909. 65 CONTINUE
3910. J2=LENGTH-2
3911. JCT=SENSL
3912. DO 70 J=1,J2
3913. J1=LENGTH-J
3914. DO 80 L=1,ISENS
3915. JCT=JCT-1
3916. DO 90 I=1,WIDTH
3917. T(I,JCT)=T(I,J1)
3918. 90 CONTINUE
3919. 80 CONTINUE
3920. 70 CONTINUE
3921. I2=WIDTH-2
3922. ICT=SENSW
3923. DO 90 I=1,I2
3924. I1=WIDTH-1
3925. DO 100 L=1,ISENS
3926. ICT=ICT-1
3927. DO 110 J=1,SENSL
3928. T(ICT,J)=T(I1,J)
3929. 110 CONTINUE
3930. 100 CONTINUE
3931. 90 CONTINUE
3932. DO 200 I=1,NOZERO
3933. IF(ZSIDE(I).EQ.1) CALL LOWER(I)
3934. IF(ZSIDE(I).EQ.4) CALL UPPER(I)
3935. IF(ZSIDE(I).EQ.2) CALL RIGHT(I)
3936. IS=1
3937. 100 CONTINUE
3938. 90 CONTINUE
3939. 80 CONTINUE
3940. 70 CONTINUE
3941. 60 CONTINUE
3942. 50 CONTINUE
3943. 40 CONTINUE
3944. 30 CONTINUE
3945. 20 CONTINUE
3946. 10 CONTINUE
3947. 5 CONTINUE
200 CONTINUE
RETURN
END

SUBROUTINE LOWER(IS)
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /IPOS/IPOS(10)
COMMON /IHGT/IHGT(10)
COMMON /IWID/IWID(10)
COMMON /KEY/KEY(10,2)
COMMON /ISENS/ISENS
COMMON /ISEND/ISEND
COMMON /GET/GET
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /RECL/RECL(20)
INTEGER RECL
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /SEND/XSEND
COMMON /GET/GET
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZREC/ZREC(10)
INTEGER ZREC

I=IS
GET=POSI(I)-IPOS(I)
CALL ROUND
I1=KEY(I,1)
I2=KEY(I,1)+(ZBOUND(I)*ISENS)-1
J1=KEY(I,2)
J2=KEY(I,2)+(IPOS(I)*ISENS)+ISEND-1
IF(POS(I).LT.R1) GO TO 100
CALL FINAJ(I1,I2,J1,J2)
GET=WID(I)-IWID(I)
CALL ROUND
J1=J2+(IWID(I)*ISENS)+ISEND
J2=KEY(I,2)+(ISENS*RECL(ZREC(I))+1)
CALL FINAJ(I1,I2,J1,J2)
GET=HEIGHT(I)-IHGT(I)
CALL ROUND
IF(IHGT(I).EQ.ZBOUND(I)) GO TO 200
J1=KEY(I,1)+(IHGT(I)*ISENS)+ISEND
J2=KEY(I,2)
CALL FINAJ(I1,I2,J1,J2)
200 RETURN
END

SUBROUTINE UPPER(IS)
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /IPOS/IPOS(10)
COMMON /IWID/IWID(10)
COMMON /KEY/KEY(10,2)
COMMON /ISENS/ISENS
COMMON /ISEND/ISEND
COMMON /GET/GET
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /RECLFN/RECLFN(20)
INTEGER RECLFN
COMMON /RECWD/RECWD(20)
INTEGER RECWD
COMMON /R/R(5)
I=IS
GET=POS(I)-POS(I)
CALL ROUND
J1=KEY(I,2)
J2=KEY(I,2)+(POS(I)*ISENS)+ISEND-1
I1=KEY(I,1)
J2=KEY(I,1)+(ZBOUND(I)*ISENS)-1
IF(POS(I).LT.R(I)) GO TO 100
CALL FINADJ(I1,I2,J1,J2)
GET=WID(I)-WID(I)
CALL ROUND
J1=J2+(WID(I)*ISENS)+ISEND+1
J2=KEY(I,2)+(ISENS*RECLFN(REC(I)))-1
CALL FINADJ(I1,I2,J1,J2)
IF(IHGT(I).EQ.ZBOUND(I)) GO TO 200
GET=HEIGHT(I)-IHGT(I)
CALL ROUND
J2=I1+(ZBOUND(I)*ISENS)-1-(IHGT(I)*ISENS)-ISEND
J1=KEY(I,2)
CALL FINADJ(I1,I2,J1,J2)
GET=HEIGHT(I)-IHGT(I)
RETURN
END
SUBROUTINE RIGHT(I)
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /POS/POS(10)
COMMON /WID/WID(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /RECLFN/RECLFN(20)
INTEGER RECLFN
I=IS
GET=POS(I)-POS(I)
CALL ROUND
J1=KEY(I,1)
J2=KEY(I,1)+(ZBOUND(I)*ISENS)-1
CALL FINADJ(I1,I2,J1,J2)
GET=WID(I)-WID(I)
CALL ROUND
J1=I2+(WID(I)*ISENS)+ISEND+1
J2=KEY(I,1)+(ISENS*ZBOUND(I)))-1
CALL FINADJ(I1,I2,J1,J2)
GET=HEIGHT(I)-IHGT(I)
CALL ROUND

J1=KEY(I,2)+(IHGT(I)*ISENS)+ISEND
J1=KEY(I,1)
CALL FINADJ(I1,I2,J1,J2)
RETURN
END

SUBROUTINE LEFT(IS)
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /IPOS/IPOS(10)
COMMON /IHGT/IHGT(10)
COMMON /WID/WID(10)
COMMON /KEY/KEY(10,2)
COMMON /ISENS/ISENS
COMMON /ISEND/ISEND
COMMON /GET/GET
COMMON /WID/WID(10)
COMMON /POS/POS(U)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /WID/WID(20)
COMMON /RECEID/RECEID(20)
INTEGER RECEID

I=IS
GET=POS(I)-IPOS(I)
CALL ROUND
I1=KEY(I,1)
I2=I2+(IPOS(I)*ISENS)+ISEND
J1=KEY(I,2)
J2=J1+(ZBOUND(I)*ISENS)-1
CALL FIANOJ(I1,I2,J1,J2)
GET=WID(I)-1*WID(I)
CALL ROUND
I1=I2+(WID(I)*ISENS)+ISEND
J1=KEY(I,1,1)
J2=KEY(I,1,1)+RECEID(1)*ZREC(I)*ISENS)-1
CALL FIANOJ(I1,I2,J1,J2)
GET=HEIGHT(I)-1-IHGT(I)
CALL ROUND
J2=J2-1(IHGT(I)*ISENS)+ISEND
I1=KEY(I,1)
I2=I2+(ZREC(I)*ZREC(I)*ISENS)-1
CALL FINADJ(I1,I2,J1,J2)
RETURN
END

SUBROUTINE ROUND
COMMON /GET/GET
COMMON /R/R(5)
COMMON /ISENS/ISENS
COMMON /ISEND/ISEND

IF(GET.LE.R(I)) GO TO 10
I1=ISENS-1
DO 10 I=1,11
Z2=Z+1
IF(GET.Z.R(I).AND.GET.LE.R(2)) GO TO 20
10 CONTINUE

ISEND=ISENS
GO TO 100
20 ISENT=1
GO TO 100
10 ISEND=0
100 RETURN

SUBROUTINE FINADJ(I1, I2, J1, J2)
COMMON /T/T(80,80)
INTEGER T
DO 10 I=I1, I2
DO 20 J=J1, J2
T(I,J)=1
20 CONTINUE
10 CONTINUE
RETURN
END

SUBROUTINE CENTER
COMMON /T/(T(80,80)
INTEGER T
COMMON /SENSW/SENSW
INTEGER SENS*
COMMON /SENSL/SENSL
INTEGER SENSL
COMMON /ISENS/ISENS
COMMON /ISTART/ISTART
COMMON /IEND/IEND
COMMON /JSTART/JSTART
COMMON /JEND/JEND
COMMON /SANDW/SANDW
INTEGER SANDW
COMMON /SANDL/SANDL
INTEGER SANDL
IS=SENSW
JS=SENSL
SENSW=SENSW-1
SENSL=SENSL-1
DO 10 I=2,SENSW
DO 20 J=2,SENSL
IF(T(I,J).NE.1) GO TO 25
20 CONTINUE
10 CONTINUE
ISTART=I
DO 30 I=ISTART,SENSW
30 CONTINUE
4161. DO 40 J=2,SENSL
4162. IF(T(I,J).NE.1) GO TO 30
4163. 40 CONTINUE
4164. IEND=I
4165. GO TO 50
4166. 30 CONTINUE
4167. DO 60 J=2,SENSL
4168. DO 70 I=2,SENSW
4169. IF(T(I,J).NE.1) GO TO 75
4170. 70 CONTINUE
4171. 60 CONTINUE
4172. JSTART=J
4173. DO 90 J=JSTART,SENSL
4174. IEND=I2,SENSW
4175. IF(T(I,J).NE.1) GO TO 90
4176. 100 CONTINUE
4177. JEND=J
4178. GO TO 200
4179. 90 CONTINUE
4180. 200 ISTART=ISTART-(SANDW*ISENS)
4181. IEND=IEND+(SANDW*ISENS)-1
4182. JSTART=JSTART-(SANDL*ISENS)
4183. JEND=JEND+(SANDL*ISENS)-1
4184. I=ISTART-1
4185. I2=IEND+1
4186. J1=JSTART-1
4187. J2=JEND+1
4188. DO 300 J=J1,SENSL
4189. T(I1,J)=0
4190. T(I2,J)=0
4191. 300 CONTINUE
4192. DO 310 I=1,SENSW
4193. T(I,J1)=0
4194. T(I,J2)=0
4195. 310 CONTINUE
4196. ISTART=11
4197. IEND=I2
4198. JSTART=J1
4199. JEND=J2
4200. SENSW=I5
4201. SENSL=JS
4202. RETURN
4203. END
4204. SUBROUTINE INTFAu
4205. COMMON /T/T(80,80)
4206. INTEGER*2 T
4207. COMMON /JSTART/JSTART
4208. COMMON /IEND/IEND
4209. COMMON /JSTART/JSTART
4210. COMMON /JEND/JEND
4211. DD 10 I=ISTART,IEND
4212. DO 20 J=JSTART,JEND
4213. IF(T(I,J).EQ.1) GO TO 50
4214. IF(T(I,J).EQ.2) GO TO 100
4215. IF(T(I,J).EQ.4) GO TO 100
4216. GO TO 20
4217. 50 I1=I-1
4218. I2=I+1
4219. J1=J-1
J2=J+1
DO 60 IT=I1,12
DO 70 JT=J1,J2
IF(IT(IT,JT).EQ.2) GO TO 80
IF(IT(IT,JT).EQ.4) GO TO 80
IF(IT(IT,JT).EQ.5) GO TO 80
70 CONTINUE
60 CONTINUE
GO TO 20
80 II=I-1
I2=I+1
J1=J-1
J2=J+1
DO 110 IT=I1,I2
DO 120 JT=J1,J2
IF(IT(IT,JT).EQ.1) GO TO 150
IF(IT(IT,JT).EQ.6) GO TO 150
120 CONTINUE
110 CONTINUE
GO TO 20
150 T(I,J)=5
20 CONTINUE
10 CONTINUE
RETURN
END
SUBROUTINE INITAL
COMMON /T/T(80,80)
INTEGER*2 T
COMMON /ISTART/I START
COMMON /IEND/IEND
COMMON /JSTART/JSTART
COMMON /JEND/JEND
COMMON /TEMP/TEMP I 80,80)
COMMON /FAC/FACUO)
COMMON /TSANO/TSANO
COMMON /TAIR/TAIR
COMMON /TSTEEL/TSTEEL
COMMON /TINTFM/TINTFM
COMMON /TINTFS/TINTFS
COMMON /TCHILL/TCHILL
COMMON /TRISER/TRISER
10 1=1,10
FAC(I)=1.0
30 CONTINUE
FAC(I)=1.41
FAC(3)=1.414
FAC(7)=1.414
SUBROUTINE SLDKEY
COMMON /NOKEY/NOKEY
COMMON /ISTART/ISTART
COMMON /IEND/IEND
COMMON /JSTART/JSTART
COMMON /JEND/JEND
COMMON /T/180.0)
INTEGERS T
COMMON /IKEY/IKEY(40)
COMMON /JKEY/JKEY(40)
IT=1
DO 10 I=ISTART,IEND
DO 20 J=JSTART,JEND
IF(IT(I,J),NE.4) GO TO 20
12=1+1
IF(TII2,J),NE.2) GO TO 20
IKEY(IT)=I
JKEY(IT)=J
IT=IT+1
10 CONTINUE
20 CONTINUE
NOKEY=IT-1
RETURN
END
SUBROUTINE QUTCOD
COMMON /ISTART/ISTART
COMMON /IEND/IEND
COMMON /JSTART/JSTART
COMMON /JEND/JEND
COMMON /T/180.0)
INTEGER*2 T
WRITE(6,10)
10 FORMAT(' I•*46X,' SET-UP SOLIDIFICATION MATRIX OF CODES')
WRITE(6,15)
15 FORMAT(•-*46X,' CODES FOLLOW')
107 FORMAT(' 1 20X,'SAND/CASTING INTERFACE - CASTING')
108 FORMAT(' 1 20X,'SAND/CASTING INTERFACE - SAND')
CALL OUTVAR
RETURN
END

SUBROUTINE OUTKEY
COMMON /NOKEY/NOKEY
COMMON /1KEY/1KEY
COMMON /JKEY/JKEY
WRITE(6,1)
1 FORMAT(' 1 20X')
WRITE(6,10)
10 FORMAT(' 1 20X', * THE FOLLOWING ARE THE KEY RISER-CASTING CONTACTS')
DO 20 I = 1, NOKEY
WRITE(6,30) I, IKEY(I), JKEY(I)
30 FORMAT(' 1 20X', ' KEY COMPONENT ', I4, ' I = ', 13, ' J = ', 13)
20 CONTINUE
RETURN
END

SUBROUTINE SIMLAT
COMMON /IKEY/1KEY
COMMON /JKEY/JKEY
COMMON /XTIME/XTIME
COMMON /XDIM/XDIM
COMMON /STDENS/STDENS
COMMON /SADENS/SADENS
DIMENSION ROW1(80), ROW2(80)
COMMON /TEMP/TEMP
COMMON /ISENS/ISENS
COMMON /STFAC/STFAC
COMMON /SANDFC/SANDFC
COMMON /XVAL/XVAL

STFAC=XTIME/STDENS*XDIM
SANDFC=XTIME/SADENS*XDIM
I=IKEY+1
J1=JKEY+1
I2=IEND-1
J2=JEND-1
DO 10 J=J1, J2
CALL COMPUT(I1, J)
ROW1(J)=XVAL
10 CONTINUE
IS=I1+1
DO 3 J=J1, J2
CALL COMPUT(IS, J)
ROW2(J)=XVAL
3 CONTINUE
IS=IS+1
DO 10 I=IS, I2
II=I-2
DO 11 J=J1, J2
TEMP(II, J)=ROW1(J)
11 CONTINUE
DO 12 J=J1, J2
ROW1(J)=ROW2(J)
12 CONTINUE
DO 13 J=J1, J2
CALL COMPUT(I,J)
ROW2(J)=XVAL
13 CONTINUE
10 CONTINUE
IFI=I-1
DO 15 J=J1,J2
TEMP(IF1,J)=ROW1(J)
15 CONTINUE
RETURN
END
SUBROUTINE COMPUT(I,J)
COMMON /TEMP/TEMP(80,80)
COMMON /T/T(80,80)
INTEGER*2 T
COMMON /XVAL/XVAL
COMMON /XKVAL/XKVAL
COMMON /SPEC/SPEC
COMMON /XTIME/XTIME
COMMON /XDIM/XDIM
COMMON /SANDFC/SANDFC
COMMON /IMIN/IMIN
CALL SPECFC(I,J)
FACT=STFAC/SPEC
IF(IF(I,J).EQ.1) FACT=SANDFC/SPEC
IMIN=1
IMIN=2
IFIIMIN.EQ.1) GO TO 100
IT=1
12=I-1
13=I+1
J2=J-1
J3=J+1
IF(IT(J,J).EQ.5) GO TO 200
IF(IF(I,J).EQ.6) GO TO 300
DO 1 11=12,13
DO 2 J1=J2,J3
1 CALL CONDUC(11,J1)
2 CONTINUE
IT=IT+1
GO TO 500
200 IT=1
DO 210 11=12,13
DO 220 J1=J2,J3
XKVAL=30.0
IF(IF(I1,J1).NE.6) CALL CONDUC(I1,J1)
220 CONTINUE
GO TO 500
300 IT=1
DC 310 11=12,13
349

4460.  DO 320 J1=J2,J3
4461.  XKVAL=30.0
4462.  IF(IT(I,J1).NE.5) CALL CONDUC(I1,J1)
4463.  COND(IT)=XKVAL
4464.  IT=IT+1
4465.  320 CONTINUE
4466.  310 CONTINUE
4467.  500 IT=1
4468.  A=0.0
4469.  B=0.0
4470.  DO 20 I1=I2,I3
4471.  DO 21 J1=J2,J3
4472.  A=A+COND(IT)*TEMP(I1,J1)/FAC(IT)
4473.  B=B+COND(IT)/FAC(IT)
4474.  IT=IT+1
4475.  21 CONTINUE
4476.  20 CONTINUE
4477.  GO TO 1000
4478.  100 I2=I-1
4479.  I3=I+1
4480.  J2=J-1
4481.  J3=J+1
4482.  CALL CONDUC(I2,J)
4483.  CALL CONDUC(I1,J2)
4484.  CALL CONDUC(I1,J3)
4485.  CALL CONDUC(I3,J)
4486.  CALL CONDUC(I4,J)
4487.  CALL CONDUC(I5,J)
4488.  A=COND(I1)*TEMP(I2,J1)+(COND(I2)*TEMP(I1,J2))*(COND(I3)*TEMP(I1,J3)) +
4489.  XCOND(I4)*TEMP(I3,J1)+(COND(I5)*TEMP(I1,J1))
4490.  B=COND(I1)+COND(I2)+COND(I3)+COND(I4)+COND(I5)
4491.  XVAL=TEMP(I1,J)-(FACT*TEMP(I1,J)*B)+(FACT*A)
4492.  GO TO 1000
4493.  RETURN
4494.  END
4495.  SUBROUTINE SPECFC(IN,JN)
4496.  COMMON /T/T(80,80)
4497.  INTEGER*2 T
4498.  COMMON /TEMP/TEMP(I,80,80)
4499.  COMMON /SPEC/SPEC
4500.  I=IN
4501.  J=JN
4502.  IF(IT(I,J).NE.1) GO TO 500
4503.  IF(TEMP(I,J).GT.800.0) GO TO 100
4504.  SPEC=.118+.00005*TEMP(I,J)-200.0)
4505.  GO TO 1000
4506.  IF(TEMP(I,J).GT.2730.0) GO TO 505
4507.  SPEC=.260+.000015*(TEMP(I,J)-800.0)
4508.  GO TO 1000
4509.  100 SPEC=.260+.000015*(TEMP(I,J)-800.0)
4510.  GO TO 1000
4511.  IF(TEMP(I,J).GT.2730.0) GO TO 500
4512.  IF(TEMP(I,J).GT.2685.0) GO TO 500
4513.  IF(TEMP(I,J).GT.2640.0) GO TO 500
4514.  IF(TEMP(I,J).GT.1800.0) GO TO 500
4515.  IF(TEMP(I,J).GT.1800.0) GO TO 500
4516.  IF(TEMP(I,J).GT.800.0) GO TO 500
4517.  100 SPEC=.118+.00005*(TEMP(I,J)-200.0)
4518.  GO TO 1000
4519.  550 SPEC=.148+.000148*(TEMP(I,J)-800.0)
GO TO 1000
560 SPEC=.222-.00108*(TEMP(I,J)-1300.0)
GO TO 1000
570 SPEC=.168+.0000381*(TEMP(I,J)-1800.0)
GO TO 1000
580 SPEC=.20-+.0579*(TEMP(I,J)-2640.0)
GO TO 1000
590 SPEC=.28-0.0576*(TEMP(I,J)-2685.0)
GO TO 1000
600 SPEC=.21
1000 IF(IT(I,J).EQ.3) SPEC=SPEC*3.0
RETURN
END

