NEURAL DYNAMICS MODELS FOR SCHEDULING/COST
OPTIMIZATION OF CONSTRUCTION PROJECTS AND OPTIMUM
DESIGN OF COLD-FORMED STEEL BEAMS

A Thesis

Presented in Partial Fulfillment of the Requirement for
the Degree Master of Science in the Graduate
School of The Ohio State University

By

Asim Salimul Karim, B.Sc.

*****

The Ohio State University
1996

Master's Examination Committee:
Professor Hojjat Adeli, Adviser
Professor S. K. Chaturvedi
Professor K. V. Balasubramanyam

Approved by

Adviser

Graduate Program in Civil Engineering
ABSTRACT

A general mathematical formulation is presented for scheduling of construction projects and applied to the problem of highway construction scheduling. Repetitive and non-repetitive tasks, work continuity considerations, multiple-crew strategies, and the effects of varying job conditions on the performance of a crew can be modeled. An optimization formulation is presented for the construction project scheduling problem with the goal of minimizing the direct construction cost. The nonlinear optimization problem is then solved by the neural dynamics model developed recently by Adeli and Park. For any given construction duration, the model yields the optimum construction schedule for the minimum construction cost automatically. By varying the construction duration, one can solve the cost-duration trade-off problem and obtain the global optimum schedule and the corresponding minimum construction cost. The new construction scheduling model provides the capabilities of both the CPM and the LSM approaches. In addition, it provides features desirable for repetitive projects such as highway construction and allows schedulers greater flexibility. It is particularly suitable for studying the effects of change order on the construction cost. The research provides the mathematical foundation for the development of a new generation of more general, flexible, and accurate construction scheduling systems.
An important advantage of cold-formed steel is the greater flexibility of cross-sectional shapes and sizes available to the structural steel designer. The lack of standard optimized shapes, however, makes the selection of the most economical shape very difficult if not impossible. This task is further complicated by the complex and highly nonlinear nature of the rules that govern their design. A general mathematical formulation and computational model is presented for optimization of cold-formed steel beams. The nonlinear optimization problem is solved by adapting the robust neural dynamics model developed recently by Adeli and Park. The basis of the design can be AISI ASD or LRFD Specifications. The computational model has been applied to three different commonly-used types of cross-sectional shapes: hat, I, and Z shapes. The robustness and generality of the approach have been demonstrated by application to three different examples. This research lays the mathematical foundation for automated optimum design of structures made of cold-formed shapes. The result would be more economical use of cold-formed steel.
To My Parents
ACKNOWLEDGMENTS

I would like to thank my adviser, Dr. Hojjat Adeli, for inspiring, guiding, and supporting me throughout this work. His constant encouragement was a great help in the timely completion of the work. Thanks also goes to Dr. S. K. Chaturvedi and Dr. K. V. Balasubramanyam for serving on my examination committee.

I am grateful to my colleague Murphy Hsu for all the help he has been.

I appreciate the support provided by my parents and my brother Saqib during the course of my studies.

Finally, the financial support from the National Science Foundation is gratefully acknowledged.
VITA

March 2, 1971 .............................................. Born - Pakistan

July 1994 .................................................. B.Sc. (Honors), University of Engineering
and Technology, Lahore, Pakistan

October 1994-December 1994 ......................... Junior Engineer, Amir, Rizwan, and Wasim
(Pvt.) Ltd., Lahore, Pakistan

June 1996-present ....................................... Graduate research associate, The Ohio State
University

FIELDS OF STUDY

Major field: Civil engineering

Minor fields: Structural engineering

Artificial neural networks

Construction scheduling
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>iv</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>v</td>
</tr>
<tr>
<td>VITA</td>
<td>vi</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>x</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xi</td>
</tr>
<tr>
<td>CHAPTERS:</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION AND OBJECTIVES</td>
<td>1</td>
</tr>
<tr>
<td>2. SCHEDULING/COST OPTIMIZATION AND NEURAL DYNAMICS MODEL FOR CONSTRUCTION PROJECTS</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Cost-duration relationship of a project</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Formulation of the scheduling optimization problem</td>
<td>9</td>
</tr>
<tr>
<td>2.3.1 Break down the work into tasks, crews, and segments</td>
<td>10</td>
</tr>
</tbody>
</table>
2.3.2 Specify the internal logic of repetitive tasks ........................................ 11
2.3.3 Specify the external logic of repetitive and non-repetitive tasks ........ 12
2.4 Artificial neural networks and scheduling ............................................. 15
2.5 Neural dynamics cost optimization model for construction projects ......... 17
  2.5.1 Formulation ...................................................................................... 17
  2.5.2 Topological characteristics ............................................................... 20
2.6 Illustrative example ................................................................................. 24
  2.6.1 Cost-duration relationship ................................................................. 24
  2.6.2 Scheduling logic .............................................................................. 28
  2.6.3 Solution of the problem ................................................................... 34
2.7 Conclusion .............................................................................................. 34

3. NEURAL NETWORK MODEL FOR OPTIMIZATION OF COLD-FORMED STEEL BEAMS ................................................................. 39
  3.1 Introduction .......................................................................................... 39
  3.2 Minimum weight design of cold-formed steel beams ......................... 40
  3.3 Neural dynamics optimization model .................................................... 51
  3.4 Neural network model for optimization of cold-formed steel beams ...... 54
  3.5 Applications ......................................................................................... 59
    3.5.1 Example 1 ..................................................................................... 60
    3.5.2 Example 2 ..................................................................................... 63
    3.5.3 Example 3 ..................................................................................... 66
  3.6 Concluding remarks .............................................................................. 69
LIST OF REFERENCES ................................................................. 70

APPENDICES:
A. COMPUTER CODE LISTING FOR CHAPTER 2............................. 74
B. COMPUTER CODE LISTING FOR CHAPTER 3.............................. 101
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>The description and type of tasks in the illustrative example</td>
</tr>
<tr>
<td>2.</td>
<td>The direct cost-duration relationship for each task</td>
</tr>
<tr>
<td>3.</td>
<td>Task details for the illustrative example</td>
</tr>
<tr>
<td>4.</td>
<td>The internal logic of repetitive tasks</td>
</tr>
<tr>
<td>5.</td>
<td>The external logic of tasks</td>
</tr>
<tr>
<td>6.</td>
<td>The direct, indirect, and total costs variation for the illustrative example</td>
</tr>
<tr>
<td>7.</td>
<td>Optimum solutions for example 1</td>
</tr>
<tr>
<td>8.</td>
<td>Optimum solutions for example 2</td>
</tr>
<tr>
<td>9.</td>
<td>Optimum solutions for example 3</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A linear planning chart</td>
<td>4</td>
</tr>
<tr>
<td>2. Typical variation of direct, indirect and total costs of a construction project</td>
<td>7</td>
</tr>
<tr>
<td>3. A linear direct cost-duration curve for unit quantity of task ( i )</td>
<td>8</td>
</tr>
<tr>
<td>4. A nonlinear direct cost-duration curve for unit quantity of task ( i )</td>
<td>8</td>
</tr>
<tr>
<td>5. The neural network topology for the neural dynamics cost optimization model</td>
<td>21</td>
</tr>
<tr>
<td>6. The linear direct cost-duration curve for unit quantity of task ( 2 )</td>
<td>26</td>
</tr>
<tr>
<td>7. The nonlinear direct cost-duration curve for unit quantity of task ( 10 )</td>
<td>27</td>
</tr>
<tr>
<td>8. The areas and types of vegetation that have to be cleared by tasks ( 1 ) and ( 9 )</td>
<td>30</td>
</tr>
<tr>
<td>9. Activity-on-node diagram for the first five tasks of the illustrative example</td>
<td>33</td>
</tr>
<tr>
<td>10. The direct cost convergence curves for the illustrative example</td>
<td>36</td>
</tr>
<tr>
<td>11. Time-cost trade-off curve for the illustrative example</td>
<td>37</td>
</tr>
<tr>
<td>12. The linear planning chart for the minimum total cost schedule</td>
<td>38</td>
</tr>
<tr>
<td>13. The variables involved in the design of hat, I, and Z-shaped cold-formed steel beams</td>
<td>41</td>
</tr>
</tbody>
</table>
14. Effective design widths of cold-formed steel shapes in bending.
   (a) Shapes with a stiffened compression flange, and (b) shapes with an
   unstiffened compression flange .................................................. 44

15. The components of the neural network model for optimization of cold-formed
    steel beams ............................................................................. 55

16. Convergence histories for example 1 ........................................... 62

17. Example 2 .............................................................................. 63

18. Convergence histories for example 2 ........................................... 65

19. Example 3 .............................................................................. 66

20. Convergence histories for example 3 ........................................... 68
CHAPTER 1

INTRODUCTION AND OBJECTIVES

Artificial neural networks (ANNs) are a functional abstraction of the human brain. They mimic the structure of the central nervous system in having a large number of interconnected processing elements. By specifying an appropriate topology and learning rule, the evolution of the ANN is controlled to yield the desired solution. ANNs are especially good at solving problems of estimation and prediction, pattern recognition, and optimization (Adeli and Yeh, 1989; Adeli and Zhang, 1993; Adeli and Hung, 1995, Adeli and Park, 1995a, b, c; Adeli and Park, 1996a).

Recently, using the concept of neural dynamics Adeli and Park (1995a) developed a robust ANN model for complicated nonlinear programming problems and applied it to structural optimization. In this work, we adapt their neural dynamics model to problems of construction cost optimization and optimum design of cold-formed steel beams. To minimize the cost of a construction project it is essential that the optimization problem be based on an accurate schedule. Techniques such as CPM and LSM are not general and flexible enough for this purpose. Moreover, their primary emphasis is on construction time and not cost. In chapter 2, we present a general scheduling/cost optimization model for
construction projects using the neural dynamics concept and apply it to a highway construction project.

Cold-formed steel beams are economical for carrying relatively small loads. However, their design is complicated by the lack of standard optimized shapes and the nonlinear nature of the design rules. In chapter 3, we present a neural network model for the minimum weight design of cold-formed beams according to the American Iron and Steel Institute's ASD and LRFD Specifications (AISI, 1989 and 1991).
CHAPTER 2

SCHEDULING/COST OPTIMIZATION AND NEURAL DYNAMICS

MODEL FOR CONSTRUCTION PROJECTS

2.1 INTRODUCTION

Most construction projects involve a combination of repetitive and non-repetitive tasks. A typical example is highway construction in which tasks such as clearing and grubbing are performed repeatedly over the length of the highway, and tasks such as site office construction are carried out only once. Presently, traditional network scheduling methods such as CPM and PERT are used for the scheduling and monitoring of such projects. Despite their extensive use these methods have a number of shortcomings:

- Network methods do not guarantee continuity of work in time which may result in crews being idle.
- Multiple-crew strategies are difficult to implement in the network methods.
- The network diagram is not suitable for monitoring the progress of a project.
- Network methods do not provide an efficient structure for the representation of repetitive tasks. All tasks are represented similarly and there is no consideration of the location of work in the scheduling.
Figure 1. A linear planning chart

To overcome these shortcomings new approaches have been proposed in the literature particularly for repetitive projects. Figure 1 presents a linear planning chart which is a graph of location (distance) versus time for the work to be carried out. Such a planning chart represents the progress of a task and can be used to monitor a project. The linear planning chart (also called LSM diagram) motivated the development of linear scheduling method (LSM). Selinger (1980) presented equations for the lines in a linear planning chart assuming non-interference of crews and continuity of work. Johnston (1981) showed the LSM is flexible and can be used to model most situations encountered in highway construction projects. Using optimal control theory, Handa and Barcia (1986) formulated
the problem as an optimization one minimizing the project duration. These early LSM models had limitations such as constant rate of production for each task, binding continuity constraints, and no provisions for the use of multiple crews.

Russell and Caselton (1988) presented a dynamic programming formulation to minimize the project duration. Their formulation can accommodate variable production rates for each task and non-binding work continuity constraints. Russell and Wong (1993) described and showed the use of a general scheduling model developed by incorporating the capabilities of CPM and LSM. In their model, each task is defined by a set of attributes which are then linked together using general precedence conditions to form a schedule.

Highway construction projects are large projects in terms of capital requirement. Minimizing cost is therefore a primary goal in the planning and scheduling of such projects. Cost, however, is closely related to time. In general, direct project cost increases with a decrease in project duration and this trade-off problem is complicated by the number of variables involved. A computer model to automate the process of project direct cost minimization is therefore highly desirable.

In the recent literature, direct cost optimization systems have been presented for LSM and CPM. Reda's (1990) LSM model assumes constant production rate for each task and binding constraints on work continuity. The cost-duration relationship for each task is assumed to be linear. Liang et al. (1995) present a hybrid linear/integer programming approach for handling a combination of discrete and linearly continuous cost-duration relationship for tasks.
In this chapter, a general mathematical formulation is presented for scheduling of construction projects. Various scheduling constraints are expressed mathematically. The construction scheduling is posed as an optimization problem where project direct cost is minimized for a given project duration assuming any combination of linear and nonlinear task cost-duration relationships. The robust neural dynamics model developed recently by Adeli and Park (1995a) is adapted for optimization.

2.2 COST-DURATION RELATIONSHIP OF A PROJECT

The major cost of a project consists of direct and indirect costs. The resources allocated to each task of a project determine the direct cost. Indirect costs are overhead costs. Additional costs may be incurred by the contractor in the form of damages if the project is not completed on time. The duration of a project is obtained by sequencing individual tasks whose durations are estimated from a knowledge of the resources allocated to each task and the job conditions. Thus, cost and duration are intricately related. Both of these parameters are of great importance to the contractor who strives to minimize cost while at the same time satisfying the contractual requirements, the most important of which is the completion deadline. Figure 2 shows the typical variation of direct, indirect and total costs of a construction project with the project duration. Assuming the sequencing constraints are not changed, the direct cost, in general, has an inverse relationship with the duration of a construction project. The indirect cost increases with an increase in the duration of the project. The total cost is the sum of these two and can increase or decrease with duration. By solving the direct cost optimization problem for
Figure 2. Typical variation of direct, indirect and total costs of a construction project

various durations the global optimum solution can be obtained from Figure 2.

There is also an inverse relationship between direct cost and the duration of an individual task. A scheduler estimates the time required to complete a task from the resources allocated to it. This time is based on assumed labor and equipment productivity rates ignoring the effects of varying job conditions. Depending on the options and the availability of resources the scheduler has for each task, a cost-duration curve can be constructed. This curve can be continuous or discrete. For efficient mathematical
Figure 3. A linear direct-cost duration curve for unit quantity of task $i$

Figure 4. A nonlinear direct-cost duration curve for unit quantity of task $i$
formulation the discrete relationship is approximated by a continuous linear (Figure 3) or nonlinear (Figure 4) curve. In this way a continuous variable optimization technique can be used to solve the construction time-cost trade-off problem.

In highway construction, it is convenient to represent cost and duration of a task in unit quantities of work. If $d_i$ is the time required to complete a unit quantity of work of task $i$ and $W_{ij}$ is the total quantity of work required in segment $j$ of task $i$, then the actual duration, $D_{ij}$, can be expressed as:

$$D_{ij} = \mu_{ij} d_i W_{ij}$$

(2.1)

where $\mu_{ij}$ is the job condition factor reflecting the effects of variable conditions such as weather, soil conditions, terrain, site congestion, learning effects, etc.

### 2.3 FORMULATION OF THE SCHEDULING OPTIMIZATION PROBLEM

A general mathematical formulation of the scheduling problem is presented in this section. The advantage of such a general formulation is that it can be specialized and reduced for the solution of specific and perhaps less complicated scheduling problems. Further, it can be effectively integrated with the general neural dynamics model for solution of optimization problems developed by Adeli and Park (1995a).