SUBROUTINE CONDUCT(IN,JN)
COMMON /T/T(80,80)
INTEGER*2 T
COMMON /TEMP/TEMP(80,80)
COMMON /XKVAL/XKVAL

I=IN
J=JN
IF(IT(I,J).NE.1) GO TO 500

IF(TEMP(I,J).GT.2730.0) GO TO 550
IF(TEMP(I,J).GT.2640.0) GO TO 560
IF(TEMP(I,J).GT.1600.0) GO TO 570

XKVAL=.38171-.0000669*TEMP(I,J)+0.000000138*1

XKVAL=XKVAL*10.0
GO TO 1000

500 XKVAL=15.0
GO TO 800

550 XKVAL=18.0-.03333*(TEMP(I,J)-2640.0)
GO TO 800

560 XKVAL=18.0-.03333*(TEMP(I,J)-2640.0)
GO TO 800

570 XKVAL=17.0+.0009615*(TEMP(I,J)-1600.0)
GO TO 800

800 IF(IT(I,J).EQ.3) XKVAL=XKVAL*3.0
IF(IT(I,J).EQ.0) XKVAL=100.0
1000 RETURN

END

SUBROUTINE SOLKEY
COMMON /TEMP/TEMP(80,80)
COMMON /NOKEY/NOKEY
COMMON /IKEY/IKEY(40)
COMMON /JKEY/JKEY(40)
COMMON /XTIME/XTIME
COMMON /SFLAG/SFLAG
INTEGER*2 SFLAG
SFLAG=1
DO 100 I=1,NOKEY
IF(TEMP(IKEY(I),JKEY(I)).GT.FREEZE) GO TO 200
SFLAG=2
100 CONTINUE
RETURN

DO 200 SFLAG=1
200 CONTINUE
RETURN

END

SUBROUTINE OUTTEMP(ITER)
COMMON /TEMP/TEMP(80,80)
COMMON /T/T(80,80)
INTEGER*2 T
COMMON /I/start/I/start
COMMON /I/end/I/end
COMMON /J/start/J/start
COMMON /J/end/J/end
COMMON /A/p/A/p(27)
LOGICAL*1 A/p
DIMENSION P(80)
LOGICAL*1 P
COMMON /L/temp/L/temp(30)
COMMON /X/time/X/time
WRITE(6,3)
3 FORMAT(‘---’, ‘) 
WRITE(6,2)
WRITE(6,3)
2 FORMAT(‘---’, 55X, ‘TEMPERATURE PROFILE’) 
WRITE(6,4)
TIME=IERT*X/IEM 
WRITE(6,5) ITERT, X/IEM 
WRITE(6,6) 
WRITE(6,7) 
WRITE(6,8) 
WRITE(6,9)
5 FORMAT(‘---’, ‘CASTING TEMPERATURE PROFILE’) 
WRITE(6,3)
DO 100 I=ISTART, IEND
DO 100 J=JSTART, JEND
IF(T(I,J).EQ.O) GO TO 140
IF(T(I,J).EQ.1) GO TO 140
IF(T(I,J).EQ.6) GO TO 140
DO 120 L=1,26
IF(TEMP(I,J).LE.L/temp(L).AND.TEMP(I,J).GT.L/temp(L+1)) GO TO 130
120 CONTINUE
130 PI(J)=A/p(L)
140 PI(J)=A/p(27)
110 CONTINUE
WRITE(6,200) PI(J), J=JSTART, JEND
200 FORMAT(‘---’, 30A1)
100 CONTINUE
WRITE(6,99)
99 FORMAT(‘---’, ‘) 
WRITE(6,3)
WRITE(6,98)
98 FORMAT(‘---’, ‘SAND TEMPERATURE PROFILE’) 
WRITE(6,3)
DO 300 I=ISTART, IEND
DO 300 J=JSTART, JEND
IF(T(I,J).EQ.2) GO TO 340
IF(T(I,J).EQ.3) GO TO 340
IF(T(I,J).EQ.4) GO TO 340
320 CONTINUE
330 PI(J)=A/p(L)
310 CONTINUE
WRITE(6,200) PI(J), J=JSTART, JEND
300 CONTINUE
320 CONTINUE
CALL OUTVAR
RETURN
END
SUBROUTINE OUTINT(ITERT)
COMMON /NOKEY/NOKEY
COMMON /IKEY/IKEY(40)
COMMON /JKEY/JKEY(40)
COMMON /TEMP/TEMP(80,80)
COMMON /XTIME/XTIME
TIME=ITERT*XTIME
WRITE(6,1)
1 FORMAT('=',41X,'KEY COMPONENT TEMPERATURES - CASTING/RISER CONTACT X')
WRITE(6,20)
20 FORMAT(' ',' 20X,'KEY COMPONENT TEMPERATURES ARE AS FOLLOWS')
WRITE(6,21)
21 FORMAT(' ',' ') DO 30 I=1,NOKEY
WRITE(6,40) I,I  KEY(I),JKEY(I),TEMP(IKEY(I),JKEY(I))
40 FORMAT('  30X,'COMPONENT NUMBER = ',13,' I  = *,I4,'J = ',14,' TEMPERATURE = ',F12.6)
30 CONTINUE
RETURN
END
SUBROUTINE SHKSIZ
COMMON /ISTART/ISTART
COMMON /IEND/IEND
COMMON /JSTART/JSTART
COMMON /JEND/JEND
COMMON /T/T(80,80)
INTEGER*2 T
COMMON /ALPHA/ALPHA
COMMON /ISENS/ISENS
COMMON /TEMP/TEMP(80,80)
COMMON /SHRINK/SHRINK
COMMON /FREEZE/FREEZE
SHRINK=0.0
DO 10 I=ISTART,IEND
10 DO 20 J=JSTART,JEND
IF(T(I,J).EQ.0) GO TO 20
IF(T(I,J).EQ.1) GO TO 20
IF(T(I,J).EQ.3) GO TO 20
IF(T(I,J).EQ.4) GO TO 20
IF(T(I,J).EQ.6) GO TO 20
SHRINK=0.0
DO 10 I=ISTART,IEND
10 DO 20 J=JSTART,JEND
IF(T(I,J).EQ.0) GO TO 20
IF(T(I,J).EQ.1) GO TO 20
IF(T(I,J).EQ.3) GO TO 20
IF(T(I,J).EQ.4) GO TO 20
IF(T(I,J).EQ.6) GO TO 20
IF(TEMP(I,J).LT.FREEZE) GO TO 20
SHRINK=SHRINK+1.0
20 CONTINUE
10 CONTINUE
SHRINK=SHRINK/(ISENS*ISENS)
SHRINK=SHRINK*ALPHA
RETURN
END
SUBROUTINE OUTSLD(ITERT)
COMMON /TEMP/TEMP(I,80,80)
COMMON /T/T(80,80)
INTEGER*2 T
DIMENSION P(80)
LOGICAL*1 P
COMMON /ISTART/ISTART
COMMON /IEND/IEND
COMMON /JSTART/JSTART
COMMON /JEND/JEND
COMMON /ISENS/ISENS
COMMON /FREEZE/FREEZE
COMMON /XTIME/XTIME
COMMON /ALP/ALP(27)
LOGICAL*1 ALP
COMMON /SHRINK/SHRINK
TIME=XTIME*ITERT
WRITE(6,2)
2 FORMAT('FINAL CASTING SOLIDIFICATION - SHRINKAGE CAVITY')
WRITE(6,1) ITERT,TIME
1 FORMAT('-','ITERATION NUMBER = ',14,' TIME = ',F12.3,' CASTING IS X SOLID ')
WRITE(6,3)
3 FORMAT(3X,'GO 10 I=ISTART,IEND
GO 20 J=JSTART,JEND
IF(TEMP(I,J).NE.2.AND.T(I,J).NE.5) GO TO 30
IF(TEMP(I,J).LT.FREEZE) GO TO 40
P(J)=ALP(26)
GO TO 20
40 P(J)=ALP(27)
GO TO 20
30 IF(TEMP(I,J).EQ.0) P(J)=ALP(15)
IF(TEMP(I,J).EQ.1) P(J)=ALP(21)
IF(TEMP(I,J).EQ.3) P(J)=ALP(3)
IF(TEMP(I,J).EQ.4) P(J)=ALP(13)
IF(TEMP(I,J).EQ.5) P(J)=ALP(26)
IF(TEMP(I,J).EQ.6) P(J)=ALP(2)
GO TO 20
WRITE(6,50) (P(J),J=JSTART,JEND)
50 FORMAT(30X,'CODE')
10 CONTINUE
WRITE(6,60)
WRITE(6,10X,'CODE')
WRITE(6,10X,'ALP(15')
WRITE(6,10X,'ALP(21')
WRITE(6,10X,'ALP(3')
WRITE(6,10X,'ALP(13')
WRITE(6,10X,'ALP(26')
WRITE(6,10X,'ALP(2')
WRITE(6,30X,'IMPLIES AIR BARRIER')
WRITE(6,30X,'IMPLIES SAND')
WRITE(6,30X,'IMPLIES SOLID CASTING')
64 FORMAT(' ',20X,'AI', 'IMPLIED CHILL MATERIAL')
65 FORMAT(' ',20X,'AI', 'IMPLIED RISER SOLID AREA')
66 FORMAT(' ',20X,'IMPLIED SHRINKAGE CAVEITY LOCATION')
CALOUTVAR
WRITE(6,100) SHRINK
100 FORMAT('SHRINKAGE CAVEITY SIZE =',F10.6)
RETURN
END
SUBROUTINE OUTVAR
COMMON /T/T(180,80)
INTEGER*2 T
COMMON /SENSW/SENSW
INTEGER SENSW
COMMON /SENSL/SENSL
INTEGER SENSL
COMMON /WID/WID(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /POS/POS(10)
INTEGER NOZERO
COMMON /ZVAL/ZVAL(10)
INTEGER*2 ZVAL
COMMON /KEY/KEY(10,2)
WRITE(6,1)
1 FORMAT(' ',57X,'VARIABLE VALUES')
DO 10 I=1,NOZERO
WRITE(6,20) I
20 FORMAT('ZERO - ONE VARIABLE ',12)
WRITE(6,25) ZVAL(I),WID(I),HEIGHT(I),POS(I)
25 FORMAT('ZERO - ONE VALUE = ',13,' WIDTH = ',F10.6,' HEIGHT = ',F10.6,' POSITION = ',F10.6)
10 CONTINUE
RETURN
END
SUBROUTINE OBJVAL
COMMON /COEFW/COEFW(10)
COMMON /COEFH/COEFH(10)
COMMON /COEFP/COEFP(10)
COMMON /COEF/COEF(18,18)
COMMON /COEFZ/COEFZ(10)
DIMENSION XMAT(20),XMATT(20)
OBJ=0.0
DO 10 I=1,NOZERO
OBJ=OBJ+(ZVAL(I)*COEFZ(I))
10 CONTINUE
IF(ZVAL(I).EQ.0) GO TO 20
OBJ=OBJ+COEFW(I)*WID(I)
OBJ=OBJ+COEFP(I)*POS(I)
OBJ=OBJ+COEFH(I)*HEIGHT(I)
10 CONTINUE
355

DO 30 I=1,10
XMAT(I)=0.0
XMAT(I)=0.0
30 CONTINUE

I1=0
DO 100 I=1,NOZERU
I1=I1+1
XMAT(I1)=WID(I)
I1=I1+1
XMAT(I1)=HEIGHT(I)
I1=I1+1
XMAT(I1)=POS(I)
100 CONTINUE

I1=NOZERU*3
DO 110 I=1,11
DO 120 J=1,I1
XMATT(I1)=XMATT(I1)+XMAT(J)*COEF(J, I)
120 CONTINUE
110 CONTINUE
DO 200 I=1,11
OBJ=OBJ+(XMATT(I1)*XMAT(I1))
200 CONTINUE
RETURN
END

SUBROUTINE MAXVAL
COMMON /NOZERO/NOZERO
COMMON /HEIGHT/HEIGHT(I)
COMMON /WID/WID(I)
COMMON /POS/POS(I)
COMMON /ZVAL/ZVAL(I)
COMMON /ZBOUND/ZBOUND(I)
INTEGER ZBOUND
COMMON /ZSIDE/ZSIDE(I)
INTEGER ZSIDE
COMMON /RECLN/RECLN(I)
INTEGER RECLN
COMMON /RECEN/RECLN(I)
COMMON /ZREC/ZREC(I)
INTEGER ZREC
DO 1 I=1,10
WID(I)=0.0
POS(I)=0.0
HEIGHT(I)=0.0
1 CONTINUE
DO 10 I=1,NOZERU
IF(ZVAL(I).EQ.0) GO TO 10
WID(I)=RECHID(ZREC(I))
10 CONTINUE
RETURN
END

SUBROUTINE AVGVAL
COMMON /NOZERO/NOZERO
COMMON /HEIGHT/HEIGHT(I)
COMMON /POS/POS(I)
COMMON /WID/WID(I)
COMMON /RECLN/RECLN(20)
INTEGER RECLEN
COMMON /RECWID/RECWID(20)
INTEGER RECWRID
COMMON /ZVAL/ZVAL(10)
INTEGER ZVAL
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /ZREC/ZREC(10)
INTEGER ZREC
MIDN = 0
MIDN = 0
HEIGHT(N) = 0
10 CONTINUE
DO 20 I=1,NUZERO
   IF(ZVAL(I).EQ.0) GO TO 20
   WID(I) = RECLN/ZREC(I)/3.0
   POS(I) = RECWID/ZREC(I)/3.0
   IF(ZSIDE(I).EQ.1 .OR. ZSIDE(I).EQ.4) MIDN = RECLN/ZREC(I)/3.0
   IF(ZSIDE(I).EQ.1 .OR. ZSIDE(I).EQ.4) POS(I) = RECLN/ZREC(I)/3.0
   HEIGHT(I) = ZBOUND(I)/3.0
20 CONTINUE
RETURN
END
SUBROUTINE DFP
COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT
COMMON /DFCTEV/DFCTEV
INTEGER * 2 DFCTEV
COMMON /IDFLAG/IDFLAG
COMMON /DGBND1/DGBND1
INTEGER * 2 DGBND1
COMMON /DGBND2/DGBND2
INTEGER * 2 DGBND2
COMMON /DAVLGH/DAVLGH
INTEGER * 2 DAVLGH
COMMON /DBL/DBL
INTEGER * 2 DBL
COMMON /DFPITR/DFPITR
INTEGER * 2 DFPITR
DFPITR = 0
CALL OFPSET
IF(IPCONT(I1).EQ.1) CALL OUTDST
CALL DFPSGD
CALL DPPFDG
IF(IPCONT(I2).EQ.1) CALL OUTDO
CALL DPPKST
IF(IPCONT(I3).EQ.1) CALL OUDWST
20 CALL DPPORT
IF(IPCONT(I4).EQ.1) CALL OUTFD
IF(DGBND1.EQ.1) CALL DGBBD1
IF(DGBND2.EQ.1) CALL DGBBD2
IF(IPCONT(I5).EQ.1) CALL OUTFBD
IF(IFCTEV.EQ.1) CALL OFEVAL
IF(IPCONT(I6).EQ.1.AND.OFCTEV.EQ.1) CALL OUEVL
IF(IPCONT(I6).EQ.1.AND.OFCTEV.EQ.1) CALL OUDSVL
DAVLG=1
4919. IF(PCONT(16).EQ.1 .AND. DFCTEV.EQ.1) CALL QDEVVL
4920. IF(PCONT(16).EQ.1 .AND. DFCTEV.EQ.1) CALL QDVSL
4921. CALL OGCLO
4922. IF(DBL.EQ.0) CALL DFGCHK
4923. CALL DGDCMP
4924. IF(PCONT(17).EQ.1) CALL OUTGLO
4925. IF(DFLAG.EQ.0) GO TO 50
4926. IF(DFLAG.EQ.1) GO TO 100
4927. IF(DFLAG.EQ.2) GO TO 200
4928. IF(DFLAG.EQ.3) GO TO 300
4929. IF(DFLAG.EQ.4) GO TO 400
4930. IF(DFLAG.EQ.5) GO TO 500
4931. CALL OERROR
4932. IF(PCONT(18).EQ.1) CALL OUDERR
4933. RETURN
4934. CALL OERROR
4935. IF(PCONT(18).EQ.1) CALL OUDERR
4936. RETURN
4937. CALL OFPRST
4938. IF(PCONT(19).EQ.1) CALL OUDNIT
4939. GO TO 21
4940. CALL OPOPT
4941. IF(PCONT(20).EQ.1) CALL OUDOPT
4942. RETURN
4943. CALL OFPPIN
4944. IF(DFLAG.EQ.5) GO TO 500
4945. IF(PCONT(21).EQ.1) CALL OUDPIN
4946. GO TO 20
4947. CALL OIFEA
4948. IF(PCONT(22).EQ.1) CALL OUDINF
4949. RETURN
4950. END
4951. SUBROUTINE OFPSET
4952. COMMON /PENVAL/PENVAL
4953. COMMON /PENINT/PENINT
4954. COMMON /DFUNC1/DFUNC1
4955. COMMON /FUNCTN/FUNCTN
4956. COMMON /CURSRK/CURSRK
4957. COMMON /CURSR1/CURSR1
4958. PENVAL=PENINT
4959. CALL SINTAL
4960. CALL AVGVAL
4961. CALL DBVAL
4962. CALL TEQVAL
4963. CURSRK=SHRINK
4964. CURSR1=SHRINK
4965. CALL OFPPFN
4966. DFUNC1=FUNCTN
4967. RETURN
4968. END
4969. SUBROUTINE SINTAL
4970. COMMON /S/S(20,20)
4971. DO 1J=1,20
4972. S(I,J)=0.0
4973. CONTINUE
CONTINUE
IT=NOZERO*3
DO 10 I=1,IT
S(I,1)=1.0
10 CONTINUE
RETURN
END

SUBROUTINE OFPVIT
COMMON /ZVAL/ZVAL(10)
INTEGER * 2 ZVAL
COMMON /NOZERO/NOZERO
DO 10 I=1,NOZERO
ZVAL(I)=1
10 CONTINUE
RETURN
END

SUBROUTINE OFPFN
COMMON /OBJ/OBJ
COMMON /PENVAL/PENVAL
COMMON /SHRINK/SHRINK
COMMON /FUNCTN/FUNCTN
FUNCTN=OBJ+PENVAL*(SHRINK**2.0)
RETURN
END

SUBROUTINE OUTOST
COMMON /DFUNC1/DFUNC1
COMMON /FUNCTN/FUNCTN
COMMON /OBJ/OBJ
COMMON /SHRINK/SHRINK
COMMON /PENVAL/PENVAL
COMMON /S/S(20,20)
COMMON /NOZERO/NOZERO
WRITE(6,1)
1 FORMAT(1*,39X,'INITIAL SET-UP FOR DAVIDON-FLETCHER-POWELL ALGORITHM')
10 WRITE(6,2)
2 FORMAT(1*,10X,'S MATRIX')
10 WRITE(6,3)
3 FORMAT(*,6X,* )
10 DO 10 I=1,IT
10 WRITE(6,11) S(I,J),J=1,IT
11 FORMAT(*,20X,*,20X,*,10X,*,10X,*,10X,*,10X,*,10X,F4.1))
20 CONTINUE
20 WRITE(6,100)
100 FORMAT(*,10X,'INITIAL OBJECTIVE FUNCTION')
200 WRITE(6,101) OBJ
201 FORMAT(*,20X,'OBJECTIVE COST FUNCTION ONLY = ',F15.5)
202 WRITE(6,102) SHRINK
203 FORMAT(*,10X,'SHRINKAGE VALUE = ',F15.9)
204 WRITE(6,103) PENVAL
205 FORMAT(*,10X,'PENALTY VALUE = ',E12.5)
206 WRITE(6,104) DFUNC1
207 FORMAT(*,10X,'TOTAL OBJECTIVE COST FUNCTION (PENALTY INCLUDED) =
208 X ',E20.7)
30 CALL OUTVAR
RETURN
END

SUBROUTINE OFPS60
COMMON /SHRINK/SHRINK
5037. COMMON /CURSRK/CURSRK
5038. COMMON /INTGO/TINTGD
5039. INTEGER * 2 TINTGD
5040. COMMON /TSGDW/TSGDW(10)
5041. COMMON /TSGDW/TSGDW(I)
5042. COMMON /TSGDP/TSGDP(10)
5043. COMMON /DFPGSW/DFPGSW(10)
5044. COMMON /DFPGSH/DFPGSH(10)
5045. COMMON /DFPGSP/DFPGSP(10)
5046. COMMON /NOZERO/NOZERO
5047. COMMON /VAL/VAL
5048. COMMON /ZVAL/ZVAL(I)
5049. INTEGER * 2 ZVAL
5050. DO 1 I=1,10
5051. DFPGSW(I)=0.0
5052. DFPGSH(I)=0.0
5053. DFPGSP(I)=0.0
5054. 1 CONTINUE
5055. IF(ITINTGD.EQ.1) GO TO 100
5056. CURSRK=SHRINK
5057. DO 10 I=1,NOZERO
5058. IF(ZVAL(I).EQ.0) GO TO 10
5059. CALL GRADH(I)
5060. DFPGSW(I)=VAL
5061. CALL GRADH(I)
5062. DFPGSH(I)=VAL
5063. CALL GRADP(I)
5064. DFPGSP(I)=VAL
5065. 10 CONTINUE
5066. GO TO 200
5067. 200 RETURN
5074. END
5075. SUBROUTINE GRADP(I)
5076. COMMON /VAL/VAL
5077. COMMON /POS/POS(I)
5078. COMMON /WID/WID(I)
5079. COMMON /ZSIDE/ZSIDE(I)
5080. INTEGER ZSIDE
5081. COMMON /ZREC/ZREC(I)
5082. INTEGER ZREC
5083. COMMON /RECLEN/RECLEN(20)
5084. INTEGER RECLEN
5085. COMMON /RECWID/RECWID(I)
5086. INTEGER RECWID
5087. COMMON /LOCFP/LOCFP(I)
5088. COMMON /SHRINK/SHRINK
5089. COMMON /CURSRK/CURSRK
5090. COMMON /PENLY/PENLY
5091. COMMON /ISENS/ISENS
5092. COMMON /PCONT/PCONT(50)
5093. INTEGER 2 PCONT
5094. COMMON /PENFN2/PENFN2
5095. INTEGER 2 PENFN2
5096. COMMON /PENFN3/PENFN3
5097. INTEGER 2 PENFN3
INTEGER*2 PENFN3
COMMON/VALO/VALO
COMMON/VALP/VALP
IF(IPC0NTI13).EQ.1 WRITE(6,1) I
1 FORMAT(1-,1*POSITION GRADIENT CALCULATION - VARIABLE = ',13)
SLACK=RECCLUDIREC(I))-WID(I)-POS(I)
I=13)
FORMAT(1-,'EVALUATION OF POSITION GRADIENT NUMBER ',13)
POS(I)=POS(I)+1.0
CALL TEST1
POS(I)=REMBER
GRSP=SHRINK-CURSRK
IF(CURSRK.EQ.-1.0.AND.SHRINK.EQ.-1.0) GRSP=0.0
IF(CURSRK.EQ.-1.0.AND.SHRINK.NE.-1.0) GRSP=SHRINK
IF(CURSRK.NE.-1.0.AND.SHRINK.EQ.-1.0) GRSP=(-1.0)*CURSRK
POS1(I)=POS1(I)+1.0
W1(I)=W1(I)-1.0
GRSP=SHRINK-CURSRK
IF(CURSRK.EQ.-1.0.AND.SHRINK.EQ.-1.0) GRSP=0.0
IF(CURSRK.EQ.-1.0.AND.SHRINK.NE.-1.0) GRSP=SHRINK
GO TO 200
CHEK=XSIDE-XT
IF(IPFGR0).EQ.CHEK) GO TO 175
REMBER=XSIDE
POS1(I)=POS1(I)+1.0
W1(I)=WID(I)
GRSP=SHRINK-CURSRK
IF(CURSRK.EQ.-1.0.AND.SHRINK.EQ.-1.0) GRSP=0.0
IF(CURSRK.EQ.-1.0.AND.SHRINK.NE.-1.0) GRSP=SHRINK
GO TO 200
REMBER=POS(I)
POS1(I)=POS1(I)-1.0
CALL TEST1
POS(I)=REMBER
GRSP=SHRINK-CURSRK
IF(CURSRK.EQ.-1.0.AND.SHRINK.EQ.-1.0) GRSP=0.0
IF(CURSRK.EQ.-1.0.AND.SHRINK.NE.-1.0) GRSP=SHRINK
GO TO 200
CALL CBGRAP(1)
VAL=VALO+VALP*GRSP
IF(IPC0NTI13).EQ.1 WRITE(6,100) I
100 FORMAT(1-,1*EVALUATION OF POSITION GRADIENT NUMBER ',13)
POS(I), SLACK, COEF, SHRINK, GRSP,
XCURSRK, VAL
5157. X 'F8.2', GRSP = 'F8.2', CURSRK = 'F8.2', VAL = 'F8.2')
5158. RETURN
5159. END
5160. SUBROUTINE GRAOHII)
5161. COMMON /VAL/VAL
5162. COMMON /HEIGHT/HEIGHT(10)
5163. COMMON /ZBOUND/ZBOUND(10)
5164. INTEGER ZBOUND
5165. COMMON /COEFH/COEFH(10)
5166. COMMON /COEFWH/COEFWH(10)
5167. COMMON /HOLD/HOLD(10)
5168. COMMON /PENP/PEP(10)
5169. COMMON /ISENS/ISENS
5170. COMMON /SHRINK/SHRINK
5171. COMMON /CURSRK/CURSRK
5172. COMMON /PCONT/PCONT(50)
5173. INTEGER 2 PCONT
5174. COMMON /PENFN2/PENFN2
5175. INTEGER 2 PENFN2
5176. COMMON /PENFN3/PENFN3
5177. INTEGER 2 PENFN3
5178. COMMON /OBJ2/OBJ2
5179. INTEGER 2 OBJ2
5180. COMMON /VALV/VALV
5181. COMMON /VALP/VALP
5182. IF(PCONT(I).EQ.1) WRITE(6,1) I
5183. 1 FORMAT(14), 'HEIGHT GRADIENT CALCULATION - VARIABLE = ',I4)
5184. SLACK=ZBOUND(I)-HEIGHT(I)
5185. IF(SLACK.LT.1.0) GO TO 100
5186. REMBER=HEIGHT(I)
5187. HEIGHT(I)=HEIGHT(I)+SLACK
5188. CALL TEST1
5189. CALL TEST1
5190. GRSP=SHRINK-CURSRK
5191. IF(CURSRK.EQ.-1.0.AND.SHRINK.EQ.-1.0) GRSP=0.0
5192. IF(CURSRK.EQ.-1.0.AND.SHRINK.NE.-1.0) GRSP=SHRINK
5193. IF(CURSRK.NE.-1.0.AND.SHRINK.EQ.-1.0) GRSP=(-1.0)*CURSRK
5194. GO TO 200
5195. 100 XI=1.0/ISENS
5196. IF(SLACK.LT.XI) GO TO 150
5197. REMBER=HEIGHT(I)
5198. HEIGHT(I)=HEIGHT(I)+SLACK
5199. CALL TEST1
5200. HEIGHT(I)=REMBER
GRSP=(SHRINK-CURSRK)/SLACK

IF(CURSRK.EQ.-1.0.AND.SHRINK.EQ.-1.0) GRSP=0.0

IF(CURSRK.EQ.-1.0.AND.SHRINK.NE.-1.0) GRSP=SHRINK/SLACK

IF(CURSRK.NE.-1.0.AND.SHRINK.EQ.-1.0) GRSP=(-1.0)*CURSRK/SLACK)

GO TO 200

REMBER=HEIGHT(I)
HEIGHT(I)=HEIGHT(I)-1.0
CALL TEST1
HEIGHT(I)=REMBER

GRSP=(SHRINK-CURSRK)*(-1.0)

IF(CURSRK.EQ.-1.0.AND.SHRINK.EQ.-1.0) GRSP=0.0

IF(CURSRK.EQ.-1.0.AND.SHRINK.NE.-1.0) GRSP=SHRINK*(-1.0)

IF(CURSRK.NE.-1.0.AND.SHRINK.EQ.-1.0) GRSP=CURSRK

CALL OBGRAH(I)
CALL PNGRA
VAL=VALO+VALP*GRSP
IF(PCONT(I)).EQ.1) WRITE(6,300) I
300 FORMAT('I', ' EVALUATION OF HEIGHT GRADIENT FOR VARIABLE NUMBER ', I)

IF(PCONT(I)).EQ.1) WRITE(6,301) HEIGHT(I),WID(I),COEFH(I),COPEW(I)
301 FORMAT('HEIGHT = ',F10.2,' WIDTH = ',F10.2,' COEFH = ',F10.2,' COPEW = ',F10.2)

IF(PCONT(I)).EQ.1) WRITE(6,302) SLACK,SHRINK,CURSRK,PENLTY,GRSP,VA
302 FORMAT('SLACK = ',F10.2,' SHRINK = ',F10.2,' CURSRK = ',F10.2,' PENLTY = ',F10.2,' GRSP = ',F10.2,' VAL = ',F10.2)
RETURN

END

SUBROUTINE GRADW(I)
COMMON /VAL/VAL
COMMON /WID/WID
COMMON /HEIGHT/HEIGHT
COMMON /ZSIDE/ZSIDE
COMMON /ZREC/ZREC
COMMON /RECLN/RECLN
COMMON /RECHN/RECHN
COMMON /SHRINK/SHRINK
COMMON /CURSRK/CURSRK
COMMON /PENLTY/PENLTY
COMMON /ISENS/ISENS
COMMON /PCONT/PCONT
COMMON /PGCNT/PGCNT
COMMON /PENFN2/PENFN2
COMMON /PENFN3/PENFN3
COMMON /PENFN4/PENFN4
COMMON /PENFN5/PENFN5
COMMON /PENFN6/PENFN6

INTEGER*2 PCONT
INTEGER*2 PENFN2
INTEGER*2 PENFN3
INTEGER*2 PENFN4
INTEGER*2 PENFN5
INTEGER*2 PENFN6
COMMON /OBJ1/OBJ1
COMMON /OBJ2/OBJ2
COMMON /OBJ3/OBJ3
COMMON /OBJ4/OBJ4
COMMON /OBJ5/OBJ5

IF(PCONT(I)).EQ.1) WRITE(6,41) I
41 FORMAT('PCONT= ',I2,' PCONT(I) = ',I2)
FORMAT = '1, 'WIDTH GRADIENT CALCULATION - VARIABLE = ' , '14)  
SLACK1 = RECLEN(ZREC(I) ) - WID(I)  
IF(ZSIDE(I).EQ.1 .OR. ZSIDE(I).EQ.4) SLACK1 = RECLEN(ZREC(I) ) - WID(I)  
SLACK2 = RECLEN(ZREC(I) ) - WID(I) - POS(I)  
IF(ZSIDE(I).EQ.1 .OR. ZSIDE(I).EQ.4) SLACK2 = RECLEN(ZREC(I) ) - WID(I) - POS(I)  
XSID1  
SLACK = SLACK1  
SLACK = SLACK2  
REM = WID(I)  
WID(I) = WID(I) + SLACK  
CALL TEST1  
WID(I) = REM  
GRSP = SHRINK - CURSRK  
IF(CURSRK.EQ.-1.0 .AND. SHRINK.EQ.-1.0) GRSP = 0.0  
IF(CURSRK.EQ.-1.0 .AND. SHRINK.NE.-1.0) GRSP = SHRINK  
IF(CURSRK.NE.-1.0 .AND. SHRINK.EQ.-1.0) GRSP = (-1.0)*CURSRK  
GOTO 200  
100  
REM = WID(I)  
WID(I) = WID(I) - 1.0  
CALL TEST1  
WID(I) = REM  
GRSP = (SHRINK - CURSRK)*1.0  
IF(CURSRK.EQ.-1.0 .AND. SHRINK.EQ.-1.0) GRSP = 0.0  
IF(CURSRK.EQ.-1.0 .AND. SHRINK.NE.-1.0) GRSP = SHRINK  
IF(CURSRK.NE.-1.0 .AND. SHRINK.EQ.-1.0) GRSP = (-1.0)*CURSRK  
GOTO 200  
150  
XSID1 = RECLEN(ZREC(I) )  
IF(ZSIDE(I).EQ.1 .OR. ZSIDE(I).EQ.4) XSID1 = RECLEN(ZREC(I) )  
CHECK = XSID1 - XT  
IF(WID(I).GT.CHECK) GO TO 175  
REM = WID(I)  
REM2 = POS(I)  
WID(I) = WID(I) + 1.0  
POS(I) = POS(I) - 1.0  
CALL TEST1  
WID(I) = REM  
POS(I) = REM  
GRSP = (SHRINK - CURSRK)  
IF(CURSRK.EQ.-1.0 .AND. SHRINK.EQ.-1.0) GRSP = 0.0  
IF(CURSRK.EQ.-1.0 .AND. SHRINK.NE.-1.0) GRSP = SHRINK  
IF(CURSRK.NE.-1.0 .AND. SHRINK.EQ.-1.0) GRSP = (-1.0)*CURSRK  
GOTO 200  
175  
WID(I) = WID(I) - 1.0  
CALL TEST1  
WID(I) = REM  
GRSP = (SHRINK - CURSRK)*(-1.0)  
IF(CURSRK.EQ.-1.0 .AND. SHRINK.EQ.-1.0) GRSP = 0.0  
IF(CURSRK.EQ.-1.0 .AND. SHRINK.NE.-1.0) GRSP = SHRINK  
IF(CURSRK.NE.-1.0 .AND. SHRINK.EQ.-1.0) GRSP = (-1.0)*CURSRK  
GOTO 200  
200  
CALL OBGRAV1  
CALL PNGRA  
VAL = VAL1 + VALP*GRSP  
IF(POCONT(I).EQ.0) WRITE(10,300) I  
300  
FUKMAT1 (-1, 'DETERMINATION OF WIDTH GRADIENT FOR VARIABLE NUMBER ', XIS)  
319  
IF(PCONT(I).EQ.0) GO TO 1000  
350
WRITE(6,301) WID(I),POS(I),HEIGHT(I),COEFW(I),COEFWH(I)
5321. 301 FORMAT(' ',F8.2,' ',F8.2,' ',F8.2,' ',F8.2),C
5322. XCOEFW = ' ',F8.2,' ',COEFWH = ' ',F8.2)
5323. WRITE(6,302) SLACK1,SLACK2,SLACK,SHRINK,CURSRK,GRSP
5324. 302 FORMAT(' ',F8.2,' ',SLACK1= ',F3.3,' SLACK2= ',F3.3,' SLACK= ',F3.3,' X SHRINK= ',F3.3,' CURSRK= ',F8.2,' GRSP = ',F8.2)
5325. WRITE(6,303) VAL
5326. 303 FORMAT(' ',VAL = ',F10.5)
5327. RETURN
5328. END
5329. SUBROUTINE O8GRAP(I)
5330. COMMON /VALO/VALO
5331. COMMON /VALP/VALP
5332. COMMON /C0EFP/COEFP(10)
5333. COMMON /COEFWP/COEFWP(10)
5334. COMMON /COEFPH/CQEFPH(10)
5335. COMMON /WID/WID(I0)
5336. COMMON /HEIGHT/HEIGHT(I0)
5337. COMMON /POS/POS(I0)
5338. VALO=COEFP(I)+2.0*COEFWP(I)*WID(I)+COEFPH(I)*HEIGHT(I)
5339. RETURN
5340. END
5341. SUBROUTINE O8GRAH(I)
5342. COMMON /VALO/VALO
5343. COMMON /COEFH/CQEFH(10)
5344. COMMON /COEFWH/CQEFWH(10)
5345. COMMON /COEFPH/CQEFPH(10)
5346. COMMON /COEFH2/CQEFH2(10)
5347. COMMON /WID/WID(I0)
5348. COMMON /HEIGHT/HEIGHT(I0)
5349. COMMON /POS/POS(I0)
5350. VALO=COEFH(I)+2.0*COEFH2(I)*HEIGHT(I)+COEFWH(I)*WID(I)+COEFPH(I)*POS(I)
5351. RETURN
5352. END
5353. SUBROUTINE O8GRAW(I)
5354. COMMON /VALO/VALO
5355. COMMON /COEFW/CQEFW(10)
5356. COMMON /COEFWR/CQEFWR(10)
5357. COMMON /COEFWH/CQEFWH(10)
5358. COMMON /COEFWP/CQEFWP(10)
5359. COMMON /WID/WID(I0)
5360. COMMON /POS/POS(I0)
5361. COMMON /HEIGHT/HEIGHT(I0)
5362. VALO=COEFW(I)+2.0*COEFWR(I)*WID(I)+COEFWH(I)*HEIGHT(I)+COEFWP(I)*POS(I)
5363. RETURN
5364. END
5365. SUBROUTINE PNGRA
5366. COMMON /CURSRK/CURSRK
5367. COMMON /PENLTY/PENLTY
5368. COMMON /VALP/VALP
5369. VALP=2.0*PENLTY*CURSRK
5370. IF(PENFN2.EQ.1) VALP=2.0*PENLTY*(1+CURSRK)
SUBROUTINE SGRAD
    COMMON /PGRAD/PGRAD(10)
    COMMON /HGRAD/HGRAD(10)
    COMMON /WGRAD/WGRAD(10)
    COMMON /VALO/VALO
    COMMON /VALP/VALP
    COMMON /VAL/VAL
    COMMON /NOZERO/NOZERO
    COMMON /ZVAL/ZVAL
    INTEGER*2 ZVAL
    DO 1 I=1,10
    PGRAD(I)=0.0
    HGRAD(I)=0.0
    WGRAD(I)=0.0
    CONTINUE
    DO 10 I=1,NOZERO
    IF(ZVAL(I).EQ.0 ) GO TO 10
    CALL OBGRAP(I)
    CALL PNGRA
    CALL SKGRAP(I)
    PGRAD(I)=VALO+VALP*VAL
    CALL OBGRAW(I)
    CALL SKGRAW(I)
    HGRAD(I)=VALO+VALP*VAL
    CALL OBGRAH(I)
    CALL SKGRAH(I)
    WGRAD(I)=VALO+VALP*VAL
    10 CONTINUE
    RETURN
END
SUBROUTINE SKGRAP(I)
    COMMON /GRADNT/GRADNT(6,3,5)
    COMMON /BREAK/BREAK(6,3,5)
    COMMON /VAL/VAL
    COMMON /RECLEN/RECLEN
    INTEGER RECLEN
    COMMON /RECD/RECD
    INTEGER RECD
    COMMON /ZREC/ZREC
    INTEGER ZREC
    COMMON /FSIZE/FSIZE
    INTEGER FSIZE
    COMMON /POS/POS
    COMMON /PGRAD/PGRAD(10)
    SLACK=RECLEN(ZREC(I) )
    IF(ZSIDE(1).EQ.1.OR.ZSIDE(1).EQ.4) SLACK=RECLEN(ZREC(I) )
    IF(POS(I).GE.BREAK(I,3,1).AND.POS(I).LT.BREAK(I,3,2)) ISET=1
    IF(POS(I).GE.BREAK(I,3,2).AND.POS(I).LT.BREAK(I,3,3)) ISET=2
    IF(POS(I).GE.BREAK(I,3,3).AND.POS(I).LT.BREAK(I,3,4)) ISET=3
    IF(ISET.EQ.1) VAL=GRADNT(I,3,1)-GRADNT(I,3,2)
    IF(ISET.EQ.2) VAL=GRADNT(I,3,2)-GRADNT(I,3,3)
    IF(ISET.EQ.3) VAL=GRADNT(I,3,3)-GRADNT(I,3,4)
    VAL=VAL/RUN
    RETURN
END
SUBROUTINE SKGRAH(I)
    COMMON /GRADNT/GRADNT(6,3,5)
5440. COMMON /BREAK/BREAK(6,3,5)
5441. COMMON /HEIGHT/HEIGHT(10)
5442. COMMON /VAL/VAL
5443. COMMON /ZBOUND/ZBOUND(10)
5444. INTEGER ZBOUND
5445. RUN=ZBOUND(1)/3.0
5446. IF(HEIGHT(I).GE.BREAK(I,2,1).AND.HEIGHT(I).LT.BREAK(I,2,2)) ISET=1
5447. IF(HEIGHT(I).GE.BREAK(I,2,2).AND.HEIGHT(I).LT.BREAK(I,2,3)) ISET=2
5448. IF(HEIGHT(I).GE.BREAK(I,2,3).AND.HEIGHT(I).LE.BREAK(I,2,4)) ISET=3
5449. IF(ISET.EQ.1) VAL=GRADNT(I,2,1)-GRADNT(I,2,2)
5450. IF(ISET.EQ.2) VAL=GRADNT(I,2,2)-GRADNT(I,2,3)
5451. IF(ISET.EQ.3) VAL=GRADNT(I,2,3)-GRADNT(I,2,4)
5452. IF(ISET.EQ.1) VAL=GRADNT(I,1,1)-GRADNT(I,1,2)
5453. IF(ISET.EQ.2) VAL=GRADNT(I,1,2)-GRADNT(I,1,3)
5454. IF(ISET.EQ.3) VAL=GRADNT(I,1,3)-GRADNT(I,1,4)
5455. VAL=VAL/RUN
5456. RETURN
5457. END
5458. SUBROUTINE SKGRAH(I)
5459. COMMON /BREAK/BREAK(6,3,5)
5460. COMMON /GRADNT/GRADNT(6,3,5)
5461. COMMON /VAL/VAL
5462. COMMON /RECLEN/RECLEN(20)
5463. INTEGER RECLEN
5464. COMMON /RECWID/RECWID(20)
5465. INTEGER RECWID
5466. COMMON /ZSIDE/ZSIDE(10)
5467. INTEGER ZSIDE
5468. COMMON /WID/WID(10)
5469. SLACK=RECWID(ZSIDE(I))
5470. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) SLACK=RECLEN(ZSIDE(I))
5471. RUN=SLACK/3.0
5472. IF(WID(I).GE.BREAK(I,1,1).AND.WID(I).LT.BREAK(I,1,2)) ISET=1
5473. IF(WID(I).GE.BREAK(I,1,2).AND.WID(I).LT.BREAK(I,1,3)) ISET=2
5474. IF(WID(I).GE.BREAK(I,1,3).AND.WID(I).LT.BREAK(I,1,4)) ISET=3
5475. IF(ISET.EQ.1) VAL=GRADNT(I,1,1)-GRADNT(I,1,2)
5476. IF(ISET.EQ.2) VAL=GRADNT(I,1,2)-GRADNT(I,1,3)
5477. IF(ISET.EQ.3) VAL=GRADNT(I,1,3)-GRADNT(I,1,4)
5478. VAL=VAL/RUN
5479. RETURN
5480. END
5481. SUBROUTINE DFPGD
5482. COMMON /DFPGSW/DFPGSW(10)
5483. COMMON /DFPGSH/DFPGSH(10)
5484. COMMON /DFPGS/DFPGS(10)
5485. COMMON /DFPGS/DFPGS(10)
5486. COMMON /DFPGW/DFPGW(10)
5487. COMMON /DFPGW/DFPGW(10)
5488. COMMON /ZVAL/ZVAL(10)
5489. INTEGER * 2 ZVAL
5490. COMMON /COEFW/COEFW(10)
5491. COMMON /COEFH/COEFH(10)
5492. COMMON /COEFPH/COEFPH(10)
5493. COMMON /COEFW/COEFW(10)
5494. COMMON /COEFW/COEFW(10)
5495. COMMON /COEFW/COEFW(10)
5496. COMMON /COEFW/COEFW(10)
5497. COMMON /COEFW/COEFW(10)
5498. COMMON /COEFW/COEFW(10)
5499. COMMON /WID/WID(10)
5500. COMMON /POS/POS(10)
5501. COMMON /HEIGHT/HEIGHT(10)
5502. COMMON /PENVAL/PENVAL
5503. COMMON /CURSR1/CURSR1
5504. DO 1 I=1,10
5505. DFPGW(I)=0.0
5506. DFPGW(I)=0.0
5507. DFPGW(I)=0.0
5508. 1 CONTINUE
5509. DO 10 I=1,10
5510. IF(ZVAL(I).EQ.0) GO TO 10
5511. 
5512. OFPGW(I)=COEF(I)+2.0*COEF2(I)*WID(I)+COEF3(I)*POS(I)+COEF4(I)*CURSR1*
5513. OFPGW(I)=COEF(I)+2.0*COEF2(I)*HEIGHT(I)+COEF3(I)*HEIG
5514. OFPGW(I)=COEF(I)+2.0*COEF2(I)*HEIGHT(I)+COEF3(I)*WID(I)+COEF4(I)*
5515. OFPGW(I)=COEF(I)+2.0*COEF2(I)*HEIGHT(I)+COEF3(I)*WID(I)
5516. CONTINUE
5518. RETURN
5519. END
5520. SUBROUTINE OUTOGO
5521. COMMON /NOZERO/NOZERO
5522. COMMON /ZVAL/ZVAL(I)
5523. INTEGER * 2 ZVAL
5524. COMMON /DFPGSW/DFPGSW(I)
5525. COMMON /OFPGSW/OFPGSW(I)
5526. COMMON /DFPGDH/DFPGDH(I)
5527. COMMON /OFPGDH/OFPGDH(I)
5528. COMMON /DFPGDP/DFPGDP(I)
5529. COMMON /OFPGDP/OFPGDP(I)
5530. COMMON /WID/WID(I)
5531. COMMON /POS/POS(I)
5532. COMMON /HEIGHT/HEIGHT(I)
5533. COMMON /PENVAL/PENVAL
5534. COMMON /SHRINK/SHRINK
5535. COMMON /TYP/TYP(I)
5536. LOGICAL * 1 TYP
5537. WRITE(6,1)
5538. 1 FORMA
5539. WRITE(6,2)
5540. 2 FORMAT(-1,-37X,DAVIDON-FLETCHER-POWELL PENALTY FUNCTION GRADIENT
5541. FORMAT(-1,-37X,DAVIDON-FLETCHER-POWELL PENALTY FUNCTION GRADIENT
5542. WRITE(6,1)
5543. WRITE(6,5) PENVAL
5544. 5 FORMAT(-1,-10X,CURRENT PENALTY VALUE = ',E12.2)
5545. WRITE(6,6) SHRINK
5546. 6 FORMAT(-1,-10X,CURRENT SHRINKAGE VALUE = ',F15.10)
5547. WRITE(6,10)
5548. 10 FORMAT(-1,-37X,ZERO-ONE VARIABLE',2X,2X,TYP',2X,2X,VARIA
5549. 2X,2X,VALUE',2X,2X,SHRINKAGE GRADIENT',2X,2X, PENALTY FUNCTION GRA
5550. 37X, X, E14.7)
5551. DO 100 I=1,10
5552. IF(ZVAL(I).EQ.0) GO TO 100
5553. WRITE(6,101)
5554. 101 FORMAT(-1,-37X,WRITE(6,102)
5555. WRITE(6,103) I,WIDTH(I),DFPGSW(I),DFPGDW(I)
5556. WRITE(6,104) I,HEIGHT(I),DFPGSH(I),DFPGDH(I)
5557. WRITE(6,105) I,POS(I),DFPGDP(I)
5558. WRITE(6,106) I,HEIGHT(I),DFPGFH(I)
5559. 102 FORMAT(-1,-37X,WIDTH ',4X,F8.3,2X,2X,E14.7,2X,2X,2X,2X,5X
5560. X,E14.7)
SUBROUTINE DFPDRT
COMMON /S/S(20,20)
COMMON /OFPGDI/OFPGDI(10)
COMMON /OFPGDO/OFPGDO(10)
COMMON /DFPGDP/DFPGDP(10)
COMMON /NOZERO/NOZERO
INTEGER NOZERO
COMMON /DIREC/DIRECT(20)
DO 1 I=1,20
DIRECT(I)=0.0
1 CONTINUE
II=NOZERO
DO 10 I=1,12
J1=I
J2=J1+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*OFPGDP(J)
10 CONTINUE
J1=J1+NOZERO
J2=J2+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*OFPGDO(J)
DO 20 J=J1,J2
J3=J—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDP(J)
20 CONTINUE
J1=J1—NOZERO
J2=J2—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
DO 30 J=J1,J2
J3=J—NOZERO
Direct(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
30 CONTINUE
J1=J1—NOZERO
J2=J2+NOZERO
DO 40 J=J1,J2
J3=J—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*OFPGDO(J)
40 CONTINUE
J1=J1—NOZERO
J2=J2+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*OFPGDO(J)
DO 50 J=J1,J2
50 CONTINUE
DIRECT(I)=—1.0*DIRECT(I)
50 CONTINUE
RETURN
END