Both non-repetitive and repetitive tasks are considered in the formulation. The non-repetitive tasks correspond to the activities of the traditional network methods such as CPM. A non-repetitive task involves no internal logic as it is performed only once. A repetitive task, on the other hand, may have an elaborate internal logic that connects the segments assigned to various crews. By specifying appropriate constraints, work
continuity considerations and multiple-crew strategies can be modeled. A crew rarely performs at ideal productivity throughout; its performance is affected by the varying job conditions. This is included in our scheduling model by means of a factor, $\mu_v$, which modifies the ideal productivity of a crew to reflect the effect of the job conditions. The external logic of each task is specified by means of a full set of precedence relationships and/or stage (distance) and time buffers.

Development of the general scheduling formulation for a construction project such as highway construction involves the following steps divided into three main categories (headings) as follows:

2.3.1 **Breakdown the work into tasks, crews, and segments**

*Step 1*

Break down the project into $N_T$ tasks. Identify non-repetitive and repetitive tasks. Let $N_{NT}$ and $N_{RT}$ be the number of non-repetitive and repetitive tasks, respectively. If $N_{RT} = 0$, skip steps 2 to 5 and go to step 6.

*Step 2*

For each repetitive task $i$, choose the number of crews to be used ($N_{ci}$). Non-repetitive tasks have only one crew that performs over one segment only.

*Step 3*

Assign $N_{si}^k$ segments of the highway to crew $k$ of repetitive task $i$. The segments are chosen considering the job conditions and quantity of work required, factors affecting the production rate. In addition, predetermined breaks in the work of a crew may influence the
choice of segments. The segments are not required to have equal lengths or constructed in sequence. Each segment is identified by $Z_{ij}^k$ and $Z_{ij}^{k'}$, the beginning and ending distances at which repetitive task $i$ is performed by crew $k$ over segment $j$. Note that each crew of a task is assigned a unique set of segments; two crews cannot perform the same task over the same portion of the highway.

2.3.2 Specify the internal logic of repetitive tasks

For each crew of a repetitive task, do the following:

Step 4

Specify the work continuity relationship between segments $j$ and $j+1$, in the following form:

$$T_{ij}^k + D_{ij}^k + S_{ij}^k \leq T_{(j+1)}^{k(t+1)}$$

(2.2)

where $T_{ij}^k$ is the time at which crew $k$ of task $i$ starts work on segment $j$, $D_{ij}^k$ is the duration of work for crew $k$ of task $i$ on segment $j$, and $S_{ij}^k$ is the idle or slack time of crew $k$ of task $i$ between segments $j$ and $j+1$. For continuity of work, $S_{ij}^k$ must be equal to zero. If a task has only one crew skip step 5 and go to step 6.

Step 5

Define the start of a crew with respect to previous crew(s). The following precedence relationships of start-to-start, finish-to-finish, and start-to-finish are used:

Start-to-start (SS):

$$T_{ii}^k + L_{SS}^k \leq T_{ii}^l$$

(2.3)
Finish-to-finish (FF):

\[ T_{\text{FF}_k}^k + D_{\text{FF}_k}^k + L_{\text{FF}_k}^k \leq T_{\text{FF}_k}^l + D_{\text{FF}_k}^l \quad (2.4) \]

Start-to-finish (SF):

\[ T_{\text{SF}_k}^k + L_{\text{SF}_k}^k \leq T_{\text{SF}_k}^l + D_{\text{SF}_k}^l \quad (2.5) \]

where the superscripts \( l \) and \( k \) refer to the current and the previous crews, respectively; \( L_{\text{SS}_k}^k \), \( L_{\text{FF}_k}^k \), \( L_{\text{SF}_k}^k \) are the start-to-start, finish-to-finish, and start-to-finish time lags between crews \( k \) and \( l \), respectively. These time lags may be given as a function of quantity of work and/or time. If more than one relationship is specified for a particular crew, only one will govern in the final minimum cost schedule obtained from the optimization algorithm. This particular relationship usually is not known in advance and all possible relationships have to be specified in the optimization model.

2.3.3 Specify the external logic of repetitive and non-repetitive tasks

Step 6

Describe the sequencing of the tasks in the project. Each task can be linked with any number of previous tasks by specifying one or more of the following precedence relationships:

Start-to-start (SS):

\[ T_{i,j}^i + L_{\text{SS}_i}^i \leq T_{j,i}^j \quad (2.6) \]

Finish-to-start (FS):

\[ T_{\text{FS}_i}^i + D_{\text{FS}_i}^i + L_{\text{FS}_i}^i \leq T_{j,1}^j \quad k = 1, \ldots, N_{\text{CI}} \quad (2.7) \]
Start-to-finish (SF) (is specified when the task has only one crew):

\[ T_i^l + L_{SFij} \leq T_{j,n} + D_{j,n} \quad l = 1 \] \hspace{1cm} (2.8)

Finish-to-finish (FF) (is specified when both tasks have one crew only)

\[ T_{ln}^k + D_{ln}^k + L_{FFij} \leq T_{j,n} + D_{j,n} \quad k = l = 1 \] \hspace{1cm} (2.9)

The quantities \( L_{SSij}, L_{FSij}, L_{SFij} \) and \( L_{FFij} \) are the respective time lags between task \( j \) and a previous task \( i \). The FS relationship can be used to ensure continuity from one task to another by specifying \( L_{FSij} = 0 \). The relationships represented by Eqs. (2.6)-(2.9) can also be written for any given crew or segment of a task rather than the whole task. For example, consider the case where crew \( B \) of task \( Y \) is the same as crew \( A \) of a previous task \( X \). Crew \( B \) can start work only after crew \( A \) has finished. Therefore, an FS relationship has to be specified between crew \( A \) of task \( X \) and crew \( B \) of task \( Y \).

**Step 7**

Define the space and/or time buffer between tasks. These constraints are essential if interference of crews on different tasks is to be prevented. If task \( i \) precedes task \( j \) by a distance buffer \( B_{ij} \), the following constraints have to be satisfied:

\[ Z_j(T_{in}^k) + B_{ij} \leq Z_{in}^k \quad k = 1,\ldots,N_{ci}, n = 1,\ldots,N_{si}^k \] \hspace{1cm} (2.10)

\[ Z_j(T_{in}^k + D_{in}^k) + B_{ij} \leq Z_{in}^k' \quad k = 1,\ldots,N_{ci}, n = 1,\ldots,N_{si}^k \] \hspace{1cm} (2.11)

\[ Z_{jn}^k + B_{ij} \leq Z_i(T_{jn}^k) \quad k = 1,\ldots,N_{cj}, n = 1,\ldots,N_{si}^k \] \hspace{1cm} (2.12)

\[ Z_{jn}^k + B_{ij} \leq Z_i(T_{jn}^k + D_{jn}^k) \quad k = 1,\ldots,N_{cj}, n = 1,\ldots,N_{si}^k \] \hspace{1cm} (2.13)
The term $Z_i(T_{jm}^k)$ denotes the location of task $i$ at the time $T_{jm}^k$. For tasks with a constant production rate during a segment of work, $Z_i(T_{jm}^k)$ is found by a linear interpolation between the values at the start ($Z_{im}^l$) and the finish ($Z_{im}^l'$) of segment $m$ performed by crew $l$ of task $i$.

$$Z_i(T_{jm}^k) = Z_{im}^l + \frac{(T_{jm}^k - T_{jm}^l)(Z_{im}^l' - Z_{im}^l)}{(T_{jm}^l + D_{im}^l) - T_{jm}^l} \quad (2.14)$$

Similarly, if task $i$ precedes task $j$ by the time buffer $B_{ij}$, then we have the following constraints:

$$T_{jm}^k + B_{ij} \leq T_j(Z_{in}^k) \quad k = 1, \ldots, N_{CI}, n = 1, \ldots, N_{Si}^k \quad (2.15)$$

$$(T_{jm}^k + D_{in}^k) + B_{ij} \leq T_j(Z_{in}^k') \quad k = 1, \ldots, N_{CI}, n = 1, \ldots, N_{Si}^k \quad (2.16)$$

$$T_i(Z_{jn}^k) + B_{ij} \leq T_{jm}^k \quad k = 1, \ldots, N_{Cj}, n = 1, \ldots, N_{Sj}^k \quad (2.17)$$

$$T_i(Z_{jn}^k') + B_{ij} \leq (T_{jm}^k + D_{jn}^k) \quad k = 1, \ldots, N_{Cj}, n = 1, \ldots, N_{Sj}^k \quad (2.18)$$

Likewise, $T_i(Z_{jn}^k)$ is found by a linear interpolation between the starting time ($T_{jm}^l$) and the stopping time ($T_{jm}^l + D_{jm}^l$) for segment $m$ performed by crew $l$ of task $i$.

$$T_i(Z_{jn}^k) = T_{jm}^l + \frac{(Z_{jn}^k - Z_{jn}^l)(T_{jm}^l + D_{jm}^l) - T_{jm}^l}{(Z_{jn}^l' - Z_{jn}^l)} \quad (2.19)$$

The optimization problem can now be formulated as the minimization of direct cost

$$C_D = \sum_{i=1}^{N_{CI}} W_i C_i(d_i) + \sum_{i=1}^{N_{CI}} \sum_{k=1}^{N_{Si}^k} \sum_{j=1}^{N_{Sj}^k} W_{ij}^k C_i(d_j^k), \quad (2.20)$$

subject to the scheduling constraints (Eqs. (2.2)-(2.13) and (2.15)-(2.18)), plus initial constraint.
\[ T_{i1}^l = \text{const}, \]  
(2.21)

project duration constraints

\[ T_i^k + D_i^k \leq D_{\text{max}} \]
\[ i = 1, \ldots, N_T, k = 1, \ldots, N_{C_i}, j = 1, \ldots, N_{S_i}^k, \]
(2.22)

task duration constraints

\[ (d_i^k)^{\text{min}} \leq d_i^k \leq (d_i^k)^{\text{max}} \]
\[ i = 1, \ldots, N_T, k = 1, \ldots, N_{C_i}, \]
(2.23)

and, non-negativity constraints

\[ T_i^k, d_i^k \geq 0 \]
\[ i = 1, \ldots, N_T, k = 1, \ldots, N_{C_i}, j = 1, \ldots, N_{S_i}^k, \]
(2.24)

where \( C_i \) is the direct cost per unit quantity of work for task \( i \); \( d_i^k \) is the time required by crew \( k \) of task \( i \) to complete a unit quantity of work based on resource allocation only; \((d_i^k)^{\text{min}}\), \((d_i^k)^{\text{max}}\) are the minimum and maximum possible values of \( d_i^k \), respectively; and \( D_{\text{max}} \) is the maximum acceptable project duration. Note that in this formulation, Eq. (2.1) can be written for each crew \( k \) of task \( i \) as:

\[ D_i^k = \mu_i^k d_i^k W_i^k \]  
(2.25)

### 2.4 ARTIFICIAL NEURAL NETWORKS AND SCHEDULING

Artificial neural networks (ANN) are a functional abstraction of the biological neural structures of the central nervous system. Their computing abilities have been proven in the fields of prediction and estimation, pattern recognition, and optimization (Adeli and Yeh, 1989; Adeli and Zhang, 1993; Adeli and Hung, 1995; Adeli and Park, 1995a, b, c; Adeli and Park, 1996a). The use of ANN for solving scheduling problems has been reported in the recent literature; however, practically all work is in the area of job-shop scheduling (Gulati et al.,
1987; Watanabe et al., 1993; Willems and Rooda, 1994; Pellerin and Herault, 1994; Foo et al., 1995). Job-shop scheduling is a resource allocation problem in which $n$ jobs have to be scheduled on $m$ machines (the resources) given their operation pattern. The performance criterion is usually the minimization of work completion time. The aforementioned papers pose the problem as an optimization problem and use the Hopfield network (Hopfield and Tank, 1985), or its variations, to solve the problem. The job-shop scheduling problem has been formulated as linear programming (Chang and Nam, 1993), integer programming (Willems and Rooda, 1994) and mixed integer programming (Foo et al., 1995). The ANN models presented in these papers are specific to the particular problem considered. Also, it should be mentioned that the job-shop scheduling problem is an NP-complete problem which requires exhaustive enumeration for solution. Construction scheduling problems, on the other hand, should be formulated as a constrained nonlinear mathematical programming problem.

Recently, Adeli and Park (1995a) developed a nonlinear neural dynamics model as a new optimization technique for solution of complex optimization problems by integrating penalty function method, Lyapunov stability theorem, Kuhn-Tucker conditions, and the neural dynamics concept. The Lyapunov stability theorem guarantees that solutions of the corresponding dynamic system (trajectories) for arbitrarily given starting points approach an equilibrium point without increasing the value of the objective function. This guarantees global convergence and robustness. But, it does not guarantee the equilibrium point is a local minimum. The Kuhn-Tucker conditions are used to verify that the equilibrium point satisfies the necessary conditions for a local minimum. The robustness of the model was demonstrated by application to both linear (Park and Adeli, 1995) and nonlinear (Adeli and Park, 1995b)
structural optimization problems. Most recently, the model was applied to optimization of very large structures including a 144-story super-high-rise building structure with over 20,000 members subjected to actual design specifications (Adeli and Park, 1996b).

An ANN model for the complete scheduling of construction projects has not been presented in the literature. Al-Suwailem and Chang (1994) used a backpropagation learning network to capture human knowledge of allocating construction resources. The ANN determines the size and number of equipment units required for earthmoving processes. Mohammed et al. (1995) formulated the problem of optimally allocating available yearly budget to bridge rehabilitation and replacement projects among a number of alternatives as an optimization problem using the Hopfield network.

2.5 NEURAL DYNAMICS COST OPTIMIZATION MODEL FOR CONSTRUCTION PROJECTS

2.5.1 Formulation

Defining $X = \{T_{ck}^i, d_c^i \mid i = 1, N_I, k = 1, N_C; j = 1, N_N^k\}$ as the vector of decision variables, the optimization problem can be written as:

Minimize

$$C_D = f(X)$$  \hspace{1cm} (2.26)

subject to inequality constraints

$$g_j(X) \leq 0 \quad j = 1, \ldots, J$$  \hspace{1cm} (2.27)

and equality constraints

$$h_k(X) = 0 \quad k = 1, \ldots, K$$  \hspace{1cm} (2.28)
where \( g_j(X) \) is the \( j \)th inequality constraint function, \( h_k(X) \) is the \( k \)th equality constraint function, \( J \) is the total number of inequality constraints, and \( K \) is the total number of equality constraints. Using the exterior penalty function method, a pseudo-objective function is defined as:

\[
P(X, r_n) = f(X) + \frac{r_n}{2} \left( \sum_{j=1}^{J} \left[ g_j^+(X) \right]^2 + \sum_{k=1}^{K} \left[ h_k(X) \right]^2 \right)
\]  

(2.29)

where \( g_j^+(X) = \max \{0, g_j(X)\} \) and \( r_n \) is a penalty parameter magnifying constraint violations.

A dynamic system is defined as:

\[
\frac{dX}{dt} = \dot{X} = F(X)
\]  

(2.30)

where \( X = \{X_1(t), X_2(t), \ldots, X_N(t)\}^T \) is the state vector tracing a trajectory in \( N \)-dimensional space where the superscript \( T \) indicates the transpose of a vector and

\[
N = \sum_{i=1}^{N_x} \sum_{s=1}^{N_s} N_{si}^r + \sum_{i=1}^{N_s} N_{ci}.
\]

The dynamic system evolves until it reaches an equilibrium point. The stability of such an equilibrium point is ensured by satisfying the Lyapunov stability theorem which states that a solution \( \dot{X} \) to the system of differential equations \( X = 0 \) is stable if

\[
\frac{dV}{dt} \leq 0 \quad \text{for all non-zero } X
\]  

(2.31)

where \( V(X) \) is the Lyapunov functional defined as an analytic function of the state variables such that \( V(0) = 0 \) and \( V(X) > 0 \) for all \( |X| > 0 \) (Kolk and Lerman, 1992). The objective (direct cost) function and the constraint functions in our construction cost optimization model
individually satisfy the conditions for a Lyapunov functional. Therefore, the pseudo-objective function \( P \) defined by Eq. (2.29) is also a valid Lyapunov functional, \( V \).