SUBROUTINE DFPDRT
COMMON /S/S(20,20)
COMMON /OFPGDI/OFPGDI(10)
COMMON /OFPGDO/OFPGDO(10)
COMMON /DFPGDP/DFPGDP(10)
COMMON /NOZERO/NOZERO
INTEGER NOZERO
COMMON /DIREC/DIRECT(20)
DO 1 I=1,20
DIRECT(I)=0.0
1 CONTINUE
II=NOZERO
DO 10 I=1,12
J1=I
J2=J1+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*OFPGDP(J)
10 CONTINUE
J1=J1+NOZERO
J2=J2+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*OFPGDO(J)
DO 20 J=J1,J2
J3=J—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDP(J)
20 CONTINUE
J1=J1—NOZERO
J2=J2—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
DO 30 J=J1,J2
J3=J—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
30 CONTINUE
J1=J1—NOZERO
J2=J2+NOZERO
DO 40 J=J1,J2
J3=J—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
40 CONTINUE
J1=J1—NOZERO
J2=J2+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
DO 50 J=J1,J2
50 CONTINUE
DIRECT(I)=—1.0*DIRECT(I)
50 CONTINUE
RETURN
END

SUBROUTINE DFPDRT
COMMON /S/S(20,20)
COMMON /OFPGDI/OFPGDI(10)
COMMON /OFPGDO/OFPGDO(10)
COMMON /DFPGDP/DFPGDP(10)
COMMON /NOZERO/NOZERO
INTEGER NOZERO
COMMON /DIREC/DIRECT(20)
DO 1 I=1,20
DIRECT(I)=0.0
1 CONTINUE
II=NOZERO
DO 10 I=1,12
J1=I
J2=J1+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*OFPGDP(J)
10 CONTINUE
J1=J1+NOZERO
J2=J2+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*OFPGDO(J)
DO 20 J=J1,J2
J3=J—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDP(J)
20 CONTINUE
J1=J1—NOZERO
J2=J2—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
DO 30 J=J1,J2
J3=J—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
30 CONTINUE
J1=J1—NOZERO
J2=J2+NOZERO
DO 40 J=J1,J2
J3=J—NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
40 CONTINUE
J1=J1—NOZERO
J2=J2+NOZERO
DIRECT(I)=DIRECT(I)+S(I,J)*DFPGDO(J)
DO 50 J=J1,J2
50 CONTINUE
DIRECT(I)=—1.0*DIRECT(I)
50 CONTINUE
RETURN
END
10 CONTINUE
I1=I1+NOZERO
I2=I2+NOZERO
DG 2O I=I1,I2
IP=I-NOZERO
WRITE(6,21) IP,DIRECT(I)
21 FORMAT(' ',10X,'VARIABLE ',13,' HEIGHT DIRECTION = ',E12.5)
20 CONTINUE
I1=I1+NOZERO
I2=I2+NOZERO
00 30 I=I1,I2
IP=I-NOZERO-NOZERO
WRITE(6,31) IP,DIRECT(I)
31 FORMAT(' ',10X,'VARIABLE ',13,' WIDTH DIRECTION = ',E12.5)
30 CONTINUE
RETURN
END
SUBROUTINE DGDBD1
COMMON /ALPBND/ALPBND
COMMON /POS/POS(10)
COMMON /WID/WID(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /ZREC/ZRECUO)
INTEGER ZREC
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /RECLEN/RECLEN(20)
INTEGER RECLEN
COMMON /RECWID/RECWID(20)
INTEGER RECWID
COMMON /NOZERO/NOZERO
INTEGER NOZERO
COMMON /DIRECT/DIRECT(20)
COMMON /AVAL/AVAL
COMMON /PCONT/PCONT(50)
INTEGER* 2 PCONT
COMMON /HGTCHK/HGTCHK
INTEGER* 2 HGTCHK
COMMON /NEGCHK/NEGCHK
INTEGER*2 NEGCHK

IF(IPCONT(28).EQ.1) WRITE(6,1)
1 FORMAT ("INTEGER*2 NEGCHK")

ALPBND=100000.0

C WIDTH CONSTRAINT BOUND DETERMINATION

DO 10 I=1,NOZERO

1 FORMAT ("BOUND ON VARIABLE",I4," WIDTH LIMIT CONSTRAINT")

IF(IPCONT(28).EQ.1) WRITE(6,12) WID(I)
12 FORMAT ("CURRENT WIDTH = ",F15.8)

IF(IPCONT(28).EQ.1) WRITE(6,13) DIRECT(I)
13 FORMAT ("CURRENT DIRECTION = ",E14.6)

SLACK1=RECLEN(ZREC(I))/WID(I)

SLACK2=RECWID(ZREC(I))/WID(I)

IF(ZSIDE(I).EQ.0. OR ZSIDE(I).EQ.3) SLACK=SLACK2

IF(ZSIDE(I).EQ.1) WRITE(6,14) SLACK
14 FORMAT ("CURRENT WIDTH = ",F15.8)

IF(DIRECT(I).LT.0.0001) GO TO 25

IF(DIRECT(I).LT.0.0001) GO TO 25

AVAL=SLACK/DIRECT(I)

IF(IPCONT(28).EQ.1) WRITE(6,15) AVAL
15 FORMAT ("BOUND IS DETERMINED")

IF(IPCONT(28).EQ.1) WRITE(6,16) AVAL
16 FORMAT ("BOUND IS LARGE BECAUSE DIRECTION IS SMALL")

20 IF(IPCONT(28).EQ.1) WRITE(6,20) I
20 FORMAT ("BOUND ON VARIABLE",I4," WIDTH LIMIT CONSTRAINT")

IF(IPCONT(28).EQ.1) WRITE(6,21) WID(I)
21 FORMAT ("CURRENT WIDTH = ",F15.8)

IP=I

IV=I+NOZERO+NOZERO

IF(IPCONT(28).EQ.1) WRITE(6,26) I
26 FORMAT ("BOUND IS LARGE BECAUSE DIRECTION IS SMALL")

ALPBND=ALPBND

IF(IPCONT(28).EQ.1) WRITE(6,30) I
30 FORMAT ("BOUND ON VARIABLE",I4," WIDTH LIMIT CONSTRAINT")

DIRECT(I)=DIRECT(I)

IF(IPCONT(28).EQ.1) WRITE(6,34) DIRECT(IP)
34 FORMAT ("COMBINED DIRECTIONS = ",E14.6)

DIRECT(IP)=DIRECT(IP)

DO=DIRECT(I)*DIRECT(IP)

IF(IPCONT(28).EQ.1) WRITE(6,37) I
37 FORMAT ("COMBINED DIRECTIONS = ",E14.6)
SLACK1 = RECLEN(ZREC(I)) - POS(I) - WID(I)
SLACK2 = RECLEN(ZREC(I)) - POS(I) - WID(I)
IF(ZSIDE(I) .EQ. 2 .OR. ZSIDE(I) .EQ. 3) SLACK = SLACK2
IF(ZSIDE(I) .EQ. 1 .OR. ZSIDE(I) .EQ. 4) SLACK = SLACK1
IF(IPCONT(28) .EQ. 1) WRITE(6,38) SLACK
38 FORMAT(' SLACK = ',10X,'F15.8')
IF(D .LT. 0.0) GO TO 40
IF(D .LT. 0.0001) GO TO 50
AVAL = SLACK/D
IF(AVAL .LT. ALPBND) ALPBND = AVAL
IF(IPCONT(28) .EQ. 1) WRITE(6,16) AVAL
GO TO 60
40 IF(IPCONT(28) .EQ. 1) WRITE(6,41)
41 FORMAT(' COMBINED GRADIENTS ARE LESS THAN ZERO THUS BOUND')
GO TO 60
50 IF(IPCONT(28) .EQ. 1) WRITE(6,51)
51 FORMAT(' DIRECTION IS SMALL AND THUS ALLOWS A LARGE BOUND')
60 IF(IPCONT(28) .EQ. 1) WRITE(6,51) AVAL
CONTINUE
C POSITION IS NEGATIVE
00 100 I = 1 .NOZERO
IP = I
IF(IPCONT(28) .EQ. 1) WRITE(6,101) I
101 FORMAT('BOUND ON VARIABLE',14, 'NON-NEGATIVE POSITION')
IF(IPCONT(28) .EQ. 1) WRITE(6,102) POS(I)
102 FORMAT(' CURRENT POSITION = ',14, 'F15.8')
IF(IPCONT(28) .EQ. 1) WRITE(6,103) DIRECT(I)
103 FORMAT(' CURRENT DIRECTION = ',14, 'E14.6')
IF(DIRECT(I) .GT. 0.0) GO TO 120
IF(DIRECT(I) .GT. 0.0001) GO TO 130
IF(NEGCHK .EQ. 1) GO TO 100
AVAL = POS(I) / DIRECT(I) * (-1.0)
IF(AVAL .LT. ALPBND) ALPBND = AVAL
IF(IPCONT(28) .EQ. 1) WRITE(6,16) AVAL
120 IF(IPCONT(28) .EQ. 1) WRITE(6,121)
121 FORMAT(' DIRECTION IS POSITIVE')
GO TO 140
130 IF(IPCONT(28) .EQ. 1) WRITE(6,131)
131 FORMAT(' DIRECTION IS SMALL')
140 IF(IPCONT(28) .EQ. 1) WRITE(6,16,2) ALPBND
CONTINUE
C HEIGHT CONSTRAINT BOUNDS
00 200 I = 1 .NOZERO
IH = I .NOZERO
IF(IPCONT(28) .EQ. 1) WRITE(6,201) I
201 FORMAT('BOUND ON VARIABLE',14, 'HEIGHT LIMITATION CONSTRAINT')
IF(IPCONT(28) .EQ. 1) WRITE(6,202) HEIGHT(I)
202 FORMAT(' CURRENT HEIGHT = ',14, 'F15.8')
IF(IPCONT(28) .EQ. 1) WRITE(6,203) DIRECT(I)
203 FORMAT(' CURRENT DIRECTION = ',14, 'E14.6')
IF(DIRECT(I) .LT. 0.0) GO TO 220
IF(DIRECT(I) .LT. 0.0001) GO TO 230
SLACK = ZBOUND(I) - HEIGHT(I)
220 IF(DIRECT(I) .LT. 0.00001) GO TO 230
230 FORMAT(' SLACK,ZBOUND(I),HEIGHT(I)
IF(NEGCHK.EQ.1) GO TO 240
AVAL=SLACK/DIRECT(IH)

IF(AVAL.LT.ALPBND) ALPBND=AVAL

IF(PCONT(28).EQ.1) WRITE(6,16) AVAL

GO TO 240

220 IF(DIRECT(IH).GT.-0.0001) GO TO 230

IF(NEGCHK.EQ.1) GO TO 240

AVAL=HEIGHT(1)/DIRECT(IH)*(-1.0)

IF(AVAL.LT.ALPBND) ALPBND=AVAL

IF(PCONT(28).EQ.1) WRITE(6,221) AVAL

GO TO 240

230 IF(PCONT(28).EQ.1) WRITE(6,231) AVAL

200 CONTINUE

300 FORMAT(' ',20X,'FINAL BOUND ON UPPER LIMIT IN GOLDEN SECTION = ',F15.8)

AVAL=ALPBND

RETURN

END

SUBROUTINE OGOBD2

COMMON /POS/POS(10)

COMMON /WID/WID(10)

COMMON /HEIGHT/HEIGHT(10)

COMMON /NOZERO/NOZERO

COMMON /ZVAL/ZVAL(10)

INTEGER ZVAL

COMMON /RECLEN/RECLEN(20)

INTEGER RECLEN

COMMON /RECHD/RECHD(20)

INTEGER RECHD

COMMON /DIRECT/DIRECT(20)

COMMON /ZBOUND/ZBOUND(10)

INTEGER ZBOUND

COMMON /ZREC/ZREC(10)

INTEGER ZREC

COMMON /ZSIDE/ZSIDE(10)

INTEGER ZSIDE

COMMON /AVAL/AVAL

COMMON /ALPBND/ALPBND

COMMON /PCONT/PCONT(50)

INTEGER * 2 PCONT

ALPBND=0.0

DO 89 I=1,NOZERO

IF(ZVAL(I).EQ.0) GO TO 89

XSIDE=RECHD(ZREC(I))

IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XSIDE=RECPEN/ZREC(I)