Following Adeli and Park (1995a), by defining

\[
\frac{dX}{dt} = \dot{X} = -\nabla f(X) - r_n \left\{ \sum_{j=1}^J g_j^* \nabla g_j(X) + \sum_{k=1}^K h_k \nabla h_k(X) \right\}
\]  \[ (2.32) \]

where \( \nabla f(X) \), \( \nabla g_j(X) \), and \( \nabla h_k(X) \) are the gradients of the objective function, the \( j \)th inequality constraint, and \( k \)th equality constraint, respectively, the Lyapunov stability theorem for the dynamic system is satisfied.

\[
\frac{dV}{dt} = \left( \frac{dV}{dX} \right) \frac{dX}{dt} = - \left[ \nabla f(X) + r_n \left\{ \sum_{j=1}^J g_j^* \nabla g_j(X) + \sum_{k=1}^K h_k \nabla h_k(X) \right\} \right]^2 \leq 0
\]  \[ (2.33) \]

This also shows that the dynamic system evolves such that the value of the pseudo-objective function always decreases. Equation (2.32) is in fact the learning rule of the neural dynamics model.

For an equilibrium point \( X \) to be a local optimum solution, we also need to satisfy the Kuhn-Tucker optimality conditions:

\[
\frac{\partial L}{\partial X_i} = \frac{\partial f(X)}{\partial X_i} + \sum_{j=1}^J u_j \frac{\partial g_j(X)}{\partial X_i} + \sum_{k=1}^K v_k \frac{\partial h_k(X)}{\partial X_i} = 0; \quad i = 1, \ldots, N
\]  \[ (2.34) \]

\[
g_j(X) + s_j^2 = 0; \quad j = 1, \ldots, J
\]  \[ (2.35) \]

\[
h_k(X) = 0; \quad k = 1, \ldots, K
\]  \[ (2.36) \]

\[
u_j s_j = 0; \quad j = 1, \ldots, J
\]  \[ (2.37) \]
\[ u_j \geq 0; \quad j = 1, \ldots, J \]  

(2.38)

\[ v_k = \text{unrestricted in sign} \]  

(2.39)

where \( L \) is the Lagrangian function defined as a linear combination of the objective and constraint functions:

\[ L(X, u, v, s) = f(X) + \sum_{j=1}^{J} u_j \left[ g_j(X) + s_j^2 \right] + \sum_{k=1}^{K} v_k h_k(X), \]  

(2.40)

in which \( s_j \) is the slack term for the \( j \)th inequality constraint, and \( u_j \) and \( v_k \) are the Lagrangian multipliers corresponding to the \( j \)th inequality and \( k \)th equality constraint, respectively.

Finally, the optimum solution to the direct cost optimization problem can be found by the integration:

\[ X = \int \dot{X} dt. \]  

(2.41)

This integration can be performed by the Euler or Runge-Kutta method.

2.5.2 Topological characteristics

The neural network topology for the neural dynamics construction cost optimization model is shown in Figure 5. The nodes in the network represent the variables and constraints of the problem. The variable layer has \( N \) nodes corresponding to the total number of decision variables. The constraint nodes are divided into \( N_{NR} \) layers corresponding to non-repetitive tasks, \( N_{RT} \) layers corresponding to repetitive tasks, and an initial constraint node. Nodes are grouped within each layer into the constraint categories described in a previous section. Variable and constraint nodes are fully interconnected
Figure 5. The neural network topology for the neural dynamics cost optimization model.
(interlayer connections). In addition, recurrent and intra-layer connections are also used to be described shortly.

Associated with each connection is a weight whose magnitude and sign affect the impulse the connected node will receive. Both excitatory (positive connection weights) and inhibitory (negative connection weights) connections are used in our model. The coefficients of the constraint functions are assigned to the excitatory connections from the variable layer to the constraint nodes. The gradients of the constraint functions are assigned to the inhibitory connections from the constraint nodes to the variable layer. The gradients of the objective function are assigned to the recurrent inhibitory connections of the variable layer. A weight of one is assigned to the intra-layer connections. This allows the outputs of nodes in competition to be compared.

The output of the variable layer is the current state vector \( \mathbf{X} \). As the coefficients of the constraint functions are encoded in the excitatory connections from the variable layer to the constraint nodes, the input to a constraint node is the magnitude of the constraint at any given state, that is, \( g_j(\mathbf{X}) \) for an inequality constraint \( j \), and \( h_k(\mathbf{X}) \) for an equality constraint \( k \). The output of a constraint node will depend on the type of the constraint it represents. For an inequality constraint \( j \), the output is:

\[
O_{c} = \begin{cases} 
0 & \text{when } g_j(\mathbf{X}) \leq 0 \\
 r_j g_j(\mathbf{X}) & \text{when } g_j(\mathbf{X}) > 0 
\end{cases} \quad (2.42)
\]

and for an equality constraint \( k \) the output is:

\[
O_{e} = \begin{cases} 
0 & \text{when } h_k(\mathbf{X}) = 0 \\
 r_k h_k(\mathbf{X}) & \text{when } h_k(\mathbf{X}) \neq 0 
\end{cases} \quad (2.43)
\]
Equations (2.42) and (2.43) represent the activation functions. They are chosen such that the output of a constraint node is the penalized constraint violation. When more than one equation is specified for a particular category of constraint, such as external logic constraint, a competition is created between the outputs of the nodes in that group. For a group of \( n \) nodes with outputs \( O_{c1}, O_{c2}, \ldots, O_{c\delta}, \ldots, O_{cn} \) such that

\[
O_{c\delta} = \max\{O_{c1}, O_{c2}, \ldots, O_{c\delta}, \ldots, O_{cn}\}
\]  

(2.44)

The outputs after competition are as follows:

\[
O_{c\delta} = O_{c\delta} \quad \text{and} \quad O_{c1}, O_{c2}, \ldots, O_{cn} = 0
\]  

(2.45)

(2.46)

Let \( w_{ji} \) and \( w_{\delta} \) be the connection weight from the \( j \)th and \( \delta \)th inequality and equality constraint node, respectively, to the \( i \)th variable node and \( Y_i \) be the weight of the recurrent connection to a node \( i \) in the variable layer. Then, the input to the \( i \)th variable node is given by:

\[
I_i = Y_i + \sum_{j=1}^{J} w_{ji} O_{\delta} + \sum_{k=1}^{K} w_{\delta k} O_{ck}
\]  

(2.47)

The new value of the \( i \)th decision variable is obtained by the integration:

\[
X_i^{new} = \int I_i \, dt
\]  

(2.48)

This integration is done by the Euler or the Runge-Kutta methods. In the construction cost optimization problem we found the simple Euler method to yield accurate results.
The network operates until no change in the decision variables occur within a given
tolerance, that is, when $\dot{X} = 0$. $X$ is the solution to the minimum direct cost construction
scheduling problem.

2.6 ILLUSTRATIVE EXAMPLE

A 5 km-long two-lane highway construction project is used to illustrate the capabilities
of the computational model presented in this chapter. The work required is divided into 14
repetitive and non-repetitive tasks summarized in Table 1. Tasks 1 to 5 represent the
establishment of a temporary site office at the beginning of the 5 km long stretch. The
errection of an asphalt concrete plant at a distance of 2.5 km from the beginning of the
roadway (at the center of the project) is represented by tasks 6 and 7 (together with
portion of task 9).

2.6.1 Cost-duration relationship

The relationship between direct cost and duration for unit quantity of work for each
task is given in Table 2. The direct cost-duration relationship for task 2 (a linear
relationship) and task 10 (a nonlinear relationship) are shown in Figures 6 and 7 as
examples. An initial cost of $5000 and, thereafter, a daily cost of $500 is used as the
indirect cost for this example.
<table>
<thead>
<tr>
<th>Task #</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Clear and grub site for temporary offices plus right-of-way</td>
<td>Non-repetitive</td>
</tr>
<tr>
<td>2</td>
<td>Grade site for temporary offices</td>
<td>Non-repetitive</td>
</tr>
<tr>
<td>3</td>
<td>Erect temporary offices</td>
<td>Non-repetitive</td>
</tr>
<tr>
<td>4</td>
<td>Construct temporary roads</td>
<td>Non-repetitive</td>
</tr>
<tr>
<td>5</td>
<td>Move-in</td>
<td>Non-repetitive</td>
</tr>
<tr>
<td>6</td>
<td>Grade asphalt concrete plant site</td>
<td>Non-repetitive</td>
</tr>
<tr>
<td>7</td>
<td>Erect asphalt concrete plant</td>
<td>Non-repetitive</td>
</tr>
<tr>
<td>8</td>
<td>Construct culverts</td>
<td>Repetitive</td>
</tr>
<tr>
<td>9</td>
<td>Clear and grub right-of-way</td>
<td>Repetitive</td>
</tr>
<tr>
<td>10</td>
<td>Earthwork</td>
<td>Repetitive</td>
</tr>
<tr>
<td>11</td>
<td>Lay sub-base</td>
<td>Repetitive</td>
</tr>
<tr>
<td>12</td>
<td>Lay base</td>
<td>Repetitive</td>
</tr>
<tr>
<td>13</td>
<td>Pave</td>
<td>Repetitive</td>
</tr>
<tr>
<td>14</td>
<td>Finish shoulders</td>
<td>Repetitive</td>
</tr>
</tbody>
</table>

Table 1. The description and type of tasks in the illustrative example

<table>
<thead>
<tr>
<th>Task #</th>
<th>Direct cost-duration relationship</th>
<th>Range (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$C = -300d +1050$</td>
<td>$1.0 \leq d \leq 1.5$</td>
</tr>
<tr>
<td>2</td>
<td>$C = -280d + 960$</td>
<td>$0.5 \leq d \leq 2.0$</td>
</tr>
<tr>
<td>3</td>
<td>$C = -200d + 250$</td>
<td>$0.25 \leq d \leq 0.50$</td>
</tr>
<tr>
<td>4</td>
<td>$C = -200d + 550$</td>
<td>$0.50 \leq d \leq 1.25$</td>
</tr>
<tr>
<td>5</td>
<td>$C = -150d + 550$</td>
<td>$1.0 \leq d \leq 2.0$</td>
</tr>
<tr>
<td>6</td>
<td>$C = -280d + 960$</td>
<td>$0.5 \leq d \leq 2.0$</td>
</tr>
<tr>
<td>7</td>
<td>$C = -400d + 5700$</td>
<td>$5 \leq d \leq 8$</td>
</tr>
<tr>
<td>8</td>
<td>$C = 1600 / d$</td>
<td>$2 \leq d \leq 3$</td>
</tr>
<tr>
<td>9</td>
<td>$C = -300d + 1050$</td>
<td>$1.0 \leq d \leq 1.5$</td>
</tr>
<tr>
<td>10</td>
<td>$C = (1600 + 500d) / d$</td>
<td>$1.0 \leq d \leq 2.0$</td>
</tr>
<tr>
<td>11</td>
<td>$C = -200d + 850$</td>
<td>$0.75 \leq d \leq 1.25$</td>
</tr>
<tr>
<td>12</td>
<td>$C = -200d + 950$</td>
<td>$0.75 \leq d \leq 1.25$</td>
</tr>
<tr>
<td>13</td>
<td>$C = -200d + 900$</td>
<td>$0.75 \leq d \leq 1.25$</td>
</tr>
<tr>
<td>14</td>
<td>$C = -100d + 800$</td>
<td>$2 \leq d \leq 4$</td>
</tr>
</tbody>
</table>

Table 2. The direct cost-duration relationship for each task
Figure 6. The linear direct cost-duration curve for unit quantity of task 2
Figure 7. The nonlinear direct cost-duration curve for unit quantity of task 10
2.6.2 Scheduling logic

The way in which each task is performed and the logic in which the tasks are carried out for a given project is not always well defined. Different schedulers may have different ideas for breaking down and sequencing each task. Often schedulers are constrained by the scheduling model available, forcing them to make simplifying assumptions. The flexible computational model presented in this chapter, however, allows schedulers a greater control over the progress of work and enables them to complete the job more efficiently.

Details of the breakdown of repetitive tasks into crews and segments, the start and finish distances, the quantities of work required, and the job condition factors for segments of work are given in Table 3. A constant number (1000 m) is used as the start distance of the project to avoid division by zero in the computation.

How the variation in the quantities of work and the job condition factors affect the breakdown of tasks can be explained by the clear and grub operations represented by tasks 1 and 9. Figure 8 shows the areas that have to be cleared and grubbed and the type of vegetation involved. Task 1 operates on the first 200 m of the roadway but also includes the area for the site office. Task 9 covers the remaining length of the highway including the site for the asphalt concrete plant. A new segment of work is required whenever there is a change in the quantity of work required per unit length of the highway and/or a change in the job condition factor. Each change will affect the production rate. To reflect such a change a separate segment of work is defined. Whenever there is no such changes, such as for task 14, there is no need to break down the work into smaller segments.
<table>
<thead>
<tr>
<th>Task #</th>
<th>Crew #</th>
<th>Segment #</th>
<th>Distances (m)</th>
<th>Quantity of work</th>
<th>Unit</th>
<th>μ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Start</td>
<td>Finish</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1000</td>
<td>1200</td>
<td>3</td>
<td>hectares</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-</td>
<td>1000</td>
<td>1200</td>
<td>3</td>
<td>hectares</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
<td>1000</td>
<td>1200</td>
<td>5</td>
<td>units</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-</td>
<td>1000</td>
<td>1200</td>
<td>3</td>
<td>100 m</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-</td>
<td>1000</td>
<td>1200</td>
<td>100</td>
<td>percent</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>-</td>
<td>3500</td>
<td>3650</td>
<td>1.5</td>
<td>hectares</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>-</td>
<td>3500</td>
<td>3650</td>
<td>100</td>
<td>percent</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1300</td>
<td>1305</td>
<td>1</td>
<td>culvert</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2750</td>
<td>2755</td>
<td>1</td>
<td>culvert</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5500</td>
<td>5505</td>
<td>1</td>
<td>culvert</td>
<td>1.0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1200</td>
<td>3000</td>
<td>4.5</td>
<td>hectares</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3000</td>
<td>3500</td>
<td>1.25</td>
<td>hectares</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3500</td>
<td>3650</td>
<td>1.875</td>
<td>hectares</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3650</td>
<td>6000</td>
<td>5.875</td>
<td>hectares</td>
<td>1.15</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>3000</td>
<td>10</td>
<td>1000 m³</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3000</td>
<td>5000</td>
<td>6</td>
<td>1000 m³</td>
<td>1.15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3500</td>
<td>5000</td>
<td>8</td>
<td>1000 m³</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5000</td>
<td>6000</td>
<td>5</td>
<td>1000 m³</td>
<td>1.0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>2000</td>
<td>4.25</td>
<td>1000 m³</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2000</td>
<td>4000</td>
<td>8.5</td>
<td>1000 m³</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4000</td>
<td>6000</td>
<td>8.5</td>
<td>1000 m³</td>
<td>1.0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>3500</td>
<td>2250</td>
<td>2.6</td>
<td>1000 m³</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2250</td>
<td>1000</td>
<td>2.6</td>
<td>1000 m³</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3500</td>
<td>4750</td>
<td>2.6</td>
<td>1000 m³</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4750</td>
<td>6000</td>
<td>2.6</td>
<td>1000 m³</td>
<td>1.05</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>3500</td>
<td>2250</td>
<td>1.25</td>
<td>1000 m³</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3250</td>
<td>1000</td>
<td>1.25</td>
<td>1000 m³</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3500</td>
<td>4750</td>
<td>1.25</td>
<td>1000 m³</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4750</td>
<td>6000</td>
<td>1.25</td>
<td>1000 m³</td>
<td>1.05</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>6000</td>
<td>3</td>
<td>hectares</td>
</tr>
</tbody>
</table>

Table 3. Task details for the illustrative example
Figure 8. The areas and types of vegetation that have to be cleared by tasks 1 and 9

Base laying and paving operations (tasks 12 and 13) require material from the asphalt concrete plant. Therefore, as the operation moves away from the plant more time will be taken to do the same amount of work. In our example, we increased the time required for operations beyond 1250 m from the plant by 5 percent indicated by the job condition factor of 1.05. Instead of a step function, a continuous linear or nonlinear function may be used for the job condition factor that will reflect the impact of increasing haul distances on the rate of operation.

The internal logic of repetitive tasks is given in Table 4. Work continuity relationships between segments of work and multiple-crew strategies are specified. Usually no slack time ($S^i_0 = 0$) is allowed between segments of the work of a crew. However, the slack
<table>
<thead>
<tr>
<th>Task #</th>
<th>Crew #</th>
<th>Segment #</th>
<th>Continuity relationship</th>
<th>Multiple-crew strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Predecessor crew</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Relationship</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>$S = 1 - \beta^a$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>$S = 1 - \beta^a$</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>$S = 0$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>$S = 0$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>$S = 0$</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>$S = 0$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>SS, L = 0</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>$S = 0$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>$S = 0$</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>$S = 0$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>FF, L = 0</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>$S = 0$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>SS, L = 0</td>
<td>-</td>
</tr>
</tbody>
</table>

$^a$ $\beta$ is the fractional portion of the finishing time of the previous segment of work

Table 4. The internal logic of repetitive tasks

term can be any function of the decision variables. We use a nonlinear slack term to model the continuity of work constraint of task 8 in the form:

$$S = 1 - \beta \leq 1.0$$

(2.49)

where $\beta < 1.0$ is the fractional portion of the finishing time (starting time plus duration) of the previous segment of work. For example, for a finishing time of 10 days and 2
<table>
<thead>
<tr>
<th>Task</th>
<th>Predecessor</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 2</td>
<td>Task 1</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td>Task 3</td>
<td>Task 2</td>
<td>SS, $L = 0$</td>
</tr>
<tr>
<td>Task 4</td>
<td>Task 4</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td>Task 5</td>
<td>Task 2</td>
<td>FS, $L = 0.25D$</td>
</tr>
<tr>
<td>Task 6</td>
<td>Task 3</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td>Task 7</td>
<td>Segment 3, Crew 1, Task 9</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td>Task 8: Crew 1, Segment 1</td>
<td>Task 9 at 1300 m</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td>Crew 1, Segment 2</td>
<td>Task 9 at 2750 m</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td>Crew 1, Segment 3</td>
<td>Task 9 at 5500 m</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td>Task 9</td>
<td>Task 1</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td>Task 10</td>
<td>Task 9</td>
<td>Space buffer, $B = 150$ m</td>
</tr>
<tr>
<td>Task 11</td>
<td>Task 10</td>
<td>Space buffer, $B = 150$ m</td>
</tr>
<tr>
<td>Task 12</td>
<td>Task 7</td>
<td>FS, $L = 0$</td>
</tr>
<tr>
<td></td>
<td>Task 11</td>
<td>Time buffer, $B = 2$ days</td>
</tr>
<tr>
<td>Task 13</td>
<td>Task 12</td>
<td>Time buffer, $B = 2$ days</td>
</tr>
<tr>
<td>Task 14</td>
<td>Task 13</td>
<td>Time buffer, $B = 2$ days</td>
</tr>
</tbody>
</table>

*Table 5. The external logic of tasks*
\[ D = \text{Duration in days} \]
\[ L = \text{Lag in days} \]

**Figure 9.** Activity-on-node diagram for the first five tasks of the illustrative example

hours (10.25 days assuming 8 hours per day) \( \beta = 0.25 \). This insures that the work on the next segment will start on the following day. As a result, adequate time is provided for the crew to move from one location to the next. Multiple-crew strategies become important when more than two crews are used.

Table 5 gives the external logic of tasks for the illustrative example. The external logic of the first 5 tasks, which are non-repetitive, can also be shown by an activity-on-node (AON) diagram (Figure 9). Standard precedence relationships, Eqs. (2.6) to (2.9), are used to link the tasks. The time lag term, however, may be any function of the decision variables.

The construction of a culvert cannot start unless the area has been cleared and grubbed. Therefore, the external logic of task 8 requires that work on any culvert be delayed until the
crews of repetitive task 9 has worked through the corresponding location. A space buffer of 150 m is provided around earthmoving operations (task 10) to make sure adequate space is available for the equipment. Tasks 12 and 13 cannot start before the completion of the asphalt concrete plant. A minimum time buffer of 2 days is provided between tasks 11, 12, 13 and 14.

2.6.3 Solution of the problem

The direct cost optimization problem is solved for project durations of 60, 65, 70, 80, 90 and 100 days. The penalty parameter, \( r_n \), is taken as (Adeli and Park, 1995a):

\[
r_n = r_0 + \frac{n}{\alpha}
\]

(2.50)

where \( r_0 \) is the initial penalty, \( n \) the iteration number and \( \alpha \) is a positive number. Through this relationship the penalty is increased gradually in each iteration to avoid the possibility of numerical ill-conditioning. As stopping criteria, a change of less than $1 in the original objective (direct cost) function and a maximum of 450 iterations are chosen. The convergence curves for the solutions are given in Figure 10. Table 6 and Figure 11 show the variation of direct, indirect, and total costs for different values of project duration. From Figure 11 and Table 6, a project duration of 70 days leads to the minimum total cost. The final global optimum schedule is shown as a linear planning chart in Figure 12.

2.7 CONCLUSION

A general formulation was presented for the scheduling of construction projects. Both repetitive and non-repetitive tasks are considered in the formulation. By specifying appropriate
<table>
<thead>
<tr>
<th>Duration (days)</th>
<th>Direct cost (dollars)</th>
<th>Indirect cost (dollars)</th>
<th>Total cost (dollars)</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>94118</td>
<td>30000</td>
<td>124118</td>
</tr>
<tr>
<td>65</td>
<td>91215</td>
<td>32500</td>
<td>123715</td>
</tr>
<tr>
<td>70</td>
<td>87314</td>
<td>35000</td>
<td>122414</td>
</tr>
<tr>
<td>80</td>
<td>85742</td>
<td>40000</td>
<td>125742</td>
</tr>
<tr>
<td>90</td>
<td>85438</td>
<td>45000</td>
<td>130438</td>
</tr>
<tr>
<td>100</td>
<td>84411</td>
<td>50000</td>
<td>134411</td>
</tr>
</tbody>
</table>

Table 6. The direct, indirect, and total costs variation for the illustrative example

constraints, work continuity considerations and multiple-crew strategies can be modeled. The effects of varying job conditions on the performance of a crew is taken into account by introducing a job conditions factor which modifies the task duration computed on the basis of resource allocation only. This factor can be a constant, a linear, or a nonlinear function depending on the complexity of the situation. An optimization formulation is presented for the construction project scheduling problem with the goal of minimizing the direct construction cost. Any linear or non-linear function can be used for task direct cost-duration relationships. The nonlinear optimization problem is then solved by the neural dynamics model developed recently by Adeli and Park (1995a). For any given construction duration, the model yields the optimum construction schedule for the minimum construction direct cost automatically. By varying this construction duration, one can solve the cost-duration trade-off problem and obtain the global optimum schedule and the corresponding minimum construction cost.

The new scheduling construction model provides the capabilities of both CPM and LSM approaches. In addition, it provides feature desirable for repetitive tasks such as highway
construction projects and allows schedulers greater flexibility in modeling construction projects more accurately. In particular, it is suitable for studying the effects of change order on the construction cost. The new scheduling model can be specialized for the solution of specific and perhaps less complicated scheduling problems.

Figure 9. The direct cost convergence curves for the illustrative example
Figure 10. Time-cost trade-off curve for the illustrative example
Figure 11. The linear planning chart for the minimum total cost schedule
CHAPTER 3

NEURAL NETWORK MODEL FOR OPTIMIZATION OF

COLD-FORMED STEEL BEAMS

3.1 INTRODUCTION

An important advantage of cold-formed steel is the greater flexibility of cross-sectional shapes and sizes available to the structural steel designer. Through cold-forming operations, steel sheets, strips or plates can easily be shaped and sized to meet a large variety of design options. Such a large number of design possibilities creates a very important challenge: How to choose the most economical cold-formed shape in design of steel structures.

Mathematical optimization techniques have been used for the optimal proportioning of structural steel members. Abuyounes and Adeli (1986) used the general geometric programming (GGP) technique to minimize the weight of a simply supported plate girder subjected to uniformly distributed load and/or any number of concentrated loads. Adeli and Chompooming (1988 and 1989) extended their work to the optimal proportioning of multispans prismatic and non-prismatic plate girders. A knowledge-based system for optimum design of plate girder bridges is presented by Adeli and Mak (1988). However,
to the best of our knowledge, no research has been reported on the optimization of cold-formed steel beams in the literature.

In this work, we adapt the neural dynamics optimization model developed recently by Adeli and Park (1995a) for minimum weight design of simply supported cold-formed steel beams based on two different specifications: (1) The American Iron and Steel Institute's (AISI) Specification for the Design of Cold-formed Steel Structural Members (AISI, 1989), hereafter referred to as the AISI ASD Specification, and (2) the AISI Load and Resistance Factor Specification for the Design of Cold-Formed Steel Structural Members (AISI, 1991), hereafter referred to as the AISI LRFD Specification. The loading on the beams is assumed to be a uniformly distributed load and/or any number of concentrated loads. Full lateral bracing, no lateral bracing, or lateral bracing at any number of specified points along the length of the beam may be assumed.

3.2 MINIMUM WEIGHT DESIGN OF COLD-FORMED STEEL BEAMS

The optimization problem is defined as:

Minimize

\[ W = F(X) \]  

subject to

\[ g_j^k(X) \leq 0 \quad j = 1, \ldots, k = 1, N_L \]  

\[ X^L \leq X \leq X^U \]  

where \( W \) is the weight of the structure, \( X \) is the vector of design variables, \( i \) is the number of design constraints, \( N_L \) is the number of load cases, and \( X^L \) and \( X^U \) are the lower and

40
Figure 13. The variables involved in the design of hat, I, and Z-shaped cold-formed steel beams
upper bounds on the design variables, respectively. In the design of cold-formed steel beams, \( X \) consists of the cross-sectional dimensions of the members. The exact form of the objective function and the constraints depend on the shape of the section used.

The optimization model presented in this article is general and can be applied to any kind of cross-section. As examples, however, we consider three commonly-used shapes: hat, I, and Z shapes. The variables involved in their design are shown in Figure 13. For these shapes, the weight (objective function) is defined as:

\[
W = \rho L X \left[ X_1 + X_3 + N_w (X_2 + 0.5\pi r^2) \right]
\]  

(3.4)

where \( \rho \) is the unit weight of steel, \( L \) is the span length of the beam, \( N_w \) is the number of webs (equal to 1 for Z shape and 2 for hat and I shapes), and \( X_1, X_2, X_3, \) and \( X_4 \) are the cross-sectional dimensions identified in Figure 13. The parameter \( r \) is the radius of curvature at the junction of plate elements.

The design constraints are those specified by the AISI ASD or LRFD Specifications. In the AISI ASD approach, no load factors are applied to the nominal loads to obtain the design loads in the evaluation of the design strengths. In the AISI LRFD approach, on the other hand, various linear combinations of the nominal loads are used for the design loads which are then compared with the design strengths. The design strength is the nominal strength divided by a factor of safety (\( \Omega \)) in the AISI ASD approach, and the nominal strength multiplied by a resistance factor (\( \phi \)) in the AISI LRFD approach. In the following paragraphs, the design constraints are formulated in a general form applicable to both AISI ASD and LRFD Specifications. The terms design loads and design strengths, however, depend on the design approach used.
1. Bending strength constraint. The maximum design bending moment, \( M_{\text{max}} \), must be less than or equal to the design bending strength, \( M_d \).

\[
\frac{M_{\text{max}}}{M_d} - 1 \leq 0
\]  

(3.5)

For beams with full lateral support, \( M_d \) is the minimum value obtained from considerations of initiation of yielding or inelastic reserve capacity and the shear lag effects. If the beam does not have full lateral support Eq. (3.5) has to be satisfied in each unbraced segment where \( M_{\text{max}} \) is the maximum design bending moment in the segment under consideration and \( M_d \) is the design lateral buckling strength of that segment.

Unlike hot-rolled shapes, cold-formed steel shapes are characterized by small wall thicknesses. As a result, elements under compression can buckle locally at stresses much lower than the yield stress. To simulate the non-uniform distribution of stresses in such compression elements the concept of an effective width is used (AISI, 1989 and 1991). In this approach, the lightly stressed portion of the element is assumed ineffective in resisting stresses. The stress distribution over the remaining portion is assumed to be constant. Figure 14 shows the effective widths of cold-formed steel shapes in bending. The effective width \( b \) of the compression flange depends on the normal stress \( f \) in the flange, the flat-width-to-thickness ratio \( (X_1/X_4) \), and the plate buckling coefficient. Figures 14a and 14b represent shapes with stiffened and unstiffened compression flanges, respectively. The effective widths \( b_1 \) and \( b_2 \) of the web are a function of the maximum compressive and tensile stresses \( f_1 \) and \( f_2 \) (Figure 14) and the flat-width-to-thickness ratio \( (X_2/X_4) \).
Figure 14. Effective design widths of cold-formed steel shapes in bending. (a) Shapes with a stiffened compression flange, and (b) Shapes with an unstiffened compression flange

Considering the local buckling effects, \( M_d = S_e F_y / \Omega \) for ASD and \( M_d = \phi_b S_e F_y \) for LRFD where \( S_e \) is the effective section modulus, \( \Omega = 1.67 \) is the factor of safety in bending, \( \phi_b \) is the resistance factor for bending strength (equal to 0.90 for I, Z, and all other shapes not braced throughout, and 0.95 for fully braced hat shapes), and \( F_y \) is the yield stress of steel. The dependence of effective widths on stresses means that \( S_e \) (and other cross-sectional properties) cannot be calculated explicitly. Following the AISI ASD and LRFD Specifications, the following stepwise procedure is used to determine the value of \( S_e \) at initiation of yielding in the cross-section:

Step 1: Assume the compression flange yields, i.e. \( f = F_y \). Calculate \( b \) and the cross-sectional properties including the distance of the neutral axis from the outermost
compression fiber, $y_{cg}$. If $y_{cg}$ is greater than $d/2$ where $d$ is the depth of the cross-section, go to step 2. If $y_{cg}$ is less than $d/2$, skip step 2 and go to step 3.

**Step 2.** The assumption $f = F_y$ is correct. Calculate widths $b_1$ and $b_2$ based on section properties calculated in step 1. If $b_1 + b_2 \geq h_c$ where $h_c$ is the flat height of the compression region of the web, stop. The web is fully effective as assumed and the calculated properties are correct. Otherwise, set the iteration number $n = 1$ and do the following:

1. Set $y_{cg}^n = y_{cg}$.

2. Calculate the cross-sectional properties with effective widths $b$, $b_1$, and $b_2$.

3. If $y_{cg} - y_{cg}^n < \varepsilon$ where $\varepsilon$ is the stopping tolerance, stop. The current properties are the correct values within the tolerance.