IF(WID(I).LT.XSIDE) GO TO 95

IF(HEIGHT(I).LT.ZBOUND(I)) GO TO 95

CONTINUE

DO 91 I=1,NOZERO

IF(ZVAL(I).EQ.0) GO TO 91

IP=I

IH=I+NOZERO

I=*1+NOZERO+NOZERO
AP=0.0
AH=0.0
AW=0.0
IF(DIRECT(IP).LT.0.0.AND.POS(IP).GT.0.0) AP=POS(IP)/DIRECT(IP)*(-1.0)
IF(DIRECT(IP).LT.0.0) AH=HEIGHT(IP)/DIRECT(IP)*(-1.0)
IF(DIRECT(IP).LT.0.0.AND.POS(IP).GT.0.0) AP=POS(IP)/DIRECT(IP)*(-1.0)
AVAL=AP
IF(AH.GT.AVAL) AVAL=AH
IF(AW.GT.AVAL) AVAL=AW
IF(AVAL.GT.APBND) ALPBND=AVAL
91 CONTINUE
911. GO TO 67
912. 95 00 10 I=1, NOZERO
913. IF(IVAL(I).EQ.0) GO TO 10
914. IF(DIRECT(IP).LT.0.0) GO TO 20
915. XSIDE=RECLEN(IPZREC(I))
916. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XSIDE=RECLEN(IPZREC(I))
917. SLACK=XSIDE-POS(IP)
918. AVAL=SLACK/DIRECT(IP)
919. IF(AVAL.GT.APBND) ALPBND=AVAL
920. GO TO 10
921. 20 AVAL=(POS(IP)/DIRECT(IP))*(-1.0)
922. IF(AVAL.GT.APBND) ALPBND=AVAL
923. 10 CONTINUE
924. DO 100 I=1, NOZERO
925. IF(IVAL(I).EQ.0) GO TO 100
926. IP=I+NOZERO+NOZERO
927. IF(DIRECT(IP).LT.0.0) GO TO 300
928. XSIDE=RECLEN(IPZREC(I))
929. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) XSIDE=RECLEN(IPZREC(I))
930. SLACK=XSIDE-ZSIDE(I)
931. AVAL=SLACK/DIRECT(IP)
932. IF(AVAL.GT.APBND) ALPBND=AVAL
933. GO TO 100
934. 200 AVAL=(WIDTH(IP)/DIRECT(IP))*(-1.0)
935. IF(AVAL.GT.APBND) ALPBND=AVAL
936. 100 CONTINUE
937. DO 300 I=1, NOZERO
938. IF(IVAL(I).EQ.0) GO TO 300
939. IP=I+NOZERO
940. IF(DIRECT(IP).LT.0.0) GO TO 350
941. SLACK=ZBOUND(I)-HEIGHT(I)
942. AVAL=SLACK/DIRECT(IP)
943. IF(AVAL.GT.APBND) ALPBND=AVAL
944. GO TO 300
945. 350 AVAL=(HEIGHT(IP)/DIRECT(IP))*(-1.0)
946. IF(AVAL.GT.APBND) ALPBND=AVAL
947. 300 CONTINUE
948. 67 AVAL=ALPBND
949. RETURN
950. END
951. SUBROUTINE OUTDBD
952. COMMON /AVAL/AVAL
953. WRITE(6,3)
954. 3 FORMAT(1,1,
955. WRITE(6,1)
956. 1 FORMAT(*", THE FOLLOWING IS THE MAXIMUM BOUND ALLOWED SO AS NOT TO LIMIT A CONSTRAINT VIOLATION*)
957. WRITE(6,2) AVAL
958. 2 FORMAT(*", BOUND = ',15.8)
959. RETURN
960. END
5960. RETURN
5961. END
5962. SUBROUTINE DFEVAL
5963. COMMON /AVAL/AVAL
5964. COMMON /DIRECT/DIRECT(20)
5965. COMMON /WID/WIDU01
5966. COMMON /POS/POSU01
5967. COMMON /HEIGHT/HEIGHT(10)
5968. COMMON /NOZERO/NOZERO
5969. COMMON /OBJ/OBJ
5970. COMMON /FUNCTN/FUNCTN
5971. COMMON /F/F(100)
5972. COMMON /XINC/XINC(100)
5973. COMMON /SIDE/SIDE(53)
5974. COMMON /IPOS/IPOS(100)
5975. COMMON /SHK/SHK(100)
5976. COMMON /SHRINK/SHRINK
5977. COMMON /IPOS2/IPOS2(100)
5978. COMMON /SIDE/SIDE(53)
5979. COMMON /DAVLGH/DAVLGH
5980. INTEGER * 2 OAVLGH
5981. DIMENSION RWIDI(101), RPOSI(101), RHTI(101)
5982. XMULT=AVAL/100.0
5983. DO 1 1=1,10
5984. RWIDI(1)=0.0
5985. RPOSI(1)=0.0
5986. RHTI(1)=0.0
5987. 1 CONTINUE
5988. DO 5 I=1,NOZERO
5989. RWIDI(I)=WIDI(I)
5990. RPOSI(I)=POS(I)
5991. RHTI(I)=HEIGHT(I)
5992. 5 CONTINUE
5993. DO 10 I=1,100
5994. XMULT=AVLGH/100.0
5995. DO 20 J=1,NOZERO
5996. IF(DAVLGH.EQ.1) Y=1
5997. IF(DAVLGH.EQ.1) X=XMULT*Y
5998. DO 20 J=1,NOZERO
5999. WIDI(J)=RWIDI(J)
6000. HEIGHT(J)=RHTI(J)
6001. POS(J)=RPOSI(J)
6002. 20 CONTINUE
6003. J1=1
6004. J2=NOZERO
6005. DO 30 J=J1,J2
6006. POS(J)=POS(J)+(X*DIRECT(J))
6007. 30 CONTINUE
6008. J1=J1+NOZERO
6009. J2=J2+NOZERO
6010. DO 40 J=J1,J2
6011. J3=J-NOZERO
6012. HEIGHT(J3)=HEIGHT(J3)+(X*DIRECT(J))
6013. 40 CONTINUE
6014. J1=J1+NOZERO
6015. J2=J2+NOZERO
6016. DO 50 J=J1,J2
6017. J3=J-NOZERO-NOZERO
6018. WIDI(J3)=WIDI(J3)+(X*DIRECT(J))
6019. 50 CONTINUE
CALL OGLCK
CALL OBJVAL
CALL TEQVAL
CALL DFPFTN
F(I)=FUNCTN
XINC(I)=X
SHK(I)=SHRINK
10 CONTINUE
DO 100 I=1,NOZERO
WID(I)=RWMID(I)
HEIGHT(I)=RMT(I)
POS(I)=RPGS(I)
100 CONTINUE
DO 200 I=1,100
IF(F(I).LT.GMIN) GMIN=F(I)
200 CONTINUE
DO 300 I=1,100
IF(F(I).GT.GMAX) GMAX=F(I)
300 CONTINUE
RANGE=GMAX-GMIN
RINC=RANGE/50.0
SIDE(I)=GMIN
DO 400 I=1,100
SIDE(I)=SIDE(I)+RINC
400 CONTINUE
SMAX=-9.0E50
DO 510 I=1,100
IF(SHK(I).GT.SMAX) SMAX=SHK(I)
510 CONTINUE
RANGE=SMAX-SMIN
RINC=RANGE/50.0
SSIDE(I)=SMIN
DO 600 I=1,100
SSIDE(I)=SSIDE(I)+RINC
600 CONTINUE
SSIDE(51)=SMAX
DO 710 J=1,50
IF(SHK(I).GE.SSIDE(J).AND.SHK(I).LE.SSIDE(J1)) GO TO 705
710 CONTINUE
705 IGP0S2(I)=J
700 CONTINUE
RETURN
SUBROUTINE OUDEVL
COMMON /F/F(100)
COMMON /XINC/XINC(100)
COMMON /SIDE/SIDE(5)
COMMON /IGPOS/IGPOS(100)
COMMON /ALP/ALP(27)
LOGICAL * 1 ALP
DIMENSION P(100)
LOGICAL * 1 P
DIMENSION XLOW(3,100)
INTEGER * 2 XLOW
WRITE(6,1)
1 FORMAT(1',37X,'PENALTY FUNCTION VERSUS ALPHA - DAVIDON-FLETCHER-P
XQWELL '\)
WRITE(6,2)
2 FORMAT(6X,'FUNCTION RANGE')
WRITE(6,3)
3 FORMAT(12.5,P.1T P = 2,2)
20 CONTINUE
DO 10 J=1,100
P(J)=ALP(27)
CONTINUE
DO 30 J=1,100
ILK=51-I
IF(IGPOS(J).EQ.ILK).P(J)=ALP(24)
30 CONTINUE
WRITE(6,50) SIDE(11),SIDE(12),(P(J),J=1,100)
50 FORMAT(12.5,'P=1.5,P='1.5,2X,100A1)
10 CONTINUE
ICT=0
DO 100 J=1,10
ICT=ICT*10+J
XLOW(1,ICT)=J
XLOW(2,ICT)=(I-1)
10 CONTINUE
IF(J.EQ.10) XLOW(1,ICT)=0
IF(J.EQ.10) XLOW(2,ICT)=1
120 CONTINUE
DO 130 J=1,10
XLOW(3,ICT)=0
CONTINUE
130 CONTINUE
WRITE(6,131) XLOW(I,J),J=1,100
131 FORMAT(12.5,'P=1.5,P='1.29X,100F11.11)
130 CONTINUE
WRITE(6,132)
132 FORMAT(12.5,'P=1.29X,'-----------------------------
X---------------------------------------------')
133 FORMAT(12.5,'P=1.29X,'-----------------------------')
134 CONTINUE
WRITE(6,133) XLOW(3,J),J=1,100
135 CONTINUE
133 FORMAT(12.5,'P=1.29X,100F11.11)
RETURN
END
SUBROUTINE OUDSVL
COMMON /SHK/SHK(100)
COMMON /XINC/XINC(100)
COMMON /SSIDE/SSIDE(100)
COMMON /IGPOS2/IGPOS2(100)
COMMON /ALP/ALP(27)
LOGICAL * 1 ALP
DIMENSION P(100)
LOGICAL * 1 P
DIMENSION XLOW(3,100)
INTEGER * 2 XLCM

WRITE(6,1)
1 FORMAT(*10,36X,'SHRINKAGE FUNCTION VERSUS ALPHA - DAVIDON-FLETCHER X-POWELL*)
2 FORMAT(*10,6X,'SHRINKAGE FUNCTION RANGE*)
WRITE(6,3)
3 FORMAT(' ',6X,' I',12X,' J',12X,' P(I,J)')
DO 10 I=1,50
10 CONTINUE
DO 20 J=1,100
20 CONTINUE
DO 30 J=1,100
ILK=51-I
IF(IGPOS2(I,J).LT.ILK) P(I,J)=ALP(27)
30 CONTINUE
WRITE(6,50) SSIDE(I),SSIDE(J),P(I,J),J=1,100
50 FORMAT(*10,12.5,' - ',12.5,2X,100A1)
10 CONTINUE
ICT=0
DO 100 I=1,10
100 CONTINUE
DO 110 J=1,10
110 CONTINUE
ICT=(I-1)*10)+J
XLOW(1,ICT)=J
XLOW(2,ICT)=(I-1)
IF(J.EQ.10) XLOW(1,ICT)=0
IF(J.EQ.10) XLOW(2,ICT)=1
120 CONTINUE
DO 130 J=1,100
130 CONTINUE
XLOW(3,ICT)=0
XLOW(1,100)=J
XLOW(2,100)=0
DO 130 I=1,12
DO 130 J=1,12
WRITE(6,131) XLOW(I,J),J=1,100
131 FORMAT(*',29X,100I1)
132 CONTINUE
WRITE(6,132)
132 FORMAT(*',29X,' ')
XLOW(3,133)=(I-1,2)
WRITE(6,133) (XLOW(I,J),J=1,100)
133 FORMAT(*',29X,100I1)
RETURN
END
SUBROUTINE DGOLO
COMMON /DIRECT/DIRECT(20)
COMMON /POS/POS(10)
COMMON /WID/WID(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /NOZERO/NOZERO
COMMON /VAL1/VAL1
COMMON /VAL2/VAL2
COMMON /DVAL/DVAL
COMMON /UDVAL/UDVAL
COMMON /DGSTOP/DGSTOP
COMMON /DGOSTP/DGOSTP
COMMON /B/B
COMMON /U/U
COMMON /SHRNK1/SHRNK1
COMMON /SHRNK2/SHRNK2
COMMON /XLAM1/XLAM1
COMMON /XLAM2/XLAM2
COMMON /OPT/OPT
COMMON /DPH/DPH
COMMON /DBL/DBL
INTEGER * 2 OBL
COMMON /DOPT/DOPT
REAL * 8 DDP
INTEGER * 2 OBL
COMMON /DOPT/DOPT
REAL * 8 DDP
COMMON /DGDITR/DGDITR
INTEGER OGDITR
GDITR=1
DBL=0
CALL DSAVE1
CALL OFPASAV
B=0.0
U=AVAL
IFLAG=0
SG=DGSTOP
IF(AVAL.LT.0.005) CALL DBLGLD
IF(AVAL.LT.0.005) DBL=1
IF(AVAL.LT.0.005) OPT=DPH
IF(AVAL.LT.0.005) GO TO 500
IF(AVAL.LT.2.0) GO TO 1
IFLAG=1
DIFF=100
IF(AVAL.LE.10.0) GO TO 100
IF(AVAL.LE.10.0) GO TO 100
X=9
X=9
X=9
DIFF=XLAM2-XLAM1
IF(DIFF.LE.0.0) GO TO 100
6241. \( U = XLAM2 \)
6242. \( XLAM2 = XLAM1 \)
6243. \( B = B \)
6243.1 \( OGDITR = OGDITR + 1 \)
6244. \( VAL2 = VAL1 \)
6245. \( SHRINK2 = SHRINK1 \)
6246. \( OPT = XLAM1 \)
6247. \( OPTSHK = SHRINK1 \)
6248. \( XLAM1 = 0.382(U - B) + B \)
6249. \( CALL QGOLD1 \)
6250. \( GO TO 10 \)
6251. \( \circ \)
6252. \( XLAM1 = XLAM2 \)
6253. \( U = U \)
6254. \( VAL1 = VAL2 \)
6254.1 \( OGDITR = OGDITR + 1 \)
6255. \( SHRINK1 = SHRINK2 \)
6256. \( OPT = XLAM2 \)
6257. \( OPTSHK = SHRINK2 \)
6258. \( XLAM2 = 0.613(U - B) + B \)
6259. \( CALL QGOLD2 \)
6260. \( GO TO 10 \)
6261. \( 500 \)
6261. \( OGDSTOP = SAV \)
6262. \( RETURN \)
6263. \( END \)
6264. \( SUBROUTINE OFPSAV \)
6265. \( COMMON / WI/WID(10) \)
6266. \( COMMON / POS/POS(10) \)
6267. \( COMMON / HEIGHT/HEIGHT(10) \)
6268. \( COMMON / NOZERO/NOZERO \)
6269. \( COMMON / DSAVMD/DSAVMD(10) \)
6270. \( COMMON / DSAVHT/DSAVHT(10) \)
6271. \( COMMON / DSAVPS/DSAVPS(10) \)
6272. \( COMMON / CURSR1/CURSR1 \)
6273. \( COMMON / DFUNCL/DFUNCL \)
6274. \( COMMON / DSAVSK/DSAVSK \)
6275. \( COMMON / DSAVFC/DSAVFC \)
6276. \( DO 10 I = 1, NOZERO \)
6277. \( DSAVMD(I) = WID(I) \)
6278. \( DSAVHT(I) = HEIGHT(I) \)
6279. \( DSAVPS(I) = POS(I) \)
6280. \( 10 CONTINUE \)
6281. \( DSAVSK = CURSR1 \)
6282. \( DSAVFC = DFUNCL \)
6283. \( RETURN \)
6284. \( END \)
6285. \( SUBROUTINE QGOLD1 \)
6286. \( COMMON / XLAM1/XLAM1 \)
6287. \( COMMON / POS/POS(10) \)
6288. \( COMMON / HEIGHT/HEIGHT(10) \)
6289. \( COMMON / WID/WID(10) \)
6290. \( COMMON / DIRECT/DIRECT(20) \)
6291. \( COMMON / OGDND2/OGDND2 \)
6292. \( INTEGER * 2 OGDND2 \)
6293. \( COMMON / NOZERO/NOZERO \)
6294. \( COMMON / VAL1/VAL1 \)
6295. \( COMMON / FUNCTION/FUNCTION \)
6296. \( COMMON / SHRINK/SHRINK \)
6297. \( COMMON / SHRINK1/SHRINK1 \)
CALL DGET2
DO 10 I=1,NOZERO
I1=I
POS(I)=POS(I)+(XLAM1*DIRECT(I1))
I2=I+NOZERO
HEIGHT(I)=HEIGHT(I)+(XLAM1*DIRECT(I2))
I3=I+NOZERO+NOZERO
WID(I)=WID(I)+(XLAM1*DIRECT(I3))
10 CONTINUE
IF(DGBND2.EQ.1) CALL DGLDCK
CALL TEQVAL
SHRNK1=SHRINK
CALL OBJVAL
CALL DFPPFTN
VAL1=FUNCTN
RETURN
END
SUBROUTINE DGLDCK
COMMON /XLAM2/XLAM2
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /WID/WID(10)
COMMON /DIRECT/DIRECT(20)
COMMON /DGBND2/DGBND2
INTEGER * 2 DGBND2
COMMON /NOZERO/NOZERO
COMMON /VAL2/VAL2
COMMON /FUNCTN/FUNCTN
COMMON /SHRNK2/SHRNK2
COMMON /SHRNK/SHRNK
CALL DGET2
DO 10 I=1,NOZERO
I1=I
POS(I)=POS(I)+(XLAM2*DIRECT(I1))
I2=I+NOZERO
HEIGHT(I)=HEIGHT(I)+(XLAM2*DIRECT(I2))
I3=I+NOZERO+NOZERO
WID(I)=WID(I)+(XLAM2*DIRECT(I3))
10 CONTINUE
IF(DGBND2.EQ.1) CALL DGLDCK
CALL OBJVAL
CALL TEQVAL
SHRNK2=SHRINK
CALL DFPPFTN
VAL2=FUNCTN
RETURN
END
SUBROUTINE DSAVE1
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /NOZERO/NOZERO
COMMON /DREMW1/DREMW1(10)
COMMON /DREM1/DREM1(10)
COMMON /DREMP1/DREMP1(10)
DO 1 I=1,10
DREMW1(I)=0.0
DREM1(I)=0.0
DREMP1(I)=0.0
1 CONTINUE
DO 10 I=1,NOZERO
DREMH1(I)=HEIGHT(I)
DREMP1(I)=POS(I)
10 CONTINUE
RETURN
END

SUBROUTINE DGET2
COMMON /WID/WID(10)
COMMON /PCS/PCS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /NOZERO/NOZERO
COMMON /DREMW1/DREMW1(10)
COMMON /DREMH1/DREMH1(10)
COMMON /DREMP1/DREMP1(10)
DO 1 I=1,10
WID(I)=0.0
HEIGHT(I)=0.0
POS(I)=0.0
1 CONTINUE
DO 10 1=1,NOZERO
WIDE(I)=DREMW1(I)
HEIGHT(I)=DREMH1(I)
POS(I)=DREMP1(I)
10 CONTINUE
RETURN
END

SUBROUTINE DGLDCK
COMMON /WID/WID(10)
COMMON /POS/POS(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /RECLEN/RECLEN(10)
INTEGER RECLEN
COMMON /RECWID/RECWID(10)
INTEGER RECWID
COMMON /NOZERO/NOZERO
COMMON /ZBOUND/ZBOUND(10)
INTEGER ZBOUND
COMMON /ZREC/ZREC(10)
INTEGER ZREC
COMMON /ZSIDE/ZSIDE(10)
INTEGER ZSIDE
DO I=1,NOZERO
IF(WID(I).LE.0.0) WID(I)=0.0
IF(POS(I).LE.0.0) POS(I)=0.0
IF(HEIGHT(I).LE.0.0) HEIGHT(I)=0.0
1 CONTINUE
DO 10 I=1,NOZERO
SLACK1=RECWID(ZREC(I))
SLACK2=RECLEN(ZREC(I))
IF(ZSIDE(I).EQ.1) OR ZSIDE(I).EQ.4) SLACK=SLACK2
IF(ZSIDE(I).EQ.2) OR ZSIDE(I).EQ.3) SLACK=SLACK1
IF(WID(I).LT.SLACK) GO TO 100
WID(I)=SLACK
POS(I)=0.0
10 GO TO 10
   100 X=WID(I)+PCS(I)
   IF(X.LT.SLACK) GO TO 10
   PCS(I)=SLACK-WID(I)
10 CONTINUE
DO 20 I=1,NOZERO
IF(HEIGHT(I).LT.ZBOUND(I)) GO TO 20
HEIGHT(I)=ZBOUND(I)
20 CONTINUE
DO 30 I=1,NOZERO
IF(WID(I).LT.0.0) WID(I)=0.0
IF(POS(I).LT.0.0) POS(I)=0.0
IF(HEIGHT(I).LT.0.0) HEIGHT(I)=0.0
30 CONTINUE
RETURN
END