4. Update $b_1$ and $b_2$ ($b$ remains unchanged as $y_{cg} > d/2$ and $f = F_y$).

5. $n = n + 1$.

6. Go to 1.

**Step 3.** The assumption that yielding initiates in the compression flange is not correct. Set the iteration number $n = 1$ and do the following:

1. Set $y_{cg}^n = y_{cg}$.

2. Calculate $f$ with neutral axis at $y_{cg}^n$ and the maximum stress in the tension flange at $F_y$. Calculate $b$.

3. Determine cross-sectional properties with effective width $b$ and assuming the web is fully effective.
4. If \( y_{cs}^n - y_{cs} < \varepsilon \) stop. Calculate \( b_1 \) and \( b_2 \). If \( b_1 + b_2 \geq h_c \) the assumption that the web is fully effective is correct and the calculated properties are the desired effective cross-sectional properties. Otherwise, set iteration counter \( m = 0 \) and do the following:

a. Set \( y_{cs}^m = y_{cs} \).

b. Calculate the cross-sectional properties with effective widths \( b, b_1, \) and \( b_2 \).

c. If \( y_{cs}^m - y_{cs} < \varepsilon \) stop. The current properties are the correct values within the tolerance.

d. Update \( b, b_1, \) and \( b_2 \).

e. \( m = m + 1 \).

f. Go to a.

5. Update \( f \) and \( b \).

6. \( n = n + 1 \).

7. Go to 1.

Some cold-formed steel shapes may be sufficiently compact to develop partial plastification. The AISI ASD and the LRFD Specifications recognize this inelastic reserve capacity and allow designers to increase the bending capacity by 25 percent provided certain provisions are satisfied. Our computational model takes into account this increase whenever the conditions are satisfied.

For short span beams with unusually wide flanges supporting concentrated loads the effective widths of both compression and tension flanges may be limited by shear lag
effect. Shear lag causes the normal flexural stresses in the flanges to decrease with increasing distance from the web. As the span-to-flange width ratio increases the effective width of the flange decreases. For beams in which shear lag is important the design bending strength is the minimum of the values computed from local buckling considerations and shear lag effects.

The bending strength of laterally unbraced segments depends on the length of the segment, the variation of bending moment over the segment, the depth of the shape \((d)\), and the moment of inertia of the compression portion of the shape with respect to the minor axis of bending. Note that the hat shape is laterally stable when bending with the top flange in compression.

2. **Shear strength constraint.** The maximum design shear, \(V_{\text{max}}\), must be less than or equal to the design shear strength, \(V_d\).

\[
\frac{V_{\text{max}}}{V_d} - 1 \leq 0 \quad (3.6)
\]

Yielding and local shear buckling of the web is controlled by this constraint. The appropriate relationship for \(V_d\) depends on the flat height \((X_2)\), and the thickness \((X_4)\) of the web.

3. **Constraint on combined bending and shear strength.** Bending and shear do not occur in isolation but rather interact with each other. The AISI ASD and the LRFD Specifications require that the following nonlinear interaction equation be satisfied throughout the beam:

47
\[
\left( \frac{M_x}{M_d} \right)^2 + \left( \frac{V_x}{V_d} \right)^2 - 1.0 \leq 0
\]  

(3.7)

where \(M_x\) and \(V_x\) are the design bending moment and shear at the location \(x\), respectively.

To find the location at which the constraint is most violated we evaluate the above expression at a number of equally spaced locations (including locations of concentrated loads) and include the one with the largest absolute value in the constraint set.

4. **Constraint on web crippling strength.** In cold-formed steel construction, it is usually impractical to provide stiffeners at reactions and locations of concentrated loads. Because of the large flat-width-to-thickness ratios of webs, crippling is an important consideration in the design of cold-formed steel members. To prevent web crippling the following constraints must be satisfied:

\[
\frac{R_A}{P_d} - 1 \leq 0
\]  

(3.8)

\[
\frac{R_B}{P_d} - 1 \leq 0
\]  

(3.9)

\[
\frac{P_i}{P_d} - 1 \leq 0 \quad i = 1, N
\]  

(3.10)

where \(R_A\) and \(R_B\) are the left and right beam reactions, respectively, and \(P_i\) is the design concentrated load \(i\) located at a distance \(x_i\) from the left support. The design web crippling strength, \(P_d\), is given by four formulas for a given shape. The appropriate formula to be used depends on: (1) the distance of the concentrated load from the beam support, and (2) the spacing between the concentrated load and the nearest concentrated load in the
opposite direction. All distances are measured from the edge of the bearing plates whose width, $B$, is a parameter in the formulas.

5. **Constraint on combined web crippling and bending strength.** The combined effect of concentrated load and bending is accounted for by the following constraint:

$$ P \left( \frac{P_i}{P_d} \right) + Q \left( \frac{M_i}{M_d} \right) - 1.0 \leq 0 \quad i = 1, N; N \neq 0 $$  \hspace{1cm} (3.11)\]

where $M_i$ is the design moment at the location of concentrated load $i$, and $P$ and $Q$ are parameters depending on the specification used, i.e. AISI ASD or LRFD Specification.

6. **Deflection constraint.** Maximum deflection, $\Delta_{\text{max}}$, of the beam must be less than or equal to the allowable deflection, $\Delta_a$.

$$ \left( \frac{\Delta_{\text{max}}}{\Delta_a} \right) - 1.0 \leq 0 $$  \hspace{1cm} (3.12)\]

The maximum deflection is determined at service loads. Because the effective widths of the compression elements depend on flexural stresses the moment of inertia varies with the bending moment along the length of the beam. The effective moment of inertia used is that at maximum bending moment. This results in a small error which is on the conservative side (Yu, 1992). The effective cross-sectional properties for deflection determination are calculated by a similar procedure to that used for bending strength determination.

7. **Constraint on flange curling.** Curling of both tension and compression flanges towards the neutral axis is a concern for unusually wide and thin flanges. Excessive curling reduces the bending capacity of the section and impairs its appearance. To limit flange curling to a given value, $c_f$, the following constraints must be satisfied:
\[
\frac{X_1 + r}{N_w} - \frac{w_f}{1.0} \leq 0 
\] (3.13)

\[
\frac{X_3 + r}{N_w} - \frac{w_f}{1.0} \leq 0 
\] (3.14)

where \(w_f\) represents the limiting value for the length of the flange projecting beyond the web for unstiffened flanges (the I and Z shapes) or half the distance between webs for stiffened flanges (the hat shape) and is given by

\[
w_f = \left( \frac{0.061dX_4E}{f_{av}} \right)^{\frac{1}{2}} \left( \frac{100c_f}{d} \right)^{\frac{1}{4}} 
\] (3.15)

in which \(f_{av}\) is the average stress in the full, unreduced flange width. Curling on the order of 5 percent of the beam depth is not uncommon. Appearance considerations may dictate the choice of \(c_f\).

8. Local buckling constraints. The AISI ASD and the LRFD Specifications require that the flat-width-to-thickness ratio of the compression flange be limited to 500 for stiffened flanges and 60 for unstiffened flanges. For beams with no web stiffeners, which is often the case, the web flat-width-to-thickness ratio is limited to 200.

For flange local buckling, when the top flange is in compression:

\[
\frac{X_1}{X_4} - \frac{500}{1.0} \leq 0 
\] for hat shape (3.16)

\[
\frac{X_1}{X_4} - \frac{60}{1.0} \leq 0 
\] for I and Z shapes (3.17)

and when the bottom flange is in compression:
\[
\frac{X_3}{X_4} - \frac{1.0}{60} \leq 0 \quad \text{for \ hat, \ i, \ and \ Z \ shapes} \tag{3.18}
\]

For web local buckling:

\[
\frac{X_3}{X_4} - \frac{1.0}{200} \leq 0 \quad \text{for \ hat, \ i, \ and \ Z \ shapes} \tag{3.19}
\]

In addition to the above constraints, practical considerations may dictate some upper and lower bounds on the design variables. For example, the thickness of a shape may be limited by the capacity of the forming equipment. Generally, the thickness of commonly used members varies from 0.5 mm to 6.0 mm.

3.3 NEURAL DYNAMICS OPTIMIZATION MODEL

Adeli and Park (1995a) presented a general neural dynamics model for optimization problems that guarantees a stable and local optimum solution. This solved one of the problems of using ANNs for optimization: How to find a neural dynamics system for a particular optimization problem that would produce a stable and local optimum solution. The neural dynamics optimization model is robust and particularly effective for large and complex optimization problems. The model has been applied to the optimum design of large steel structures, including a 144-story super-high-rise building structure with over 20,000 members (Adeli and Park, 1996a & b).

The constrained optimization problem (Eqs. (3.1), (3.2), and (3.3)), can be written in the following reduced form by combining Eqs. (3.2) and (3.3):

Minimize

\[ W = F(X) \tag{3.20} \]
subject to inequality constraints

\[ g^+_j(X) \leq 0 \quad J = 1, J; k = 1, N \_L \]  (3.21)

where \( J \) is the total number of inequality constraints including side constraints. Using the exterior penalty function method, the constrained optimization problem can be written as an unconstrained optimization problem as follows:

\[ V(X, r_n) = F(X) + \frac{r_n}{2} \left\{ \sum_{j=1}^{J} \left[ g^+_j(X) \right]^2 \right\} \]  (3.22)

where \( g^+_j(X) = \max \{0, g_j(X)\} \) and \( r_n \) is the penalty parameter magnifying constraint violations.

A dynamic system is defined as a system of first order differential equations of the state variables:

\[ \frac{dX}{dt} = \dot{X} = f(X) \]  (3.23)

where the vector \( X(t) = \{X_1(t), X_2(t), \ldots, X_N(t)\}^T \) represents the time evolution of node activations. A solution \( \dot{X} \) to the system of equations \( \dot{X} = 0 \) is an equilibrium point of the system. In the theory of nonlinear system dynamics, the Lyapunov stability theorem is used to determine the stability of the equilibrium point. The theorem states that \( \dot{X} \) is stable if the time derivative of a functional, \( V \), called the Lyapunov functional (or energy functional), is negative semi-definite for all non-zero \( X \) i.e. \( \frac{dV}{dt} \leq 0 \). The Lyapunov functional is defined as an analytic function of the system variables, \( V(X) \), such that \( V(0) = 0 \) and \( V(X) > 0 \) for all \( |X| > 0 \) (Kolk and Lerman, 1992).
In the design of cold-formed steel beams, each term $X_i$ in the vector $X$ represents a cross-sectional dimension and is always greater than zero. Therefore, both the objective function and the penalized constraint functions separately satisfy the conditions for a Lyapunov functional. As Eq. (3.22) is made of these two terms it is therefore a valid Lyapunov functional.

Using the chain rule, we have

$$
\frac{dV}{dt} = \left( \frac{dV}{dX} \right) \left( \frac{dX}{dt} \right) = \left[ \nabla F(X) + r_n \left\{ \sum_{j=1}^{J} g_j^* \nabla g_j(X) \right\} \right] \dot{X}
$$

(3.24)

where $\nabla F(X)$ and $\nabla g_j(X)$ are the gradients of the objective function and the $j$th constraint function, respectively. To satisfy the Lyapunov stability theorem, following Adeli and Park (1995a) we define the neural dynamics system as (Adeli and Park, 1995a):

$$
\frac{dX}{dt} = \dot{X} = -\nabla F(X) - r_n \left\{ \sum_{j=1}^{J} g_j^* \nabla g_j(X) \right\}
$$

(3.25)

In this case

$$
\frac{dV}{dt} = \left[ \nabla F(X) + r_n \left\{ \sum_{j=1}^{J} g_j^* \nabla g_j(X) \right\} \right]^2 \leq 0
$$

(3.26)

and the Lyapunov stability theorem is satisfied. This ensures that the system evolves such that the value of the penalized objective function always decreases.

To guarantee that an equilibrium point $\hat{X}$ is an optimal solution of the problem the Kuhn-Tucker optimality conditions are satisfied (Adeli and Park, 1995a):

$$
\frac{\partial L}{\partial X_i} = \frac{\partial F(X)}{\partial X_i} + \sum_{j=1}^{J} \frac{\partial g_j(X)}{\partial X_i} = 0 \; \text{for all } i
$$

(3.27)
\[ g_j(X) + s_j^2 = 0; \quad j = 1, J \tag{3.28} \]

\[ u_j s_j = 0; \quad j = 1, J \tag{3.29} \]

\[ u_j \geq 0; \quad j = 1, J \tag{3.30} \]

where \( L \) is the Lagrangian function defined as a linear combination of the objective and constraint functions:

\[ L(X, u, s) = F(X) + \sum_{j=1}^{J} u_j \left[ g_j(X) + s_j^2 \right], \tag{3.31} \]

in which \( s_j \) is the slack term for the \( j \)th inequality constraint, and \( u_j \) is the Lagrangian multiplier corresponding to the \( j \)th inequality constraint.

The optimum solution can be found by the integration:

\[ X = \int \dot{X} dt \tag{3.32} \]

This integration is done by either the Euler or Runge-Kutta method. For this neural dynamics model the relationship, \( \dot{X} = 0 \), is the learning rule which governs the evolution of the network towards a local optimal solution of the optimization problem.

### 3.4 NEURAL NETWORK MODEL FOR OPTIMIZATION OF COLD-FORMED STEEL BEAMS

The neural network model for optimization of cold-formed steel beams has two major components: (1) An information server composed of modules representing the cold-formed steel beam design problem, and (2) a neural network topology representing the dynamic system.
Figure 15. The components of the neural network model for optimization of cold-formed steel beams
corresponding to the optimization problem. The neural network topology, the modules in the information server, and the interaction between the two are shown in Figure 15.

The purpose of the information server is to obtain data from the user, define the cold-formed steel beam design problem, and to provide the necessary information to the neural dynamics model. The user input is obtained only once in the beginning. Information to the neural dynamics model, on the other hand, is provided at each iteration based on the updated values of the design variables. The information server has three modules that interact with one another. The *Shape* module provides information on the selected shape to the other modules and the neural dynamics model. It also computes the objective function (weight of the beam) and its gradients. The design loads (bending moments and shear) are computed by the *Beam analysis* module. This module is used only once in the beginning as the span, the bracing conditions, and the design loads are constants in the optimization problem. The *AISI Specifications* module contains the provisions of both AISI ASD and LRFD Specifications. At each iteration it provides the coefficients of the constraint functions and their gradients to the neural dynamics model.

In general, the topology of a neural network is represented by a matrix of weighted connections between vectors (or layers) of nodes. The input to a node in a layer is the weighted sum of outputs of nodes in the connected layer(s). The output of a node is obtained by applying an appropriate activation function to the input. The operation of the network is governed by a learning rule that controls the evolution of the connection weights or the node outputs. This learning rule needs to converge to a stable state representing the desired solution.
The topology of our neural dynamics model has one variable layer and \( N_c \) constraint layers. The variable layer has four (4) nodes corresponding to the design variables in the design of cold-formed steel beams, that is, the cross-sectional dimensions identified in Figure 13. The nodes in each constraint layer correspond to the design constraints for a particular loading case. All constraint layers are fully interconnected with the variable layer. Both excitatory (having positive weights) and inhibitory (having negative weights) connections are used to link the nodes. In addition to the commonly used interlayer connections, recurrent connections are also used in the variable layer.

The neural dynamics algorithm for the optimization of cold-formed steel beams follows:

**Step 1:** Set the iteration counter, \( n = 0 \).

**Step 2:** Select an initial decision vector \( \mathbf{X}^0 = X^0 \), the initial penalty parameter \( r_0 \), and the objective function tolerance, \( \delta \), used in the stopping criterion.

**Step 3:** Calculate the gradients of the objective function and assign them to the inhibitory recurrent connections of the variable layer. For the \( i \)th node in the variable layer, the weight of the recurrent connection is:

\[
C_i = -\frac{\partial F(\mathbf{X}^n)}{\partial X_i} \quad i = 1, 4
\]

**Step 4:** Assign the coefficients of the constraint functions to the excitatory connections from the variable layer to the constraint layer(s). The input to the \( j \)th node in the \( k \)th constraint layer is therefore the magnitude of the constraint, \( g_j(\mathbf{X}^n) \).
Step 5: Calculate the output of the nodes in the constraint layer(s). The output of node $j$ in layer $k$ is given by:

$$O_j^k = r_n \max \left[0, g_j^k(X^n) \right] \quad j = 1, J; k = 1, N_L$$  \hspace{1cm} (3.34)

This is the activation function of the node. The output is zero when the constraint is satisfied and equal to the penalized constraint violation otherwise. If there is only one load case skip step 6 and go to step 7.

Step 6: Create a competition between the outputs of the corresponding nodes in the various constraint layers. The governing output of the $j$th node is taken as the maximum value obtained from all the loading cases.

$$O_j = \max \left[O_j^1, O_j^2, O_j^3, \ldots, O_j^{N_L} \right] \quad j = 1, J$$  \hspace{1cm} (3.35)

In this way, only the most violated constraint is included in the constraint set.

Step 7: Assign the gradients of the violated constraint functions to the inhibitory connections from the constraint layer to the variable layer. The weight of the connection from the $j$th constraint node to the $i$th variable node is given by:

$$Z_{ji} = \frac{\partial g_j(X^n)}{\partial X_i} \quad j = 1, J; i = 1, 4$$  \hspace{1cm} (3.36)

All gradients are calculated by the finite difference method.

Step 8: Calculate the input to the variable layer. For the $i$th node in the variable layer, the input is given by:

$$I_{vi} = C_i + \sum_{j=1}^{J} Z_{ji} O_j \quad i = 1, 4$$  \hspace{1cm} (3.37)
\( L_i \) is the direction of the steepest descent of the equivalent unconstrained optimization problem along \( X_i \).

**Step 9:** Update the decision variables using the following learning rule:

\[
X_i^{n+1} = X_i^n + \int I_i \, dt \quad \quad i = 1,4
\]  
(3.38)

The Euler method is used to evaluate the integral.

**Step 10:** Calculate the new value of the objective function, \( F(X^{n+1}) \). If

\[
\left| \frac{F(X^n) - F(X^{n+1})}{F(X^n)} \right| < \delta
\]
(3.39)

stop. The current state vector, \( X^{n+1} \), is the optimal solution of the problem. Otherwise, set \( n = n + 1 \) and update the penalty parameter using the expression (Adeli and Park, 1995a):

\[
r_n = r_o + \frac{n}{\alpha}
\]
(3.40)

where \( \alpha \) is a real positive number. This function is chosen to avoid the possibility of numerical ill-conditioning by gradually increasing the penalty with increasing iterations. Go to step 3.

### 3.5 APPLICATIONS

Three simply-supported example beams are used to test the developed neural network model for optimization of cold-formed steel beams. They include hat, I, and Z-shaped beams designed according to the AISI ASD and the LRFD Specifications subjected to various loading conditions and having different lateral bracing conditions. The following data have been used in all the examples: \( \rho = \) Unit weight of steel = 77 MN/m\(^3\) (0.2836 lb/in\(^3\)); \( F_y = \) Yield stress = 345 N/mm\(^2\) (50.8 ksi); \( E = \) Modulus of elasticity = 203 kN/mm\(^2\) (29.44 \times 10^3 ksi); \( \Delta_n = \)
Allowable maximum deflection = 15 mm (0.79 in.); \( B \) = Length of bearing plate at reactions and locations of concentrated loads = 90 mm (3.54 in.); \( c_f \) = Allowable amount of flange curling = 6 mm (0.24 in.); \( r \) = Inner radius at corners of shape = 8 mm (0.31 in.); \( X_{q}^{U} \) = 8 mm (0.31 in.). No lower limit is placed on the thickness of the shapes. For the stopping criterion, \( \delta = 1 \times 10^{-4} \).

3.5.1 Example 1

In this example a simply-supported beam of span length \( L = 3 \) m is subjected to uniformly distributed dead and live loads of 3 kN/m and 12 kN/m, respectively. Full lateral support is assumed for all the cases. Three types of shapes are designed for this beam: hat, I, and Z shapes. Two sets of initial design variables are used in order to investigate the effects of the initial design on the optimum solution: \( A \{50, 50, 150, 8 \text{ mm}\} \) and \( B \{100, 100, 200, 8 \text{ mm}\} \).

The optimization results are summarized in Table 7 and convergence results are shown in Figure 16. A number of important observations can be made. First, the dominant design variable is the thickness. For the optimum solution it can change drastically from the initial design value. Second, the optimization algorithm yields a local optimum solution in the vicinity of the initial values for the depth and the flange widths. Third, in all the examples the bending strength and the combined bending and shear strength control the optimum design. For the live-to-dead load ratio of 4 used in this example, there is no significant difference in the optimum designs based on the AISI ASD and the LRFD Specifications.
<table>
<thead>
<tr>
<th>Case #</th>
<th>Shape</th>
<th>Specification</th>
<th>$X_1$ mm (in.)</th>
<th>$X_2$ mm (in.)</th>
<th>$X_3$ mm (in.)</th>
<th>$X_4$ mm (in.)</th>
<th>$W$ N (Ib)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Design A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Hat</td>
<td>ASD</td>
<td>50 (2.0)</td>
<td>50 (2.0)</td>
<td>150 (5.9)</td>
<td>8.00 (0.31)</td>
<td>1110.8 (249.7)</td>
</tr>
<tr>
<td>2</td>
<td>Hat</td>
<td>LRFD</td>
<td>54 (2.1)</td>
<td>47 (1.9)</td>
<td>149 (5.9)</td>
<td>4.04 (0.16)</td>
<td>559.9 (125.9)</td>
</tr>
<tr>
<td>3</td>
<td>I</td>
<td>ASD</td>
<td>54 (2.1)</td>
<td>47 (1.9)</td>
<td>149 (5.9)</td>
<td>4.20 (0.17)</td>
<td>582.1 (130.9)</td>
</tr>
<tr>
<td>4</td>
<td>I</td>
<td>LRFD</td>
<td>55 (2.2)</td>
<td>47 (1.9)</td>
<td>151 (5.9)</td>
<td>4.17 (0.16)</td>
<td>582.5 (131.0)</td>
</tr>
<tr>
<td>5</td>
<td>Z</td>
<td>ASD</td>
<td>48 (1.9)</td>
<td>48 (1.9)</td>
<td>153 (6.0)</td>
<td>4.68 (0.18)</td>
<td>377.6 (84.9)</td>
</tr>
<tr>
<td>6</td>
<td>Z</td>
<td>LRFD</td>
<td>49 (1.9)</td>
<td>48 (1.9)</td>
<td>157 (6.2)</td>
<td>4.60 (0.18)</td>
<td>375.6 (84.4)</td>
</tr>
<tr>
<td>Initial Design B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Hat</td>
<td>ASD</td>
<td>100 (3.9)</td>
<td>100 (3.9)</td>
<td>200 (7.9)</td>
<td>8.00 (0.31)</td>
<td>1480.4 (332.8)</td>
</tr>
<tr>
<td>2</td>
<td>Hat</td>
<td>LRFD</td>
<td>96 (3.8)</td>
<td>96 (3.8)</td>
<td>191 (7.5)</td>
<td>2.43 (0.10)</td>
<td>436.3 (98.1)</td>
</tr>
<tr>
<td>3</td>
<td>I</td>
<td>ASD</td>
<td>96 (3.8)</td>
<td>96 (3.8)</td>
<td>193 (7.6)</td>
<td>2.51 (0.10)</td>
<td>449.3 (101.0)</td>
</tr>
<tr>
<td>4</td>
<td>I</td>
<td>LRFD</td>
<td>96 (3.8)</td>
<td>96 (3.8)</td>
<td>194 (7.6)</td>
<td>2.58 (0.10)</td>
<td>465.9 (104.7)</td>
</tr>
<tr>
<td>5</td>
<td>Z</td>
<td>ASD</td>
<td>95 (3.7)</td>
<td>95 (3.7)</td>
<td>195 (7.7)</td>
<td>3.87 (0.15)</td>
<td>433.8 (97.5)</td>
</tr>
<tr>
<td>6</td>
<td>Z</td>
<td>LRFD</td>
<td>95 (3.7)</td>
<td>95 (3.7)</td>
<td>195 (7.7)</td>
<td>3.95 (0.16)</td>
<td>442.6 (99.5)</td>
</tr>
</tbody>
</table>

**Table 7.** Optimum solutions for example 1
Figure 16. Convergence histories for example 1
3.5.2 Example 2

In this example, the beam is subjected to concentrated loads in addition to uniformly distributed load, as shown in Figure 17. Three different lateral support conditions are considered as noted in Table 8. For this example an I shape is optimized using both the AISI ASD and the LRFD Specifications. Again, all cases are solved using two initial sets of design variables: A (70, 70, 200, 8 mm) and B (70, 70, 300, 8 mm).

The optimization results are summarized in Table 8 and the convergence results shown in Figure 18. For cases 1 and 2, the lateral buckling strength of the unbraced segment controlled the optimum design. For cases 3 and 4, the lateral buckling strength of the unbraced segment(s), the bending strength, and the combined bending and shear strength were all active at the optimum design. The optimum designs for cases 5 and 6 were controlled by bending strength and the combined bending and shear strength. The other conclusions regarding the thickness and the local optimum solutions reached for example 1 are also valid for this example.
<table>
<thead>
<tr>
<th>Case #</th>
<th>Lateral bracing</th>
<th>Specification</th>
<th>$X_1$ mm (in.)</th>
<th>$X_2$ mm (in.)</th>
<th>$X_3$ mm (in.)</th>
<th>$X_4$ mm (in.)</th>
<th>$W$ N (lb)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Design A</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>None</td>
<td>ASD</td>
<td>70 (2.8)</td>
<td>70 (2.8)</td>
<td>200 (7.9)</td>
<td>8.00 (0.31)</td>
<td>1597.7 (359.2)</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
<td>LRFD</td>
<td>160 (6.3)</td>
<td>69 (2.7)</td>
<td>232 (9.1)</td>
<td>4.71 (0.19)</td>
<td>1137.5 (255.7)</td>
</tr>
<tr>
<td>3</td>
<td>Midspan</td>
<td>ASD</td>
<td>159 (6.3)</td>
<td>69 (2.7)</td>
<td>231 (9.1)</td>
<td>4.71 (0.19)</td>
<td>1132.8 (254.7)</td>
</tr>
<tr>
<td>4</td>
<td>Midspan</td>
<td>LRFD</td>
<td>100 (3.9)</td>
<td>77 (3.0)</td>
<td>222 (8.7)</td>
<td>4.82 (0.19)</td>
<td>1067.5 (240.0)</td>
</tr>
<tr>
<td>5</td>
<td>Full</td>
<td>ASD</td>
<td>100 (3.9)</td>
<td>77 (3.0)</td>
<td>222 (8.7)</td>
<td>4.78 (0.19)</td>
<td>1059.3 (238.2)</td>
</tr>
<tr>
<td>6</td>
<td>Full</td>
<td>LRFD</td>
<td>92 (3.6)</td>
<td>76 (3.0)</td>
<td>226 (8.9)</td>
<td>4.78 (0.19)</td>
<td>1057.2 (237.7)</td>
</tr>
<tr>
<td><strong>Initial Design B</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>None</td>
<td>ASD</td>
<td>70 (2.8)</td>
<td>70 (2.8)</td>
<td>300 (11.8)</td>
<td>8.00 (0.31)</td>
<td>2028.9 (456.1)</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
<td>LRFD</td>
<td>149 (5.9)</td>
<td>66 (2.6)</td>
<td>308 (12.1)</td>
<td>3.84 (0.15)</td>
<td>1067.2 (240.0)</td>
</tr>
<tr>
<td>3</td>
<td>Midspan</td>
<td>ASD</td>
<td>148 (5.8)</td>
<td>66 (2.6)</td>
<td>307 (12.1)</td>
<td>3.82 (0.15)</td>
<td>1061.4 (238.6)</td>
</tr>
<tr>
<td>4</td>
<td>Midspan</td>
<td>LRFD</td>
<td>90 (3.5)</td>
<td>68 (2.7)</td>
<td>301 (11.9)</td>
<td>3.74 (0.15)</td>
<td>969.3 (217.9)</td>
</tr>
<tr>
<td>5</td>
<td>Full</td>
<td>ASD</td>
<td>90 (3.5)</td>
<td>68 (2.7)</td>
<td>301 (11.9)</td>
<td>3.71 (0.15)</td>
<td>962.4 (216.4)</td>
</tr>
<tr>
<td>6</td>
<td>Full</td>
<td>LRFD</td>
<td>70 (2.8)</td>
<td>70 (2.8)</td>
<td>299 (11.8)</td>
<td>3.41 (0.13)</td>
<td>864.3 (194.3)</td>
</tr>
</tbody>
</table>

Table 8. Optimum solutions for example 2
Figure 18. Convergence histories for example 2
2 kN/m (dead)  
5 kN/m (live)  
5 kN/m (wind)  
5 kN/m (snow)

Figure 19. Example 3

3.5.3 Example 3

In this example a 4 m long Z-shaped roof purlin is designed (Figure 19). The loading consists of uniformly distributed dead, live, snow, and wind loads. Different bracing conditions are considered as indicated in Table 9. All cases are solved starting from two sets of initial design variables: A{100, 100, 200, 8 mm} and B{70, 70, 300, 8 mm}.