SUBROUTINE OUTGLD
COMMON /TYP/TYP(10,6)
LOGICAL * 1 TYP
COMMON /WID/WID(10)
COMMON /POS/POSU6)
COMMON /HEIGHT/HEIGHTU6)
COMMON /NOZERO/NOZERO
COMMON /DFUNC1/DFUNC1
COMMON /DFUNC2/DFUNC2
COMMON /OPT/OPT
COMMON /PENVAL/PENVAL
COMMON /CURSR1/CURSR1
COMMON /CURSR2/CURSR2
COMMON /DREM1/DREM1U6)
COMMON /DREM2/DREM2U6)
COMMON /OUTFLG/OUTFLG
INTEGER * 2 OUTFLG
COMMON /DIRECT/DIRECTU6)
COMMON /DFPITR/DFPITR
INTEGER DFPITR
COMMON /DGDITR/DGDITR
INTEGER DGDITR
COMMON /DPNITR/DPNITR
INTEGER DPNITR
DFPITR=DFPITR+1
WRITE16,1)
1 FORMAT('GOLDEN SECTION RESULTS - DAVIDY-FLETCHER-PGWELL')
WRITE(6,2)
2 FORMAT('ZERO-ONE VARIABLE',2X,'TYPE',1X,'NEW VALUE = ORIGIN
XAL VALUE + ■  •,4X,'LAMBOA',5X,'DIRECTION VECTOR')
DO 20 I=1,NOZERO
WRITE(6,19)
19 FORMAT(7X,12,8X,9X,7X,12,8X,9X,7X,12,8X,9X,7X)
DO 20 I=1,NOZERO
WRITE(6,19)
20 CONTINUE
WRITE16,100)
100 FORMAT(*-*,*)
WRITE(6,101)
101 FORMAT(1,'ITERATION COUNTS')
WRITE(6,102) DPNITR
102 FORMAT(1,'ITERATION NUMBER FOR PENALTY UPDATES - ',13)
WRITE(6,103) DPNITR
103 FORMAT(1,'DAVIDON-FLETCHER-POWELL ITERATION NUMBER - ',13)
WRITE(6,104) OGDITR
104 FORMAT(1,'NUMBER OF GOLDEN SECTION ITERATIONS WITHIN DAVIDON-
          X-FLETCHER-POWELL ITERATION - ',13)
WRITE(6,200)
200 FORMAT(*,-20X,4X,'ORIGINAL',4X,'NEW',6X,'DIFFERENCE',3X)
WRITE(6,201)
201 FORMAT(1,'ORIGINAL',4X,'NEW',6X,'DIFFERENCE',3X)
WRITE(6,202) OFUNC1,OFUNC2,DOUT
202 FORMAT(1,'FUNCTIONAL VALUE',4X,1E14.7,1X,1E13.5,1X,2X,1E12.5)
WRITE(6,203) CURSRI,CURSR2,DOUT
203 FORMAT(1,'SHRINKAGE VALUE',5X,1E14.7,1X,1E13.5,1X,2X,1E12.5)
IF(QUTFLG.EQ.1) WRITE(6,300)
300 FORMAT(1,'NOTE ADDITIONAL OPTIMIZATION WAS UTILIZED TO REACH
          OPTIMUM')
DGDTIR=0
RETURN
END
SUBROUTINE DBLGID
COMMON /DU/DU
REAL * 8 DU
COMMON /DB/DB
REAL * 8 DB
COMMON /DXLAM1/DXLAM1
REAL * 8 DXLAM1
COMMON /DXLAM2/DXLAM2
REAL * 8 DXLAM2
COMMON /DVAL1/DVAL1
REAL * 8 DVAL1
COMMON /DVAL2/DVAL2
REAL * 8 DVAL2
COMMON /DOPT/DOPT
REAL * 8 DOPT
COMMON /DOPTSK/DOPTSK
REAL * 8 DOPTSK
COMMON /DSHK1/DSHK1
REAL * 8 DSHK1
COMMON /DSHK2/DSHK2
REAL * 8 DSHK2
COMMON /DVAL/DVAL
COMMON /DDIFF/DDIFF
REAL * 8 DDIFF
COMMON /DBLSTP/DBLSTP
REAL * 8 DBLSTP
CALL DBLSET
CALL DBLSAV
DBLSTP=1.0E-7
DU=AVAL
JS=0.0
DXLAM1=0.382*(DU-DB)+0B
DXLAM2=0.613*(DU-DB)+0B
CALL DBLGID1
CALL DBLGID2
10 DDIFF=DXLAM2-DXLAM1
6537. IF(DDIFF.LE.0.0) DDIFF=DDIFF*(-1.0)
6538. IF(DDIFF.LT.OBLSTP) GO TO 500
6539. IF(DVAL1.GT.DVAL2) GO TO 100
6540. DU=DXLAM2
6541. DXLAM2=DXLAM1
6542. OB=OB
6543. DVAL2=DVAL1
6544. DSHK2=DSHK1
6545. DOPT=DXLAM1
6546. DOPTSX=DSHK1
6547. DXLAM1=0.382*(DU-OB)+OB
6548. CALL OBLGD1
6549. GO TO 10
6550. 100 DB=DXLAM1
6551. DXLAM1=DXLAM2
6552. DU=DU
6553. DVAL1=DVAL2
6554. DSHK1=DSHK2
6555. DOPT=DXLAM2
6556. DOPTSX=DSHK2
6557. DXLAM2=0.618*(DU-OB)+OB
6558. CALL OBLGD2
6559. GO TO 10
6560. 500 CALL OBLGND
6561. RETURN
6562. END
6563. SUBROUTINE OBLSET
6564. COMMON /W/ID/WID10)
6565. COMMON /HEIGHT/HEIGHT10)
6566. COMMON /POS/POS10)
6567. COMMON /DBLWID/DBLWID10)
6568. REAL * 8 DBLWID
6569. COMMON /DBLP/DBLP10)
6570. REAL * 8 OBLPUS
6571. COMMON /OBLHGT/OBLHGT10)
6572. REAL * 8 OBLHGT
6573. COMMON /NOZERO/NOZERO
6574. COMMON /DIRECT/DIRECT20)
6575. COMMON /OBLDRT/OBLDRT20)
6576. REAL * 8 OBLDRT
6577. COMMON /TSGD/TSGD10)
6578. COMMON /TSGD/TSGD10)
6579. COMMON /TSGD/TSGD10)
6580. COMMON /WRSTSK/WRSTSK
6581. COMMON /OBLGDW/OBLGDW10)
6582. REAL * 8 OBLGDW
6583. COMMON /OBLGDH/OBLGDH10)
6584. REAL * 8 OBLGDH
6585. COMMON /OBLGDW/OBLGDW10)
6586. REAL * 8 OBLGDW
6587. COMMON /OBLTSK/OBLTSK
6588. REAL * 8 OBLTSK
6589. COMMON /COEFW/COEFW10)
6590. COMMON /COEFH/COEFH10)
6591. COMMON /COEFP/COEFP10)
6592. COMMON /COEF/I/COEFI10)
6593. COMMON /COEF/COEF10)
6594. COMMON /OBLCFW/OBLCFW10)
6595. REAL * 8 OBLCFW
6596. COMMON /DBLCFH/ DBLCFH(10)
6597. REAL * 8 DBLCFH
6598. COMMON /DBLCFP/ DBLCFP(10)
6599. REAL * 8 DBLCFP
6600. COMMON /DBLCOF/ DBLCOF(18,18)
6601. REAL * 8 DBLCOF
6602. COMMON /DBLCFZ/ DBLCFZ(10)
6603. REAL * 8 DBLCFZ
6604. COMMON /PENVAL/ PENVAL
6605. COMMON /OPEN/ OPEN
6606. REAL * 8 OPEN
6607. DO 1 I=1,10
6608. DBLHIDI(I)=0.0
6609. QBLPOS(I)=0.0
6610. DBLHGT(I)=0.0
6611. DBLGDW(I)=0.0
6612. DBLGOHI(I)=0.0
6613. DBLGOF(I)=0.0
6614. 1 CONTINUE
6615. DO 2 I=1,20
6616. DBLORT(I)=0.0
6617. 2 CONTINUE
6618. DBLTSK=0.0
6619. DO 3 I=1,10
6620. OBLCFH(I)=0.0
6621. OBLCPH(I)=0.0
6622. OBLCFZ(I)=0.0
6623. 3 CONTINUE
6624. DO 4 I=1,18
6625. DO 5 J=1,18
6626. DBLCOF(I,J)=0.0
6627. 5 CONTINUE
6628. 4 CONTINUE
6629. DO 10 I=1,NOZERO
6630. OBLWID(I)=WID(I)
6631. OBLHGT(I)=HEIGHT(I)
6632. OBLPOS(I)=POS(I)
6633. 10 CONTINUE
6634. DO 10 I=1,NOZERO
6635. OBLGDW(I)=TSGDW(I)
6636. OBLGDH(I)=TSGDH(I)
6637. OBLGDOP(I)=TSGDOP(I)
6638. 10 CONTINUE
6639. DBLTSK=WRSTSK
6640. IT=NOZERO+3
6641. DO 30 I=1,IT
6642. DBLORT(I)=DIRECT(I)
6643. 30 CONTINUE
6644. DO 40 I=1,NOZERO
6645. OBLCFH(I)=COEFH(I)
6646. OBLCPH(I)=COEFP(I)
6647. OBLCFZ(I)=COEFZ(I)
6648. 40 CONTINUE
6649. DO 50 I=1,18
6650. DO 51 J=1,13
6651. DBLCOF(I,J)=COEF(I,J)
6652. 51 CONTINUE
6653. 50 CONTINUE
SUBROUTINE DBLGET
COMMON /DBLWID/DBLWID(10)
REAL * 8 DBLWID
COMMON /DBLHGT/DBLHGT(10)
REAL * 8 DBLHGT
COMMON /DBLPOS/DBLPOS(10)
REAL * 8 DBLPOS
COMMON /DBLSWD/DBLSWD(10)
REAL * 8 DBLSWD
COMMON /DBLSHT/DBLSHT(10)
REAL * 8 DBLSHT
COMMON /OBLHGT/OBLHGT(10)
REAL * 8 OBLHGT
COMMON /DBLPOS/OBLPOS(10)
REAL * 8 OBLPOS
COMMON /DBLSW/OBLSW(10)
REAL * 8 OBLSW
COMMON /DBLSHP/OBLSPS(10)
REAL * 8 OBLSPS
COMMON /DBLSHG/DBLSHT(10)
REAL * 8 DBLSHT
COMMON /OBLNO/OBLNO(10)
INTEGER * 2 OBLNO
DO 10 I = 1, OBLNO
IF (OBLNO(I) .EQ. 0) GO TO 10
DBLWID(I) = DBLSWD(I)
DBLHGT(I) = DBLSHT(I)
OBLPOS(I) = OBLPOS(I)
10 CONTINUE
RETURN
END

SUBROUTINE OBLGET
COMMON /OBLWID/OBLWID(10)
REAL * 8 OBLWID
COMMON /OBLHGT/OBLHGT(10)
REAL * 8 OBLHGT
COMMON /OBLPOS/OBLPOS(10)
REAL * 8 OBLPOS
COMMON /OBLSW/OBLSW(10)
REAL * 8 OBLSW
COMMON /OBLSPS/OBLSPS(10)
REAL * 8 OBLSPS
COMMON /OBLSHG/OBLSPS(10)
REAL * 8 OBLSPS
COMMON /OBLNO/OBLNO(10)
INTEGER * 2 OBLNO
DO 10 I = 1, OBLNO
IF (OBLNO(I) .EQ. 0) GO TO 10
DBLSWD(I) = DBLWID(I)
DBLSHT(I) = DBLHGT(I)
OBLPOS(I) = OBLPOS(I)
10 CONTINUE
RETURN
END

SUBROUTINE UBLGET
COMMON /UBLAM1/UBLAM1
REAL * 3 UBLAM1
COMMON /UBLPOS/UBLPOS(10)
REAL * 8 OBLPOS
COMMON /UBLHGT/UBLHGT(10)
REAL * 8 UBLHGT
COMMON /UBLWID/UBLWID(10)
REAL * 8 UBLWID
6716. COMMON /DBLDRT/OBLDRT(20)
6717. REAL * 8 OBLDRT
6718. COMMON /NOZERO/NOZERO
6719. COMMON /OVAL1/OVAL1
6720. REAL * 8 OVAL1
6721. COMMON /OBLFTN/OBLFTN
6722. REAL * 8 OBLFTN
6723. COMMON /DSHK/DSHK
6724. REAL * 8 DSHK
6725. COMMON /DSHK1/DSHK1
6726. REAL * 8 DSHK1
6727. COMMON /DBGN02/DBGN02
6728. INTEGER * 2 DBGN02
6729. CALL OBLGET
6730. DO 10 I=1,NOZERO
6731. I=I
6732. DBLPOS(I)=DBLPOS(I)+OXLAM1*DBLDRT(I1)
6733. I=I+NOZERO
6734. DBLHGT(I)=DBLHGT(I)+OXLAM1*DBLDRT(I2)
6735. I=I+NOZERO
6736. DBLWID(I)=DBLWID(I)+OXLAM1*DBLDRT(I3)
6737. CONTINUE
6738. IF(DBGN02.EQ.1) CALL DBLCHK
6739. CALL DTQVAL
6740. DSHK1=DSHK
6741. CALL DBLOBJ
6742. CALL DBLFUN
6743. OVAL1=OBLFTN
6744. RETURN
6745. END
6746. SUBROUTINE OBLG02
6747. COMMON /DBLG02/OBLG02
6748. REAL * 8 OBLG02
6749. COMMON /DBLP0S/DBLP0S(10)
6750. REAL * 8 DBLP0S
6751. COMMON /DBLHGT/DBLHGT(10)
6752. REAL * 8 DBLHGT
6753. COMMON /DBLWID/DBLWID(10)
6754. REAL * 8 DBLWID
6755. COMMON /DBLDRT/DBLDRT(20)
6756. REAL * 8 DBLDRT
6757. COMMON /NOZERO/NOZERO
6758. COMMON /OVAL2/OVAL2
6759. REAL * 8 OVAL2
6760. COMMON /OBLFTN/OBLFTN
6761. REAL * 8 DBLFTN
6762. COMMON /DSHK/DSHK
6763. REAL * 8 DSHK
6764. COMMON /DSHK2/DSHK2
6765. REAL * 8 DSHK2
6766. COMMON /DG3ND2/DG3ND2
6767. INTEGER * 2 DG3ND2
6768. CALL OBLGET
6769. DO 10 I=1,NOZERO
6770. 11=1
6771. DBLPQStl)=DBLPOSlI J+tOXLAM2*OBLQRTl111)
6772. 12=I+NOZERO
6773. DGLKGT(I)=OBLHGT(I)+(DXLAM2*DBLDRT(I2))
6774. 13=I+NOZERO*NOZERO
6775. D8LWIDII)=DBLWIDII )+I 0XLAM2*0BL0RT(I 3))
6776. 10 CONTINUE
6777. IF(DG3ND2.EQ.1) CALL OBLCHK
6778. CALL OTQVAL
6779. DSHK2=DSHK
6780. CALL OBLOBJ
6781. CALL OBLFUN
6782. OVAL2=DBLFTN
6783. RETURN
6784. END
6785. SUBROUTINE OBLCHK
6786. COMMON /D8LHID/CQLWID(10)
6787. REAL * 8 DBLWID
6788. COMMON /OBLPOS/DBLPOS(10)
6789. REAL * 3 OBLPOS
6790. COMMON /DBLHGT/DBLHGT(10)
6791. REAL * 8 OBLHGT
6792. COMMON /RECLEN/RECLEN(20)
6793. INTEGER RECLEN
6794. COMMON /RECWID/RECWID(20)
6795. INTEGER RECWID
6796. COMMON /NOZERO/NOZERO
6797. COMMON /ZBOUND/ZBOUND(10)
6798. INTEGER ZBOUND
6799. COMMON /ZREC/ZREC(10)
6800. INTEGER ZREC
6801. COMMON /ZSIDE/ZSIDE(10)
6802. INTEGER ZSIDE
6803. REAL * 8 X
6804. DO 10 I=1,NOZERO
6805. IF(10000(I).LT.0.0) DBLWID(I)=0.0
6806. IF(DBLHGT(I).LT.0.0) DBLHGT(I)=0.0
6807. IF(DBLPOS(I).LT.0.0) DBLPOS(I)=0.0
6808. 10 CONTINUE
6809. DO 20 I=1,NOZERO
6810. SLACK1=RECWID(ZREC(I))
6811. SLACK2=RECLEN(ZREC(I))
6812. IF(ZSIDE(I).EQ.1.OR.ZSIDE(I).EQ.4) SLACK=SLACK2
6813. IF(ZSIDE(I).EQ.2.OR.ZSIDE(I).EQ.3) SLACK=SLACK1
6814. IF(DBLHGT(I).LT.SLACK) Go To 100
6815. DBLHGT(I)=SLACK
6816. DBLPOS(I)=0.0
6817. Go To 20
6818. 100 *=DBLWID(I)+DBLPOS(I)
6819. IF(X.LT.SLACK) Go To 20
OBLPOS(I) = SLACK - DBLWID(I)

20 CONTINUE

DO 30 I = 1, NOZERO
6822.
6823. IF (DBLHGT(I) .LT. ZBOUND(I)) GO TO 30
6824.
6825. 30 CONTINUE
6826. RETURN
6827. END

SUBROUTINE DBLOBJ

COMMON /DBLCFW/ DBLCFW(10)
6829. REAL * 8  DBLCFW

COMMON /DBLCFH/ DBLCFH(10)
6830. REAL * 8  DBLCFH

COMMON /DBLCFP/ DBLCFP(10)
6831. REAL * 8  DBLCFP

COMMON /DBLCFZ/ DBLCFZ(10)
6832. REAL * 8  DBLCFZ

COMMON /DBLCF/ DBLCF(18, 18)
6833. REAL * 8  DBLCF

COMMON /DBLWID/ DBLWID(10)
6834. REAL * 8  DBLWID

COMMON /DBLPGS/ DBLPOS(10)
6835. REAL * 8  DBLPOS

COMMON /DBLHGT/ DBLHGT(10)
6836. REAL * 8  DBLHGT

COMMON /NOZERO/ NOZERO
6837. COMMON /DSHK/ DSHK

COMMON /OOSJ/ DOBJ
6838. COMMON /ZVAL/ ZVAL(10)

REAL * 8  XMAT(20), XMAT120)
6839. DOBJ = 0.