The optimum solutions and the convergence results are given in Table 9 and Figure 20. For cases 1 and 2, the lateral buckling strength of the unbraced segment controlled the optimum design. For cases 3A to 6A, all three of the following constraints were active at the optimum design: lateral buckling strength of the unbraced segments, the bending strength, and the combined bending and shear strength. The optimum design of cases 7A and 8A were controlled by the bending and combined bending and shear strengths. Providing lateral bracing at the midspan reduced weight significantly as compared to the unbraced beam. However, additional lateral bracing had no significant effect on the optimum designs. The web crippling strength at the reactions controlled the optimum design of cases 3B to 8B. As a result, for these cases the lateral bracing condition had no effect on the optimum design. The same
<table>
<thead>
<tr>
<th>Case #</th>
<th>Lateral bracing</th>
<th>Specification</th>
<th>$X_1$ (mm)</th>
<th>$X_2$ (mm)</th>
<th>$X_3$ (mm)</th>
<th>$X_4$ (mm)</th>
<th>$W$ (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Design A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>None</td>
<td>ASD</td>
<td>100 (3.9)</td>
<td>100 (3.9)</td>
<td>200 (7.9)</td>
<td>8.00 (0.31)</td>
<td>1233.3 (277.3)</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
<td>LRFD</td>
<td>131 (5.2)</td>
<td>83 (3.3)</td>
<td>213 (8.4)</td>
<td>4.37 (0.17)</td>
<td>683.4 (153.6)</td>
</tr>
<tr>
<td>3</td>
<td>Midspan</td>
<td>ASD</td>
<td>96 (3.8)</td>
<td>93 (3.7)</td>
<td>200 (7.9)</td>
<td>4.04 (0.16)</td>
<td>710.9 (159.8)</td>
</tr>
<tr>
<td>4</td>
<td>Midspan</td>
<td>LRFD</td>
<td>97 (3.8)</td>
<td>91 (3.6)</td>
<td>204 (8.0)</td>
<td>4.15 (0.16)</td>
<td>609.4 (137.0)</td>
</tr>
<tr>
<td>5</td>
<td>QP, midspan</td>
<td>ASD</td>
<td>95 (3.7)</td>
<td>95 (3.7)</td>
<td>195 (7.7)</td>
<td>4.05 (0.16)</td>
<td>630.4 (141.7)</td>
</tr>
<tr>
<td>6</td>
<td>QP, midspan</td>
<td>LRFD</td>
<td>93 (3.7)</td>
<td>94 (3.7)</td>
<td>199 (7.8)</td>
<td>4.18 (0.16)</td>
<td>605.8 (136.2)</td>
</tr>
<tr>
<td>7</td>
<td>Full</td>
<td>ASD</td>
<td>95 (3.7)</td>
<td>95 (3.7)</td>
<td>196 (7.7)</td>
<td>4.04 (0.16)</td>
<td>626.6 (140.9)</td>
</tr>
<tr>
<td>8</td>
<td>Full</td>
<td>LRFD</td>
<td>91 (3.6)</td>
<td>92 (3.6)</td>
<td>203 (8.0)</td>
<td>4.12 (0.16)</td>
<td>604.7 (135.9)</td>
</tr>
<tr>
<td>Initial Design B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>None</td>
<td>ASD</td>
<td>70 (2.8)</td>
<td>70 (2.8)</td>
<td>300 (11.8)</td>
<td>8.00 (0.31)</td>
<td>1331.9 (299.4)</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
<td>LRFD</td>
<td>114 (4.5)</td>
<td>58 (2.3)</td>
<td>297 (11.7)</td>
<td>3.50 (0.14)</td>
<td>613.9 (138.0)</td>
</tr>
<tr>
<td>3</td>
<td>Midspan</td>
<td>ASD</td>
<td>111 (4.4)</td>
<td>65 (2.6)</td>
<td>298 (11.7)</td>
<td>3.72 (0.15)</td>
<td>658.4 (148.0)</td>
</tr>
<tr>
<td>4</td>
<td>Midspan</td>
<td>LRFD</td>
<td>64 (2.5)</td>
<td>64 (2.5)</td>
<td>294 (11.6)</td>
<td>3.50 (0.14)</td>
<td>563.9 (126.8)</td>
</tr>
<tr>
<td>5</td>
<td>QP, midspan</td>
<td>ASD</td>
<td>64 (2.5)</td>
<td>64 (2.5)</td>
<td>294 (11.6)</td>
<td>3.72 (0.15)</td>
<td>598.5 (134.6)</td>
</tr>
<tr>
<td>6</td>
<td>QP, midspan</td>
<td>LRFD</td>
<td>64 (2.5)</td>
<td>64 (2.5)</td>
<td>294 (11.6)</td>
<td>3.50 (0.14)</td>
<td>563.9 (126.8)</td>
</tr>
<tr>
<td>7</td>
<td>Full</td>
<td>ASD</td>
<td>64 (2.5)</td>
<td>64 (2.5)</td>
<td>294 (11.6)</td>
<td>3.72 (0.15)</td>
<td>598.5 (134.6)</td>
</tr>
<tr>
<td>8</td>
<td>Full</td>
<td>LRFD</td>
<td>64 (2.5)</td>
<td>64 (2.5)</td>
<td>294 (11.6)</td>
<td>3.50 (0.14)</td>
<td>563.9 (126.8)</td>
</tr>
</tbody>
</table>

QP = Quarter points.

**Table 9.** Optimum solutions for example 3
Figure 20. Convergence histories for example 3
conclusions are reached regarding the thickness and the convergence to a local optimum solution as for examples 1 and 2.

3.6 CONCLUDING REMARKS

In order to take advantage of the flexibility of cold-formed steel design, it is essential to have an efficient computational model to automate the design process. The design of cold-formed steel is governed by complex and highly nonlinear rules that make efficient optimization difficult. In this article, we presented a robust neural network model for optimum design of cold-formed steel beams and applied it to three commonly used shapes (hat, I, and Z) according to the AISI ASD and the LRFD Specifications. Any combination of uniformly distributed and/or concentrated loads can be modeled. The beam may be fully braced, unbraced, or braced at a specified number of points.

The computational model has been implemented in an object-oriented programming (OOP) language, C++ on an IBM RISC 6000 workstation. The choice of an OOP language as the programming environment was based on its desirable characteristics of abstraction, inheritance, modularity, and encapsulation of data and operations (Adeli and Yu, 1993; Yu and Adeli, 1993; Adeli and Kao, 1996).

The convergence curves obtained for a large number of examples including those presented in Figures 16, 18, and 20 demonstrate the robustness of the computational model and algorithm. Further, the computational model is efficient in terms of computer processing time. The optimum results for various examples presented in this article are obtained on an IBM RISC 6000 in less than 6.5 seconds.
LIST OF REFERENCES


AISI (1989), *Specification for the Design of Cold-Formed Steel Structural Members*, American Iron and Steel Institute, Washington, DC.

AISI (1991), *Specification for the Load and Resistance Factor Design of Cold-Formed Steel Structural Members*, American Iron and Steel Institute, Washington, DC.


APPENDIX A

COMPUTER CODE LISTING FOR CHAPTER 2

A.1 optimize.C

#include "nd.H"
#include "schedl.H"

main()
{
    NeuralDynamics<Schedule> opt;
    opt.optimize();
}

A.2 schedl.H

#ifndef _SCHEDL_H_
define _SCHEDL_H_

// class defining the scheduling/cost optimization problem
class Schedule
{
    long double* x;        // no. of variables (starting times; durations)
    long double* gg;       // g in NeuralDynamics
    long double* gg_plus_h; // g_plus_h in NeuralDynamics
    long double* hh;       // h in ND
    long double* hh_plus_h; // h_plus_h in ND

    // variables defining data for the illustrative example
    // t9 = task 9. First index = segment number-1. second index = type of data
    long double t9[4][4];


74
long double t1[3][4];
long double t1c1[2][4];
long double t1c2[2][4];
long double t1c3[2][4];
long double t1c4[1][4];

public:
  int no_of_variables;
  int no_of_inequality_constraints;
  int no_of_equality_constraints;

  long double DMAX; // max. duration of project

  long double objective_function();
  long double* inequality_constraints (int plus);
  long double* equality_constraints (int plus);

  long double* current_var_state();
  void change_variables (long double* _xx);
  long double* side_constraints (long double* _xx);

  Schedule();
};

#endif

A.3 schedl.C

#include "absolute.H"
#include "frac.H"
#include "schedl.H"
#include <iostream.h>

// for the illustrative example
#define NOV 47     // no. of var
#define NOIC 45    // no. of inequality constraints
#define NOEC 31    // no. of equality constraints

// initialize Schedule

75
Schedule::Schedule()
{
    no_of_variables = NOV;
    no_of_inequality_constraints = NOIC;
    no_of_equality_constraints = NOEC;

    x = new long double[NOV];
    gg = new long double[NOIC];
    gg_plus_h = new long double[NOIC];
    hh = new long double[NOEC];
    hh_plus_h = new long double[NOEC];

    cout << "Input DMAX: ";
    cin >> DMAX;

    // initial decision variables
    for (int i = 0; i < 17; i++)
        x[i] = 0.2;
    x[17] = 20;
    int k;
    for (i = 18, k = 1; i < NOV; i++, k++)
        x[i] = 20.0 + 3*k;

    // initialize data
    t9[0][0] = 1200;
    t9[0][1] = 3000;
    t9[0][2] = 4.5;
    t9[0][3] = 1;
    t9[1][0] = 3000;
    t9[1][1] = 3500;
    t9[1][2] = 1.25;
    t9[1][3] = 1.1;
    t9[2][0] = 3500;
    t9[2][1] = 3650;
    t9[2][2] = 1.875;
    t9[2][3] = 1.1;
    t9[3][0] = 3650;
    t9[3][1] = 6000;
    t9[3][2] = 5.875;
    t9[3][3] = 1.15;

    t10c1[0][0] = 1000;
    t10c1[0][1] = 3000;
    t10c1[0][2] = 10.0;
t10c1[0][3] = 1.0;
t10c1[1][0] = 3000;
t10c1[1][1] = 3500;
t10c1[1][2] = 6.0;
t10c1[1][3] = 1.15;
t10c2[0][0] = 3500;
t10c2[0][1] = 5000;
t10c2[0][2] = 8.0;
t10c2[0][3] = 1.05;
t10c2[1][0] = 5000;
t10c2[1][1] = 6000;
t10c2[1][2] = 5.0;
t10c2[1][3] = 1.0;

t11[0][0] = 1000;
t11[0][1] = 2000;
t11[0][2] = 4.25;
t11[0][3] = 1;
t11[1][0] = 2000;
t11[1][1] = 4000;
t11[1][2] = 8.5;
t11[1][3] = 1.05;
t11[2][0] = 4000;
t11[2][1] = 6000;
t11[2][2] = 8.5;
t11[2][3] = 1;

t12c1[0][0] = 3500;
t12c1[0][1] = 2250;
t12c1[0][2] = 2.6;
t12c1[0][3] = 1;
t12c1[1][0] = 2250;
t12c1[1][1] = 1000;
t12c1[1][2] = 2.6;
t12c1[1][3] = 1.05;

t12c2[0][0] = 3500;
t12c2[0][1] = 4750;
t12c2[0][2] = 2.6;
t12c2[0][3] = 1;
t12c2[1][0] = 4750;
t12c2[1][1] = 6000;
t12c2[1][2] = 2.6;
t12c2[1][3] = 1.05;
t13c2[0][0] = 3500;
t13c2[0][1] = 4750;
t13c2[0][2] = 1.25;
t13c2[0][3] = 1.0;
t13c2[1][0] = 4750;
t13c2[1][1] = 6000;
t13c2[1][2] = 1.25;
t13c2[1][3] = 1.05;
t13c1[0][0] = 3500;
t13c1[0][1] = 2250;
t13c1[0][2] = 1.25;
t13c1[0][3] = 1.0;
t13c1[1][0] = 2250;
t13c1[1][1] = 1000;
t13c1[1][2] = 1.25;
t13c1[1][3] = 1.05;


t14[0][0] = 1000;
t14[0][1] = 6000;
t14[0][2] = 3;
t14[0][3] = 1;

}

// direct cost of project
long double Schedule::objective_function ()
{
    return (3*(-300*x[0]+1050)+3*(-280*x[1]+960)+5*(-200*x[2]+250)+
    3*(1600/x[7])+13.5*(-300*x[8]+1050)+16.0*(1600+500*x[9])/x[9]+
    3*(-100*x[16]+800));
}

// scheduling constraints-equality
// when more than one constraint is specified the most violated is taken as
// the governing constraint
long double* Schedule::equality_constraints (int plus)
{

78
long double* a;
if (plus == 0)
{
    a = hh;
}
else
{
    a = hh_plus_h;
}

long double ct, ct1;

// task 2
a[0] = (x[17]+(1.0*3.0*x[0]))/x[18]-1.0;

// task 3
c = x[18]/x[19]-1.0;
c1 = (x[20]+(1.0*3.0*x[3]))/x[19]-1.0;
if (ct1 > ct) ct = ct1;
a[1] = ct;

// task 4
a[2] = (x[18]+(1.0*3.0*x[1]*0.25))/x[20]-1.0;

// task 5
a[3] = (x[19]+(1.0*5.0*x[2]))/x[21]-1.0;

// task 6
a[4] = (x[29]+(1.1*1.875*x[8]))/x[22] - 1.0;

// task 7
c = (x[22]+(1-frac(x[22])))/x[23] - 1.0;
c1 = (x[22]+(1.0*1.5*x[5]))/x[23]-1.0;
if (ct1 > ct) ct = ct1;
a[5] = ct;

// task 8
a[7] = (x[25]+1.0*1.0*x[7]+1-frac(x[25]+1.0*1.0*x[7]))/x[26]-1.0;
ct = (x[27]+1.0*100*4.5*x[8]/1800)/x[24]-1.0;
c1 = (x[27]+1.0*1550*4.5*x[8]/1800)/x[25]-1.0;
if (ct1 > ct) ct = ct1;
ct1 = (x[30]+1.15*5.875*1850*x[8]/2350)/x[26]-1.0;
if (ct1 > ct) ct = ct1;

a[8] = ct;

// task 9
a[9] = (x[27]+(1.0*4.5*x[8]))/x[28]-1.0;
a[10] = (x[28]+(1.1*1.25*x[8]))/x[29]-1.0;
a[11] = (x[29]+(1.1*1.875*x[8]))/x[30]-1.0;

a[12] = (x[17]+1.0*3.0*x[0])/x[27]-1.0;

// task 10
a[13] = (x[31]+1.0*10.0*x[9])/x[32]-1.0;
a[14] = (x[33]+1.05*8.0*x[10])/x[34]-1.0;

a[15] = x[31]/x[33]-1.0;

ct = (x[21]+1.0*1.0*x[4]+1-frac(x[21]+1.0*1.0*x[4]))/x[31] - 1.0;

// buffers
long double buff = 150.0;
long double xpd;
int i = 0, j = 0;

for (i = 0; i < 4; i++) // no of segments of task 9 (continuous)
{
    xpd = x[27+i];
    for (j = 0; j < 2; j++)
    {
        if (xpd < x[31+j] || xpd > (x[31+j]+t10c1[j][2]*t10c1[j][3]*x[9]))
        {
            continue;
        }
    }
else
    {
        ct1 = (t10c1[j][0]+((xpd-x[31+j])*(t10c1[j][1]-t10c1[j][0]))
            /(x[31+j]+t10c1[j][2]*t10c1[j][3]*x[9]-x[31+j]) + buff)/t9[i][9]-1;
        if (ct1 > ct) ct = ct1;
        break;
    }
}
for (i = 0; i < 4; i++) // no of segments of task 9 (continuous)
{
    xpd = x[27+i]+t9[i][2]*t9[i][3]*x[8];
    for (j = 0; j < 2; j++)
    {
        if (xpd < x[31+j] || xpd > (x[31+j]+t10c1[j][2]*t10c1[j][3]*x[9]))
        {
            continue;
        }
    }
}

for (i = 0; i < 4; i++) // no of segments of task 9 (continuous)
{
    xpd = x[27+i];
    for (j = 0; j < 2; j++)
    {
        if (xpd < x[33+j] || xpd > (x[33+j]+t10c2[j][2]*t10c2[j][3]*x[10]))
        {
            continue;
        }
    }
}

for (i = 0; i < 4; i++) // no of segments of task 9 (continuous)
{
    xpd = x[27+i]+t9[i][2]*t9[i][3]*x[8];
}
for (j = 0; j < 2; j++)
{
    if (xpds < x[33+j] || xpds > (x[33+j]+t10c2[j][2]*t10c2[j][3]*x[10]))
    {
        continue;
    }
else
    {
        ct1 = (t10c2[j][0]+((xpds-x[33+j])*(t10c2[j][1]-t10c2[j][0]))
            /(x[33+j]+t10c2[j][2]*t10c2[j][3]*x[10]-x[33+j]) + buffy/t9[i][1]-1;
        if (ct1 > ct) ct = ct1;
        break;
    }
}

for (i = 0; i < 2; i++)
{
    xpds = x[31+i];
    for (j = 0; j < 4; j++)
    {
        if (xpds < x[27+j] || xpds > (x[27+j]+t9[j][2]*t9[j][3]*x[8]))
        {
            continue;
        }
    }
else
    {
        ct1 = (t10c1[i][0]+buff)/t9[j][0]+((xpds-x[27+j])*(t9[j][1]-t9[j][0]))
            /(x[27+j]+t9[j][2]*t9[j][3]*x[8]-x[27+j])-1;
        if (ct1 > ct) ct = ct1;
        break;
    }
}

for (i = 0; i < 2; i++)
{
    xpds = x[31+i]+t10c1[i][2]*t10c1[i][3]*x[9];
    for (j = 0; j < 4; j++)
    {
        if (xpds < x[27+j] || xpds > (x[27+j]+t9[j][2]*t9[j][3]*x[8]))
        {
            continue;
        }
    }
else
{
    ct1 = (t10c2[i][1]+buff)/(t9[j][0]+((xpd-x[27+j])*(t9[j][1]-t9[j][0]))
     /(x[27+j]+t9[j][2]*t9[j][3]*x[8]-x[27+j])-1;
    if (ct1 > ct) ct = ct1;
    break;
}
}

for (i = 0; i < 2; i++)
{
    xpd = x[33+i];
    for (j = 0; j < 4; j++)
    {
        if (xpd < x[27+j] || xpd > (x[27+j]+t9[j][2]*t9[j][3]*x[8]))
        {
            continue;
        }
    }
else
    {
        ct1 = (t10c2[i][0]+buff)/(t9[j][0]+((xpd-x[27+j])*(t9[j][1]-t9[j][0]))
         /(x[27+j]+t9[j][2]*t9[j][3]*x[8]-x[27+j])-1;
        if (ct1 > ct) ct = ct1;
        break;
    }
}

for (i = 0; i < 2; i++)
{
    xpd = x[33+i]+t10c2[i][2]*t10c2[i][3]*x[10];
    for (j = 0; j < 4; j++)
    {
        if (xpd < x[27+j] || xpd > (x[27+j]+t9[j][2]*t9[j][3]*x[8]))
        {
            continue;
        }
    }
else
    {
        ct1 = (t10c2[i][1]+buff)/(t9[j][0]+((xpd-x[27+j])*(t9[j][1]-t9[j][0]))
         /(x[27+j]+t9[j][2]*t9[j][3]*x[8]-x[27+j])-1;
        if (ct1 > ct) ct = ct1;
        break;
    }