DO 10 I = 1, NOZERO
6840.
6841. DOBJ = DOBJ + (ZVAL(I)) * DBLCFZ(I)
6842.
6843. 10 CONTINUE

DO 20 I = 1, NOZERO
6844.
6845. IF (ZVAL(I) .EQ. 0) GO TO 20
6846.
6847. DOBJ = DOBJ + (DBLWID(I)) * DBLCFW(I)
6848.
6849. 20 CONTINUE

DO 30 I = 1, NOZERO
6850.
6851. XMAT(I) = DBLWID(I)
6852.
6853. 30 CONTINUE

II = 0
6854.
6855. GG 100 II = 1, NOZERO
6856.
6857. II = II + 1
6858.
6859. XMAT(I) = DBLWID(I)
6860.
6861. II = II + 1
6862.
6863. XMAT(I) = DBLHGT(I)
6864.
6865. II = II + 1
6866.
6867. XMAT(I) = DBLPOS(I)
6868.
6869. 100 CONTINUE

II = NOZERO + 3
6870.
6871. GG 100 II = 1, 11
6872.
6873. II = II + 1
6874.
6875. XMAT(I) = XMAT(I) + (XMAT(J)) * DBLCF(J, I)
6876.
6877. GG 120 J = 1, 11
CONTINUE
CGO200(I)=XMATT(I)*XMATT(I))
CONTINUE
RETURN
END
SUBROUTINE DTQVAL
    COMMON /NOZERO/NOZERO
    COMMON /ZVAL/ZVAL(I)
    INTEGER * 2 ZVAL
    COMMON /DBLWID/DBLWID(I)
    REAL * 8 DBLWID
    COMMON /DBLPOS/DBLPOS(I)
    REAL * 8 DBLPOS
    COMMON /DLHGT/DBLHGT(I)
    REAL * 8 DBLHGT
    COMMON /OBLGDW/OBLGDW(I)
    REAL * 3 OBLGDW
    COMMON /OBLGDH/OBLGDH(I)
    REAL * 3 OBLGDH
    COMMON /DBLGDP/DBLGDP(I)
    REAL * 8 OBLGD
    REAL * 8 OBLPOS
    COMMON /DBLTSK/DBLTSK
    REAL * 8 OBLTSK
    COMMON /SHK/DH
    REAL * 8 DSHK
    COMMON /SHKTOL/SHKTOL
    DSHK=OBLTSK
    DO 10 I=1,NOZERO
        IF(ZVAL(I).EQ.0) GO TO 10
        DSHK=DSHK+(DBLWID(I)*OBLGDW(I))
        DSHK=DSHK+IDBLHGT(I)*OBLGDH(I))
        DSHK=DSHK+(DBLPOS(I)*OBLGD(I))
    10 CONTINUE
    IF(DSHK.LT.DSHK.TCL) DSHK=0.0
    RETURN
END
SUBROUTINE DBLFUN
    COMMON /DBLFTN/DBLFTN
    REAL * 8 DBLFTN
    COMMON /SHK/DH
    REAL * 8 DSHK
    COMMON /OBLTSK/DBLTSK
    DSHK=OBLTSK
    DO 10 I=1,NOZERO
        IF(ZVAL(I).EQ.0) GO TO 10
        DSHK=DSHK+(DBLWID(I)*OBLGDW(I))
        DSHK=DSHK+IDBLHGT(I)*OBLGDH(I))
        DSHK=DSHK+(DBLPOS(I)*OBLGD(I))
    10 CONTINUE
    IF(DSHK.LT.DSHK.TCL) DSHK=0.0
    RETURN
END
SUBROUTINE DBLEND
    COMMON /NOZERO/NOZERO
    COMMON /OBLN/DBLPOS
    INTEGER * 2 OBLN
    COMMON /POS/POS(I)
    COMMON /HEIGHT/HEIGHT(I)
    COMMON /VFUNC/DBLGDP
    COMMON /CURSR2/DBLTSK
    COMMON /OBLPOS/DBLPOS(I)
    NOZERO
6940. REAL * 8 DBLPOS
6941. COMMON /DBLHGT/DBLHGT(10)
6942. REAL * 8 DBLHGT
6943. COMMON /DBLWID/DBLWID(10)
6944. REAL * 8 OBLPOS
6945. COMMON /DOPT/OOPT
6946. REAL * 8 DOPT
6947. COMMON /DBLRT/DBLRT(20)
6948. REAL * 8 DBLRT
6949. COMMON /DOPTSK/OOPTSK
6950. REAL * 8 DOPTSK
6951. COMMON /DSHK/DSHK
6952. REAL * 8 DSHK
6953. COMMON /DBLFTN/DBLFTN
6954. REAL * 8 DBLFTN
6955. COMMON /SHRINK/SHRINK
6956. CALL DBLGET
6957. DO 10 I=1,1,NOZERO
6958. 1=I
6959. DBLPOS(I)=DBLPOS(I)+(DOPT*DBLRT(I))
6960. I2=I+NOZERO
6961. DBLHGT(I)=DBLHGT(I)+DOPT*DBLRT(I)
6962. I3=I+NOZERO+NOZERO
6963. DBLWID(I)=DBLWID(I)+DOPT*DBLRT(I)
6964. 10 CONTINUE
6965. IF IDGBND2.EQ.1) CALL DBLCHK
6966. DO 20 I=1,10
6967. WID(I)=DBLWID(I)
6968. HEIGHT(I)=DBLHGT(I)
6969. POS(I)=DBLPOS(I)
6970. 20 CONTINUE
6971. CALL OBJ
6972. CURSR2=DOPTSK
6973. SHRINK=OOPTSK
6974. DSHK=OOPTSK
6975. CALL DBLFUN
6976. DFUNC2=DBLFTN
6977. RETURN
6978. END
6979. SUBROUTINE DFGCHX
6980. COMMON /DFUNC1/DFUNC1
6981. COMMON /DFUNC2/DFUNC2
6982. COMMON /CURSR1/CURSR1
6983. COMMON /CURSR2/CURSR2
6984. COMMON /AVLMIN/AVLMIN
6985. COMMON /AVLMAX/AVLMAX
6986. COMMON /AVL/AVL
6987. COMMON /OPT/OPT
6988. COMMON /OPTSHK/OPTSHK
6989. COMMON /IDFLAG/IDFLAG
6990. COMMON /WID/WID(10)
6991. COMMON /HEIGHT/HEIGHT(10)
6992. COMMON /POS/POS(10)
6993. COMMON /NOZERO/NOZERO
6994. COMMON /XLAM1/XLAM1
6995. COMMON /XLAM2/XLAM2
6996. COMMON /DGSTOP/DGSTOP
6997. COMMON /VAL/VAL
6998. COMMON /VAL2/VAL2
6999. COMMON /DIRECT/DIRECT(20)
COMMON /FUNCTN/FUNCTN
COMMON /SHRNL1/SHRNL1
COMMON /SHRNL2/SHRNL2
COMMON /DGBN2/DGBN2
INTEGER * 2 DGBN2
COMMON /OBJ/OBJ
COMMON /SHRNLX/SHRNLX
COMMON /OUTFLG/OUTFLG
INTEGER * 2 OUTFLG
OUTFLG=0
CALL OGENDS
CALL DLMCHK
IF(DFUNC2.LE.DFUNCI) GO TO 500
OUTFLG=1
IF(IOPT.EQ.AVLMIN) GO TO 100
IF(IOPT.EQ.AVLMAX) GO TO 200
GO TO 300
100 B=0.3
U=OPT
DXTRA=DGSTOP/10.0
XLAM1=0.382*(U-B)+B
XLAM2=0.618*(U-B)+B
CALL DGOLD1
CALL DGOLD2
110 DIFF=XLAM2-XLAM1
IF(DIFF.LE.0.0) DIFF=DIFF*(-1.0)
IF(DIFF.GT.DXTRA) GO TO 150
IF(DIFF.LT.DXTRA) GO TO 120
U=XLAM2
XLAM2=XLAM1
B=B
VAL2=VAL1
SHRNL2=SHRNL1
OPT=XLAM1
OPTSHK=SHRNL1
XLAM1=0.382*(U-B)+B
CALL DGOLD1
GO TO 110
120 S=XLAM1
XLAM1=XLAM2
U=U
VAL1=VAL2
SHRNL1=SHRNL2
OPT=XLAM2
OPTSHK=SHRNL2
XLAM2=0.618*(U-B)+B
CALL DGOLD2
GO TO 110
150 CALL OGGET2
GO 160 I=1,NUZERO
160 CONTINUE
IF(DGBN2.EQ.1) CALL DGLCDK
CALL OBJVAL
SHRNLX=OPTSHK
CALL DFPFTN

DFUNCX=FUNCTN

IF(DFUNCX.LT.DFUNC1) DFUNC2=DFUNCX

IF(DFUNCX.LT.DFUNC1) CURSR2=OPTSHK

IF(DFUNCX.LT.DFUNC2) DFUNC2=DFUNCX

IF(DFUNCX.LT.DFUNC2) CURSR2=OPTSHK

IF(DFUNCX.GT.DFUNC2) CALL DCHKBD

RETURN

CALL DGET2

GO TO 210

I=1

IF(DFUNCX.EQ.1) CALL DGLDCK

CALL UBJVAL

CALL TVAL

CALL DPPFTN

DFUNCX=FUNCTN

IF(DFUNCX.LT.DFUNC1) GO TO 280

DEXTA=DSTOP/10.0

U=AVAL

XLAM1=0.382*(U-B)+B

XLAM2=0.618*(U-B)+B

CALL DGOLD1

CALL DGLD2

CALL DGET2

GO TO 220

DIFF=XLAM2-XLAM1

IF(DIFF.LE.0.0) DIFF=DIFF*(-1.0)

IF(DIFF.LE.DEXTRA) GO TO 250

IF(VAL1.LT.VAL2) GO TO 230

U=XLAM2

XLAM2=XLAM1

B=B

VAL2=VAL1

SHRNK2=SHRNK1

OPT=XLAM1

OPTSHK=SHRNK1

XLAM1=0.382*(U-B)+B

CALL DGOLD1

GO TO 220

B=XLAM1

XLAM1=XLAM2

U=U

VAL1=VAL2

SHRNK1=SHRNK2

OPT=XLAM2

OPTSHK=SHRNK2

XLAM2=0.618*(U-B)+B

CALL DGOLD2

GO TO 220

CALL DGET2

GO TO 270

I=1

IF(I.EQ.1) CALL DGLDCK

IF(I.EQ.1) CALL TVAL

IF(I.EQ.1) CALL DPPFTN

DFUNCX=FUNCTN

IF(DFUNCX.LT.DFUNC1) DFUNC2=DFUNCX

IF(DFUNCX.LT.DFUNC1) CURSR2=OPTSHK

IF(DFUNCX.LT.DFUNC2) DFUNC2=DFUNCX

IF(DFUNCX.LT.DFUNC2) CURSR2=OPTSHK

IF(DFUNCX.GT.DFUNC2) CALL DCHKBD

RETURN

CALL DGET2

GO TO 210
7120.  I3=I+NOZERO+NOZERO
7121.  WID(I)=WID(I)+(OPT*DIRECT(I3))
7122.  CONTINUE
7123.  SHRINK=OPTSHK
7124.  CALL OBJVAL
7125.  CALL DFPFTN
7126.  DFUNC=DFUNCTN
7127.  IF(DFUNC.LT.DFUNC1) DFUNC2=DFUNC
7128.  IF(DFUNC.LT.DFUNC1) CURSR2=OPTSHK
7129.  IF(DFUNC.LT.DFUNC2) DFUNC2=DFUNC
7130.  IF(DFUNC.LT.DFUNC2) CURSR2=OPTSHK
7131.  IF(DFUNC.GT.DFUNC2) CALL DCHKBD
7132.  RETURN
7133.  280 GPT=AVAL
7134.  DFUNC2=DFUNC
7135.  CURSR2=SHRINK
7136.  RETURN
7137.  300 ABND=AVAL/10.0
7138.  B=OPT-ABND
7139.  U=OPT-ABND
7140.  IF(B.LT.0.0) 9=0.0
7141.  IF(U.GT.AVAL) U=AVAL
7142.  DEXTRA=QSTOP/10.0
7143.  XLAM1=0.382*(U-B)*B
7144.  XLAM2=0.618*(U-B)*B
7145.  CALL DG0LD1
7146.  CALL DG0LD2
7147.  CALL DG0LD
7148.  DIFF=XLAM2-XLAM1
7149.  IF(DIFF.LE.0.0) DIFF=DIFF+1.0
7150.  IF(DIFF.LT.DEXTRA) GO TO 350
7151.  U=XLAM2
7152.  XLAM2=XLAM1
7153.  B=U
7154.  VAL2=VAL1
7155.  SHRNK2=SHRNK1
7156.  OPT=XLAM1
7157.  OPTSHK=SHRNK1
7158.  XLAM1=0.382*(U-B)*B
7159.  CALL DG0LD1
7160.  GO TO 310
7161.  320 B=XLAM1
7162.  XLAM1=XLAM2
7163.  U=U
7164.  VAL1=VAL2
7165.  SHRNK1=SHRNK2
7166.  OPT=XLAM2
7167.  OPTSHK=SHRNK2
7168.  XLAM2=0.618*(U-B)*B
7169.  CALL DG0LD2
7170.  GO TO 310
7171.  350 CALL DGET2
7172.  DO 360 I=1,NOZERO
7173.  I1=I
7174.  POS(I1)=POS(I1)*(OPT*DIRECT(I1))
7175.  I2=I+NOZERO
7176.  HEIGHT(I1)=HEIGHT(I1)*(OPT*DIRECT(I2))
7177.  I3=I+NOZERO+NOZERO
7178.  WID(I1)=WID(I1)*(OPT*DIRECT(I3))
7179.  CONTINUE
7180.  360 CONTINUE
CALL CEJVAL
SHRINK=OPTSHK
CALL DFPFTN
DFUNCX=FUNCXT
IF (DFUNCX .LT. DFUNC1) DFUNC2 = DFUNCX
IF (DFUNCX .LT. DFUC1) CURSR2 = OPTSHK
IF (DFUNCX .LT. DFUNC2) DFUNC2 = DFUNCX
IF (DFUNCX .LT. DFUC2) CURSR2 = OPTSHK
IF (DFUNCX .GT. DFUNC2) CALL DCHKBD
GO TO 500
RETURN
ENC
SUBROUTINE DCENDS
COMMON /ClPT/OPT
COMMON /OPTSHK/OPTSHK
COMMON /WIDWID10
COMMON /POS/POS10
COMMON /HEIGHT/HEIGHT10
COMMON /NOZERO/NOZERO
COMMON /DIRECT/DIRECT10
COMMON /DFUNC2/DFUNC2
COMMON /DFPSEW/DFPSEW10
COMMON /DFPSEP/DFPSEP10
COMMON /DGNOZ/DGNOZ
INTEGER * 2 DGBND2
COMMON /FUNCXT/FUNCXT
COMMON /CURSR2/CURSR2
CALL DGET2
DO 1 = 1, 10
DFPSEW(I) = 0.0
DFPSEH(I) = 0.0
CONTINUE
DO 10 I = 1, NOZERO
1 = I
POS(I) = POS(I) + (0PT * DIRECT(I))
I = I + NOZERO
HEIGHT(I) = HEIGHT(I) + (0PT * DIRECT(I))
I = I + NOZERO
WID(I) = WID(I) + (OPT * DIRECT(I))
CONTINUE
IF (DGNOZ .EQ. 1) CALL DGLOCK
DO 20 I = 1, NOZERO
DFPSEW(I) = NID(I)
DFPSEH(I) = HEIGHT(I)
DFPSEP(I) = POS(I)
CONTINUE
20 CALL CEJVAL
CALL OJVAL
CURSR2 = OPTSHK
SHINK = OPTSHK
CALL DFPFTN
DFUNC2 = FUNCXT
RETURN
END
SUBROUTINE OLMCHK
COMMON /AVAL/AVAL
COMMON /DCSTCP/DCSTCP
COMMON /AVLMIN/AVLMIN
COMMON /AVLMAX/AVLMAX
8=0.0
U=AVAL
XLAM1=0.382*(U-B)+B
XLAM2=0.618*(U-B)+B
DO 100 I=1,2100
U=XLAM2
XLAM2=XLAM1
8=B
XLAM1=0.382*(U-B)+B
DIFF=XLAM2-XLAM1
IF(DIFF.LE.0.0) DIFF=DIFF*(-1.0)
IF(DIFF.LE.DGSTOP) GO TO 110
100 CONTINUE
AVLMIN=XLAM1
B=0.0
U=AVAL
XLAM1=0.382*(U-B)+B
XLAM2=0.618*(U-B)+B
DO 200 I=1,2100
B=XLAM1
XLAM1=XLAM2
U=U
XLAM2=0.618*(U-B)+B
DIFF=XLAM2-XLAM1
IF(DIFF.LE.0.0) DIFF=DIFF*(-1.0)
IF(DIFF.LE.DGSTOP) GO TO 210
200 CONTINUE
AVLMAX=XLAM2
RETURN
END
SUBROUTINE OCHKBD
COMMON /DFPSEW/DFPSEW(10)
COMMON /DFPSEP/DFPSEP(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /POS/POS(10)
COMMON /NOZERO/NOZERO
COMMON /CURSR2/CURSR2
COMMON /DFUNC2/DFUNC2
DO 10 I=1,NOZERO
7281.     WID(I)=DFPSEW(I)
7282.     HEIGHT(I)=DFPSEH(I)
7283.     POS(I)=DFPSEP(I)
7284.     10 CONTINUE
7285.     DFUNC2=DFUNC2
7286.     CURSR2=CURSR2
7287.     RETURN
7288.     END
7289.     SUBROUTINEDGDCMP
7290.     COMMON /DFUNC1/DFUNC1
7291.     COMMON /DFUNC2/DFUNC2
7292.     COMMON /CURSR2/CURSR2
7293.     COMMON /CURSR1/CURSR1
7294.     COMMON /IDFLAG/IDFLAG
7295.     COMMON /DFPSTP/DFPSTP
7296.     COMMON /DFPDFF/DFPDFF
7297.     COMMON /SHKTOL/SHKTOL
7298.     COMMON /PENVAL/PENVAL
7299.     COMMON /PENMAX/PENMAX
7300.     COMMON /PCONT/PCONT
7301.     COMMON /DSHKVL/DSHKVL
7302.     INTEGER * 2 PCONT
7303.     IF(DFUNC2.GT.DFUNC1) GO TO 500
7304.     DFPDFF=DFUNC1-DFUNC2
7305.     IF(DFPDFF.LT.DFPSTP) GO TO 200
7306.     IDFLAG=2
7307.     RETURN
7308.     200 IF(CURSR2.GT.SHKTOL) GO TO 300
7309.     CALL TEQVAL
7310.     SHKCHK=SHKTOL-DSHKVL
7311.     IF(SHKCHK.GT.0.20) GO TO 300
7312.     IDFLAG=3
7313.     RETURN
7314.     300 IF(PENVAL.GT.PENMAX) GO TO 400
7315.     IDFLAG=4
7316.     RETURN
7317.     400 IDFLAG=5
7318.     GO TO 670
7319.     500 DFPDFF=DFUNC2-DFUNC1
7320.     IF(DFPDFF.LT.0.0) DFPDFF=DFPDFF*(-1.0)
7321.     IF(DFPDFF.LT.DFPSTP) GO TO 200
7322.     CALL DFPASG
7323.     IF(PCONT(29).EQ.1) CALL QDRST
7324.     GO TO 200
7325.     670 RETURN
7326.     END
7327.     SUBROUTINE DFPASG
7328.     COMMON /HID/HID(10)
7329.     COMMON /POS/POS(10)
7330.     COMMON /HEIGHT/HEIGHT(10)
7331.     COMMON /NOZERO/NOZERO
7332.     COMMON /DSAVW/DASAVW(10)
7333.     COMMON /DSAVHHT/DSAVHHT(10)
7334.     COMMON /DSAVPS/DSAVPS(10)
7335.     COMMON /CURSR2/CURSR2
7336.     COMMON /USAVSK/USAVSK
7337.     COMMON /DFUNC2/DFUNC2
7338.     COMMON /DSAVFC/DSAVFC
7339.     DO 10 I=1,NOZERO
**SUBROUTINE OUDRST**

**COMMON /CURSR2/CURSR2**

**COMMON /DFUNC2/DFUNC2**

**COMMON /PENVAL/PENVAL**

**WRITE(6,1)**

1 FORMAT('DUE TO A POOR DIRECTION VECTOR, NO IMPROVEMENT IN THE OBJECTIVE FUNCTION COULD OCCUR')

**WRITE(6,2)**

2 FORMAT('BECAUSE OF THIS, THE FOLLOWING WILL TAKE PLACE;')

**WRITE(6,3)**

3 FORMAT(')

**WRITE(6,4)**

4 FORMAT(' PENALTY FUNCTION WILL BE UPDATED IF ITS LIMIT HAS NOT BEEN EXCEEDED')

**WRITE(6,5)**

5 FORMAT(' CURRENT PENALTY VALUE = ',E10.3)

**WRITE(6,6)**

6 FORMAT(' OBJECTIVE FUNCTION VALUE WILL BE RE-SET TO THE PREVIOUS VALUE')

**WRITE(6,7)**

7 FORMAT(' PREVIOUS AND NOW CURRENT PENALTY FUNCTION VALUE X= ',E15.7)

**WRITE(6,8)**

8 FORMAT(' PREVIOUS AND NOW CURRENT OBJECTIVE FUNCTION VALUE X= ',E15.7)

**WRITE(6,9)**

9 FORMAT(' PREVIOUS AND NOW CURRENT SHRINKAGE FUNCTION VALUE X= ',E15.7)

**WRITE(6,10)**

10 FORMAT(' THE VALUE OF THE INDIVIDUAL VARIABLES WILL BE RE-SET TO THE PREVIOUS VALUES')

**CALL OUTVAR**

**RETURN**

**END**

**SUBROUTINE OERROR**

**WRITE(6,1)**

1 FORMAT('OBJECTIVE FUNCTION MUST NOT BE CONVEX')

**RETURN**

**END**

**SUBROUTINE OUDERR**

**WRITE(6,1)**

1 FORMAT('OBJECTIVE FUNCTION MUST NOT BE CONVEX')

**RETURN**

**END**

**SUBROUTINE OFPRST**

**COMMON /CURSR1/CURSR1**

**COMMON /CURSR2/CURSR2**

**COMMON /DFUNC1/DFUNC1**

**COMMON /DFUNC2/DFUNC2**
SUBROUTINE DFDIRR
    COMMON /DIRECT/DIRECT(20)
    COMMON /NOZERO/NOZERO
    COMMON /ZVAL/ZVAL(10)
    INTEGER * 2 ZVAL
    COMMON /D/D(20)
    COMMON /NUMSOL/NUMSOL
    NUMSOL=0
    DO 10 I=1,NOZERO
    IF(ZVAL(I).EQ.1) NUMSOL=NUMSOL+1
    10 CONTINUE
    II=0
    I2=NUMSOL
    I3=NUMSOL+NUMSOL
    DO 20 L=1,NOZERO
    IF(ZVAL(L).EQ.0) GO TO 20
    II=II+1
    I2=I2+1
    I3=I3+1
    L1=L
    L2=L+NOZERO
    L3=L+NOZERO+NOZERO
    D(1)=DIRECT(L1)
    D(2)=DIRECT(L2)
    D(3)=DIRECT(L3)
    20 CONTINUE
    RETURN
    END
SUBROUTINE UPSET
    COMMON /P/P(20)
    COMMON /D/D(20)
    COMMON /DPT/GPT
    COMMON /NUMSOL/NUMSOL
    IT=NUMSOL+3
    DO 10 I=1,IT
    P(I)=D(I)*GPT
    10 CONTINUE
    RETURN
    END
SUBROUTINE DGRDSV
    COMMON /DPPGDH/DPPGDH(10)
    COMMON /DPPGDH/DPPGDH(10)
    COMMON /DPPGDP/DPPGDP(10)
    COMMON /NOZERO/NOZERO
    COMMON /GDHSAV/GDHS(10)
    COMMON /GDHSAV/GDHS(10)
 COMMON /GDPSAV/GDPSAVUO
    DO 10 I=1,NOZERO
    GDWSAV(I)=DFPGDW(I)
    GDHSAV(I)=DFPGDH(I)
    GDPSAV(I)=DFPGDP(I)
 10 CONTINUE
RETURN
END

SUBROUTINE DQSET
COMMON /Q/Q(20)
COMMON /DFPGDd/DFPGDNI(I)
COMMON /DFPGDH/DFPGDH1(I)
COMMON /DFPGDP/DFPGDP(10)
COMMON /GDWSAV/GDWSAVI(I)
COMMON /GDHSAV/GDHSAVI(I)
COMMON /GDPSAV/GDPSAVI(I)
COMMON /ZVAL/ZVALI(I)
INTEGER * 2 ZVAL
COMMON /NUMSOL/NUMSOL
COMMON /NOZERO/NOZERO
DO 1 IR=1,20
   Q(IR)=0.0
1 CONTINUE
11=0
I2=NUMSOL
I3=NUMSOL+NUMSOL
DO 10 I=1,NOZERO
   IF(ZVAL(I).EQ.0) GO TO 10
   I1=I1+1
   I2=I2+1
   I3=I3+1
   Q(I1)=DFPGDP(I1)-GDPSAV(I1)
   Q(I2)=DFPGDH(I2)-GDHS4V(I2)
   Q(I3)=DFPGDW(I3)-GDWSAV(I3)
 10 CONTINUE
RETURN
END

SUBROUTINE DTERM2
COMMON /P/P(20)
COMMON /NOZERO/NOZERO
COMMON /Q/Q(20)
COMMON /STF2M2/STF2M2(I1,J)
COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT
COMMON /NUMSOL/NUMSOL
I1=NUMSOL*3
DO 10 I=1,11
   DO 20 J=1,I1
      STF2M2(I,J)=P(I)*P(J)
   20 CONTINUE
 10 CONTINUE
DENOM=0.0
DO 30 I=1,11
   DENOM=DENOM+P(I)*Q(I)
30 CONTINUE
DO 40 I=1,11
   DO 50 J=1,I1
      STF2M2(I,J)=STF2M2(I,J)/DENOM
   50 CONTINUE
40 CONTINUE
IF(PCONT(20).EQ.1) WRITE(6,100)
100 FORMAT( * , 'TERM2 MATRIX IS SET UP' )
DO 110 I=1,11
110 CONTINUE
IF(PCONT(20).EQ.1) WRITE(6,120) (STERM2(I,J),J=1,11)
120 FORMAT( ',8(I1*X,F12.6)')
RETURN
END

SUBROUTINE DTERM3
COMMON /S/S(20,20)
COMMON /NOZERO/NOZERO
COMMON /Q/Q(20,20)
COMMON /STERM2/STERM2(20,20)
COMMON /STERM3/STERM3(20,20)
DIMENSION VAL(20),VI(20,20)
COMMON /PCONT/PCONT(50)
INTEGER*2 PCONT
COMMON /NUMSOL/NUMSOL
IF(PCONT(21).EQ.1) WRITE(6,150)
150 FORMAT( * , 'TERM3 MATRIX SET UP' )
I=NUMSOL+3
DO 10 I=1,11
VAL(I)=0.0
10 CONTINUE
DO 11 I=1,11
11 VAL(I)=VAL(I)+(S(I,J)*Q(J))
DO 20 J=1,11
20 CONTINUE
11 CONTINUE
IF(PCONT(21).EQ.1) WRITE(6,200) (VALID,1=1,11)
200 FORMAT( ',8(I1*X,F12.6)')
DO 21 I=1,11
21 VI(I,J)=VAL(I)*Q(J)
21 CONTINUE
11 CONTINUE
IF(PCONT(21).EQ.1) WRITE(6,210)
210 FORMAT( * , 'VI MATRIX' )
DO 220 I=1,11
220 VI(I,J)=0.0
220 CONTINUE
DO 230 J=1,11
230 CONTINUE
210 CONTINUE
DO 240 J=1,11
240 CONTINUE
230 CONTINUE
DO 250 K=1,11
250 STERM3(I,J)=STERM3(I,J)+VI(I,K)*S(K,J)
250 CONTINUE
DO 260 K=1,11
260 CONTINUE
250 CONTINUE
IF(PCONT(21).EQ.1) WRITE(6,300)
300 FORMAT( * , 'STERM3 INITIAL SET UP' )
DO 310 I=1,11
310 IF(PCONT(21).EQ.1) WRITE(6,320) (STERM3(I,J),J=1,11)
403 FORMAT(' ',8(I10,F12.6))

310 CONTINUE
311 DO 90 I=1,11
312 VAL(I)=0.0
313 90 CONTINUE
314 DO 100 I=1,11
315 VAL(I)=VAL(I)+Q(I)*S(I,I)
316 100 CONTINUE
317 IF(CONT(21).EQ.1) WRITE(6,400)
318 400 FORMAT(' ',VAL VECTOR SET UP ')
319 DO 410 I=1,11
320 IF(CONT(21).EQ.1) WRITE(6,420) I,VAL(I)
321 420 FORMAT(' ',VAL = ',I4,1X,F12.6)
322 410 CONTINUE
323 DENEOM=0.0
324 DO 120 I=1,11
325 DENEOM=DENEOM-VAL(I)*Q(I)
326 120 CONTINUE
327 IF(CONT(21).EQ.1) WRITE(6,500) DENEOM
328 500 FORMAT(' ',DENOMINATOR = ',F12.6)
329 DO 130 I=1,11
330 DO 140 J=1,11
331 STERM3(I,J)=STERM3(I,J)/DENEOM
332 140 CONTINUE
333 130 CONTINUE
334 IF(CONT(21).EQ.1) WRITE(6,600)
335 600 FORMAT(' ',FINAL SET UP OF STERM3')
336 DO 610 I=1,11
337 IF(CONT(21).EQ.1) WRITE(6,620) S(I,J),J=1,11
338 620 FORMAT(8(1X,F12.6))
339 610 CONTINUE
340 RETURN
341 END
342 SUBROUTINE DSETS
343 COMMON /S/S(120,20)
344 COMMON /STERM2/STERM2(120,20)
345 COMMON /STERM3/STERM3(120,20)
346 COMMON /NOZERO/NOZERO
347 COMMON /PCONT/PCONT(50)
348 INTEGER*2 PCONT
349 COMMON /NUMSOL/NUMSOL
350 IL=NUMSOL*3
351 DO 10 I=1,11
352 DO 20 J=1,11
353 S(I,J)=S(I,J)+STERM2(I,J)-STERM3(I,J)
354 20 CONTINUE
355 10 CONTINUE
356 IF(CONT(19).EQ.1) WRITE(6,100)
357 100 FORMAT(' ',S MATRIX IS SET UP ')
358 DO 110 I=1,11
359 IF(CONT(19).EQ.1) WRITE(6,120) S(I,J),J=1,11
360 120 FORMAT(8(1X,F12.6))
361 110 CONTINUE
362 RETURN
363 END
364 SUBROUTINE DOIRCT
365 COMMON /U/U(120)
366 COMMON /OU/DD120)
7640. COMMON /S/S(20,20)
7641. COMMON /DFPGOM/DFPGOM(10)
7642. COMMON /DFPGOM/DFPGOM(10)
7643. COMMON /DFPGOM/DFPGOM(10)
7644. COMMON /NUMSOL/NUMSOL
7645. COMMON /ZVAL/ZVAL(10)
7646. INTEGER * 2 ZVAL
7647. COMMON /NOZERO/NOZERO
7648. COMMON /DIRECT/DIRECT(20)
7649. DO 1 I = 1, 20
7650. DIRECT(I) = 0.0
7651. 1 CONTINUE
7652. 1 = 0
7653. 2 = NUMSOL
7654. I3 = NUMSOL + NUMSOL
7655. DD(I) = 0.0
7656. 11 = 11 + 1
7657. 12 = 12 + 1
7658. 13 = 13 + 1
7659. IF(ZVAL(I).EQ.0) GO TO 10
7660. I1 = I1 + 1
7661. 12 = 12 + 1
7662. 13 = 13 + 1
7663. DD(I1) = DFPGOM(I1)
7664. 11 = 11 + 1
7665. 12 = 12 + 1
7666. 13 = 13 + 1
7667. DO 10 I = 1, NOZERO
7668. 10 CONTINUE
7669. ICT = Q
7670. I1 = 0
7671. I2 = NOZERO
7672. I3 = NOZERO + NOZERO
7673. ICT = 0
7674. DO 100 I = 1, NOZERO
7675. I1 = I1 + 1
7676. 12 = 12 + 1
7677. 13 = 13 + 1
7678. IF(ZVAL(I).EQ.0) GO TO 110
7679. ICT = ICT + 1
7680. I1 = ICT
7681. I2 = ICT + NUMSOL
7682. I3 = ICT + NUMSOL + NUMSOL
7683. DIRECT(I1) = 0(I1)
7684. DIRECT(I2) = 0(I2)
7685. DIRECT(I3) = 0(I3)
7686. GO TO 100
7687. DIRECT(I1) = 0.0
7688. DIRECT(I2) = 0.0
7689. DIRECT(I3) = 0.0
7690. 100 CONTINUE
7691. DIRECT(I1) = DIRECT(I1) * (-1.0)
7692. DIRECT(I2) = DIRECT(I2) * (-1.0)
7693. DIRECT(I3) = DIRECT(I3) * (-1.0)
7694. RETURN
7695. END
7696. SUBROUTINE CUONIT
7697. COMMON /P/P(120)
COMMON /Q/Q(20)
COMMON /D/D(20)
COMMON /S/S(20,20)
COMMON /DIRECT/DIRECT(20)
COMMON /STERM2/STERM2(20,20)
COMMON /STERM3/STERM3(20,20)
COMMON /NOZERO/NOZERO
COMMON /NUMSOL/NUMSOL
WRITE(6,1)
1 FORMAT('1**33X,'UPDATE OF DIRECTION VECTOR INFORMATION - DAVIDCN-
XFLETCHER-Powell*)
WRITE(6,2)
2 FORMAT('**8X,'P*,8X,8X,'Q*,8X,8X,'D*,8X,8X,'DD*,7X)
WRITE(6,3)
3 FORMAT('11=NUMSOL*3
DO 10 I=1,11
WRITE(6,11)I,P(I),Q(I),D(I),DD(I)
10 CONTINUE
WRITE(6,20)
20 FORMAT('*-STERM2 - MATRIX')
WRITE(6,3)
DO 30 I=1,11
WRITE(6,31)STERM2(I,J),J=1,10
30 CONTINUE
WRITE(6,50)
50 FORMAT('11=NUMSOL*3
WRITE(6,51)I, D(I)
51 CONTINUE
WRITE(6,60)
60 FORMAT('11=NOZERO*3
WRITE(6,61)I,D(I)
61 CONTINUE
RETURN
SUBROUTINE DFPOPT
RETURN
END

SUBROUTINE QUDOPT
COMMON /WID/WID(10)
COMMON /HEIGHT/HEIGHT(10)
COMMON /POS/POS(10)
COMMON /DFUNC2/DFUNC2
COMMON /NOZERO/NOZERO
COMMON /ZVAL/ZVAL(20)
INTEGER * 2 ZVAL
COMMON /PENVAL/PENVAL
COMMON /CURSR2/CURSR2
WRITE(6,10)
10 FORMAT(*1',57X,'OPTIMAL SOLUTION')
WRITE(6,11) PENVAL,CURSR2,DFUNC2
11 FORMAT('FINAL PENALTY VALUE = ',E10.3,'FINAL SHRINKAGE VALUE = ',E10.7)
DO 20 I=1,NOZERO
WRITE(6,21) (WID(I))
21 FORMAT(13,13,WIDTH = ',F15.8)
WRITE(6,22) I
22 FORMAT(13,13,'HEIGHT = ',F15.8)
WRITE(6,23) PCS(I)
23 FORMAT(13,13,'POSITION = ',F15.8)
20 CONTINUE
RETURN
END

SUBROUTINE DFPPIN
COMMON /PCONT/PCONT(50)
INTEGER * 2 PCONT
COMMON /PENVAL/PENVAL
COMMON /PEMFCT/PEMFCT
COMMON /IDFLAG/IDFLAG
COMMON /CURSR2/CURSR2
COMMON /CURSR1/CURSR1
COMMON /DFUNC1/DFUNC1
COMMON /func/func
COMMON /PENMLT/PENMLT
7801. COMMON /SHRINK/SHRINK
7802. PENVAL=PENVAL*PENMLT
7803. IF(PENVAL.GT.PENMAX) IDFLAG=5
7804. IF(PENVAL.GT.PENMAX) RETURN
7805. CALL SINTAL
7806. CALL OBJVAL
7807. SHRINK=CURSR2
7808. CALL DFPFNT
7809. DFUNC1=FUNCNTN
7810. CURSR1=CURSR2
7811. CALL DFPSGD
7812. CALL DFPFGD
7813. IF(PCONT12).EQ.1) CALL OUTOGD
7814. RETURN
7815. ENO
7816. SUBROUTINE OUDPIN
7817. COMMON /PENVAL/PENVAL
7818. WRITE(6,1)
7819. 1 FORMAT(*,*,*)
7820. WRITE(6,2) PENVAL
7821. 2 FORMAT(*,'CURRENT PENALTY VALUE = ',10.2)
7822. RETURN
7823. ENO
7824. SUBROUTINE DINFEA
7825. WRITE(6,1)
7826. 1 FORMAT(*,'SOLUTION IS NOT FEASIBLE')
7827. RETURN
7828. ENO
7829. SUBROUTINE QUOINF
7830. WRITE(6,1)
7831. 1 FORMAT(*,'SOLUTION IS NOT FEASIBLE')
7832. RETURN
7833. ENO
7834. SUBROUTINE RONBEN
7835. COMMON /COMUSE/COMUSE(100)
7836. INTEGER * 2 COMUSE
7837. COMMON /COMB/COMB(100,6)
7838. INTEGER * 2 COMB
7839. COMMON /NOZERO/NOZERO
7840. COMMON /ZVAL/ZVAL(100)
7841. INTEGER * 2 ZVAL
7842. COMMON /NUMCMB/NUMCMB
7843. COMMON /SHRINK/SHRINK
7844. COMMON /SHKTOL/SHKTOL
7845. DD 100 =1,NUMCMB
7846. IF(COMUSE(I).EQ.0) GO TO 100
7847. DC 101 J=1,NOZERO
7848. ZVAL(J)=COMB(I,J)
7849. 101 CONTINUE
7850. CALL MAXVAL
7851. CALL TEQVAL
7852. IF(SHRINK.LT.SHKTOL) GO TO 100
7853. COMUSE(I)=0
7854. 100 CONTINUE
7855. RETURN
7856. ENO
7857. //GO.SYSIN DO *
7858. 6
7859. 5 6
<p>|   7860. |   5 4 |
|   7861. |   4 2 |
|   7862. |   4 2 |
|   7863. |   4 4 |
|   7864. |   4 4 |
|   7865. |   6   |
|   7866. |   3 4 1 1 4 |
|   7867. |  1812 |
|   7868. |  1316 |
|   7869. |  2112 |
|   7870. |  2116 |
|   7871. |   4 4 4 2 1 |
|   7872. |   412  |
|   7873. |   416  |
|   7874. |   712  |
|   7875. |   716  |
|   7876. |   4 4 4 3 1 |
|   7877. |   4 8   |
|   7878. |   411  |
|   7879. |   7 8   |
|   7880. |   711  |
|   7881. |   4 4 4 5 1 |
|   7882. |   4 4   |
|   7883. |   4 7   |
|   7884. |   7 4   |
|   7885. |   7 7   |
|   7886. |   3 3 1 6 4 |
|   7887. |  16 4   |
|   7888. |  16 7   |
|   7889. |  18 4   |
|   7890. |  18 7   |
|   7891. |   3 3 1 4 4 |
|   7892. |   12 8  |
|   7893. |  1211  |
|   7894. |   14 8  |
|   7895. |   1411  |
|   7896. |   1924  |
|   7897. | 00000000000000000000 |
|   7898. | 0111111111111111111111 |
|   7899. | 0111111111111111111111 |
|   7900. | 01144444444444444411 |
|   7901. | 01144444444444444411 |
|   7902. | 01144444444444444411 |
|   7903. | 01144444444444444411 |
|   7904. | 01122222222222222211 |
|   7905. | 01122222222222222211 |
|   7906. | 01122222222222222211 |
|   7907. | 01122222222222222211 |
|   7908. | 01122222222222222211 |
|   7909. | 01122222222222222211 |
|   7910. | 01122222222222222211 |
|   7911. | 01122222222222222211 |
|   7912. | 01133331111222222211 |
|   7913. | 01133331111222222211 |
|   7914. | 01133331111333333110 |
|   7915. | 0111111111133333110 |
|   7916. | 0111111111133333110 |
|   7917. | 0111111111133333110 |
|   7918. | 01111111111111111111 |
|   7919. | 01111111111111111111 |</p>
<table>
<thead>
<tr>
<th></th>
<th>CHILL</th>
<th>RISER</th>
<th>RISER</th>
<th>RISER</th>
<th>CHILL</th>
<th>RISER</th>
<th>CHILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4000.0</td>
<td>3000.0</td>
<td>2000.0</td>
<td>50.0</td>
<td>4000.0</td>
<td>3000.0</td>
<td>2000.0</td>
</tr>
<tr>
<td>2</td>
<td>1500.0</td>
<td>2500.0</td>
<td>1200.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>500.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>3500.0</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
</tr>
<tr>
<td>5</td>
<td>1600.0</td>
<td>1500.0</td>
<td>2500.0</td>
<td>1200.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>6</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>7</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>8</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>9</td>
<td>1300.0</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>11</td>
<td>2000.0</td>
<td>5200.0</td>
<td>500.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>12</td>
<td>3500.0</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
</tr>
<tr>
<td>13</td>
<td>1600.0</td>
<td>1500.0</td>
<td>2500.0</td>
<td>1200.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>14</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>15</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>16</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>17</td>
<td>1300.0</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>18</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>19</td>
<td>2000.0</td>
<td>5200.0</td>
<td>500.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>3500.0</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
</tr>
<tr>
<td>21</td>
<td>1600.0</td>
<td>1500.0</td>
<td>2500.0</td>
<td>1200.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>22</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>23</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>24</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>25</td>
<td>1300.0</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>26</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>27</td>
<td>2000.0</td>
<td>5200.0</td>
<td>500.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>28</td>
<td>3500.0</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
</tr>
<tr>
<td>29</td>
<td>1600.0</td>
<td>1500.0</td>
<td>2500.0</td>
<td>1200.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>30</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>31</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>32</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>33</td>
<td>1300.0</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>34</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>35</td>
<td>2000.0</td>
<td>5200.0</td>
<td>500.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>36</td>
<td>3500.0</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
</tr>
<tr>
<td>37</td>
<td>1600.0</td>
<td>1500.0</td>
<td>2500.0</td>
<td>1200.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>38</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>39</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>40</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>41</td>
<td>1300.0</td>
<td>100.0</td>
<td>120.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>42</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>43</td>
<td>2000.0</td>
<td>5200.0</td>
<td>500.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>44</td>
<td>3500.0</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
</tr>
<tr>
<td>45</td>
<td>1600.0</td>
<td>1500.0</td>
<td>2500.0</td>
<td>1200.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>46</td>
<td>550.0</td>
<td>320.0</td>
<td>300.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>47</td>
<td>150.0</td>
<td>250.0</td>
<td>60.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>