83
a[16] = ct;

// task 11 sub-base
a[18] = (x[36]+8.5*1.05*x[11])/x[37] - 1;

// buffers
ct = (x[31]+3)/x[35]-1.0;

for (i = 0; i < 2; i++)
{
    xpdi = x[31+i];
    for (j = 0; j < 3; j++)
    {
        if (xpdi < x[35+j] || xpdi > (x[35+j]+t11[j][2]*t11[j][3]*x[11]))
        {
            continue;
        }
    }
}

for (i = 0; i < 2; i++)
{
    xpdi = x[31+i]+t10c1[i][2]*t10c1[i][3]*x[9];
    for (j = 0; j < 3; j++)
    {
        if (xpdi < x[35+j] || xpdi > (x[35+j]+t11[j][2]*t11[j][3]*x[11]))
        {
            continue;
        }
    }
}

}
ct1 = (t11[j][0]+((xpd-x[35+j])*(t11[j][1]-t11[j][0]))
/(x[35+j]+t11[j][2]*t11[j][3]*x[11]-x[35+j])+buff)/t10c1[i][1]-1;
if (ct1 > ct) ct = ct1;
break;
}
}
}

for (i = 0; i < 2; i++)
{
  xpd = x[33+i];
  for (j = 0; j < 3; j++)
  {
    if (xpd < x[35+j] || xpd > (x[35+j]+t11[j][2]*t11[j][3]*x[11]))
    {
      continue;
    }
  }
else
  {
    ct1 = (t11[j][0]+((xpd-x[35+j])*(t11[j][1]-t11[j][0]))
/(x[35+j]+t11[j][2]*t11[j][3]*x[11]-x[35+j])+buff)/t10c2[i][0]-1;
    if (ct1 > ct) ct = ct1;
    break;
  }
}
}

for (i = 0; i < 2; i++)
{
  xpd = x[33+i]+t10c2[i][2]*t10c2[i][3]*x[10];
  for (j = 0; j < 3; j++)
  {
    if (xpd < x[35+j] || xpd > (x[35+j]+t11[j][2]*t11[j][3]*x[11]))
    {
      continue;
    }
  }
else
  {
    ct1 = (t11[j][0]+((xpd-x[35+j])*(t11[j][1]-t11[j][0]))
/(x[35+j]+t11[j][2]*t11[j][3]*x[11]-x[35+j])+buff)/t10c2[i][1]-1;
    if (ct1 > ct) ct = ct1;
    break;
  }
}
for (i = 0; i < 3; i++)
{
    xpd = x[35+i];
for (j = 0; j < 2; j++)
{
    if (xpd < x[31+j] || xpd > (x[31+j]+t10c1[j][2]*t10c1[j][3]*x[9]))
    {
        continue;
    }
else
    {
        ct1 = (t11[i][0]+buff)/(t10c1[j][0]+((xpd-x[31+j])*(t10c1[j][1]-t10c1[j][0]))
        /(x[31+j]+t10c1[j][2]*t10c1[j][3]*x[9]-x[31+j]))-1;
        if (ct1 > ct) ct = ct1;
        break;
    }
}
}
for (i = 0; i < 3; i++)
{
    xpd = x[35+i]+t11[i][2]*t11[i][3]*x[11];
for (j = 0; j < 2; j++)
{
    if (xpd < x[31+j] || xpd > (x[31+j]+t10c1[j][2]*t10c1[j][3]*x[9]))
    {
        continue;
    }
else
    {
        ct1 = (t11[i][1]+buff)/(t10c1[j][0]+((xpd-x[31+j])*(t10c1[j][1]-t10c1[j][0]))
        /(x[31+j]+t10c1[j][2]*t10c1[j][3]*x[9]-x[31+j]))-1;
        if (ct1 > ct) ct = ct1;
        break;
    }
}
for (i = 0; i < 3; i++)
{
    xpd = x[35+i];
}
for (j = 0; j < 2; j++)
{
    if (xpd < x[33+j] || xpd > (x[33+j]+t10c2[j][2]*t10c2[j][3]*x[10]))
    {
        continue;
    }
else
    {
        ct1 = (t11[i][j]+buff)/(t10c2[j][j][0]+((xpd-x[33+j])*(t10c2[j][1]-t10c2[j][0]))
           /(x[33+j]+t10c2[j][2]*t10c2[j][3]*x[10]-x[33+j]))-1;
        if (ct1 > ct) ct = ct1;
        break;
    }
}

for (i = 0; i < 3; i++)
{
    xpd = x[35+i]+t11[i][2]*t11[i][3]*x[11];
    for (j = 0; j < 2; j++)
    {
        if (xpd < x[33+j] || xpd > (x[33+j]+t10c2[j][2]*t10c2[j][3]*x[10]))
        {
            continue;
        }
else
        {
            ct1 = (t11[i][j]+buff)/(t10c2[j][j][0]+((xpd-x[33+j])*(t10c2[j][1]-t10c2[j][0]))
               /(x[33+j]+t10c2[j][2]*t10c2[j][3]*x[10]-x[33+j]))-1;
            if (ct1 > ct) ct = ct1;
            break;
        }
    }
}

a[19] = ct;

// task 12
a[20] = 3*((x[38]+(2.6*1.0*x[12]))/x[39]-1.0);
a[21] = 3*((x[40]+(2.6*1.0*x[13]))/x[41]-1.0);
a[22] = 2*((x[39]+2.6*1.05*x[12])/(x[41]+2.6*1.05*x[13])-1.0);

ct = (x[23]+(1.0*1.0*x[6]))/x[38]-1.0;
ctl = (x[36]+1500/2000*8.5*1.05*x[11]+2)/x[40]-1.0;
if (ctl > ct) ct = ctl;
ctl = (x[36]+1500/2000*8.5*1.05*x[11]+2)/x[38]-1.0;
if (ctl > ct) ct = ctl;
ctl = (x[37]+2)/(x[40]+500/1250*2.6*1.0*x[13])-1.0;
if (ctl > ct) ct = ctl;
ctl = (x[37]+750/2000*8.5*1.0*x[11]+2)/x[41]-1.0;
if (ctl > ct) ct = ctl;
ctl = (x[37]+8.5*1.0*x[11]+2)/(x[41]+2.6*1.05*x[13])-1.0;
if (ctl > ct) ct = ctl;
a[23] = 2*ct;

// task 13
a[24] = 2*((x[42]+(1.25*1.0*x[14]))/x[43]-1.0);
a[25] = 2*((x[44]+(1.25*1.0*x[15]))/x[45]-1.0);
a[26] = 3*(x[42]/x[44]-1.0);

cr = (x[40]+2)/x[42]-1.0;
ctl = (x[40]+2)/x[44]-1.0;
if (ctl > ct) ct = ctl;
ctl = (x[41]+2)/x[45]-1.0;
if (ctl > ct) ct = ctl;
ctl = (x[41]+1.05*2.6*x[13]+2)/(x[45]+1.05*1.25*x[15])-1.0;
if (ctl > ct) ct = ctl;
ctl = (x[39]+2)/x[43]-1.0;
if (ctl > ct) ct = ctl;
ctl = (x[39]+2.6*1.05*x[12]+2)/(x[43]+1.05*1.25*x[14])-1.0;
if (ctl > ct) ct = ctl;
a[27] = 3*ct;

// task 14
ct = (x[43]+1.05*1.25*x[14]+2)/x[46]-1.0;
ctl = (x[45]+1.05*1.25*x[15]+2)/(x[46]+3*1*x[16])-1.0;
if (ctl > ct) ct = ctl;
a[28] = 5*ct;

cr = (x[46]+3.0*1.0*x[16])/DMAX-1.0;
a[29] = 2*ct;
a[30] = 2*(x[17]/20-1.0);
return a;
}

// inequality constraints
long double* Schedule::inequality_constraints (int plus)
{
    long double* a;
    if (plus == 0)
    {
        a = gg;
    }
    else
    {
        a = gg_plus_h;
    }

    // a[0] = -0.0005;
    a[0] = 1-x[0]/1.0;
    a[1] = x[0]/1.5-1.0;
    a[2] = 1-x[1]/0.5;
    a[3] = x[1]/2.0-1.0;
    a[4] = 1-x[2]/0.25;
    a[5] = x[2]/0.5-1.0;
    a[6] = 1-x[3]/0.5;
    a[7] = x[3]/1.25-1.0;
    a[8] = 1-x[4]/1.0;
    a[9] = x[4]/2.0-1.0;
    a[10] = 1-x[5]/0.5;
    a[11] = x[5]/2.0-1.0;
    a[12] = 1-x[6]/5.0;
    a[13] = x[6]/8-1.0;
    a[14] = 1-x[7]/2.0;
    a[15] = x[7]/3.0-1.0;
    a[16] = 1-x[8]/1.0;
\[ a[17] = x[8]/1.5-1.0; \]
\[ a[18] = 1-x[9]/1.0; \]
\[ a[19] = x[9]/2.0-1.0; \]
\[ a[20] = 1-x[10]/1.0; \]
\[ a[21] = x[10]/2.0-1.0; \]
\[ a[22] = 1-x[11]/0.75; \]
\[ a[23] = x[11]/1.25-1.0; \]
\[ a[24] = 1-x[12]/0.75; \]
\[ a[25] = x[12]/1.25-1.0; \]
\[ a[26] = 1-x[13]/0.75; \]
\[ a[27] = x[13]/1.25-1.0; \]
\[ a[28] = 1-x[14]/0.75; \]
\[ a[29] = x[14]/1.25-1.0; \]
\[ a[30] = 1-x[15]/0.75; \]
\[ a[31] = x[15]/1.25-1.0; \]
\[ a[32] = 1-x[16]/2.0; \]
\[ a[33] = x[16]/4.0-1.0; \]
\[ a[34] = 2*((x[45]+1.25*1.05*x[15])/DMAX-1.0); \]
\[ a[35] = 2*((x[44]+1.25*1.0*x[15])/DMAX-1.0); \]
\[ a[36] = 2*((x[43]+1.25*1.05*x[14])/DMAX-1.0); \]
\[ a[37] = 2*((x[42]+1.25*1.0*x[14])/DMAX-1.0); \]
\[ a[38] = 2*((x[41]+2.6*1.05*x[13])/DMAX-1.0); \]
\[ a[39] = 2*((x[39]+2.6*1.05*x[12])/DMAX-1.0); \]
\[ a[40] = 2*((x[37]+8.5*x[11])/DMAX-1.0); \]
\[ a[41] = 2*((x[30]+5.875*1.15*x[8])/DMAX-1.0); \]
\[ a[42] = 2*((x[26]+x[7])/DMAX-1.0); \]
\[ a[43] = 2*((x[34]+5*x[10])/DMAX-1.0); \]
\[ a[44] = 2*((x[32]+6*1.15*x[9])/DMAX-1.0); \]

return a;
};

long double* Schedule::current_var_state()
{

90
return x;
}

void Schedule::change_variables (long double* _xx)
{
    for (int i = 0; i < NOV; i++)
    {
        x[i] = _xx[i];
    }
}

long double* Schedule::side_constraints (long double* _xx)
{
    // the start of the first segment of work is set as some constant (20)

    for (int i = 18; i < NOV; i++)
    {
        if (_xx[i] < 20) _xx[i] = 20;
        if (_xx[i] > 150) _xx[i] = 150;
    }

    for (i = 0; i < 17; i++)
    {
        if (_xx[i] < 0.2) _xx[i] = 0.2;
    }

    return _xx;
}

A.4  nd.H

#ifndef _NDOS_H_
#define _NDOS_H_

#include "sec_opt.H"

#define OBJECTIVE_TOLERANCE 0.00001
#define CONSTRAINT_TOLERANCE 0.00005
#define OT 0.0001
#define CT 0.0001
#define STEP 0.0001 // finite difference for gradient calculation

// defines an appropriate neural dynamics optimization model for
// class domain defining the optimization problem
template<class domain>
class NeuralDynamics
{
    // parameters used in the neural dynamics model
    int ITERATIONS; // iterations at constant \( r_n \)
    int Ni; // counter for changing \( r_n \) (penalty parameter)
    long double L_PENALTY; // initial penalty
    long double ALPHA; // const. number alpha
    long double TIME_STEP; // time step for Euler integration

domain problem; // class defining optimization problem
int P; // no. of variables
int Q; // no. of inequality constraints
int R; // no. of equality constraints

long double* delta_x;

long double f; // objective function
long double fn; // normalized objective function
long double* g; // inequality constraints
long double* gp; // violated inequality constraints
long double* g_plus_h; // used for gradient calculations

long double* h; // equality constraints
long double* hp; // violated equality constraints
long double* h_plus_h; // used for gradient calculation

int inequality_constraints_tolerance ();
int equality_constraints_tolerance ();
void inequality_constraints_violation ();
void equality_constraints_violation ();
long double variable_node_input (const int& var_no,
      const long double& _rn);

int simulate_network (const long double& _rn);
public:
    NeuralDynamics();
    int optimize();
};

#include "nd.C"
#endif

A.5 nd.C

#ifndef _ND_C_
#define _ND_C_

#include "absolute.H"
#include <fstream.h>
#include <stdlib.h>

#include "nd.H"

// initialize the network
template <class domain>
NeuralDynamics<domain>::NeuralDynamics()
{
    cout << "Input ITERATIONS: ";
    cin >> ITERATIONS;
    cout << "Input N: ";
    cin >> N;
    cout << "Input I_PENALTY: ";
    cin >> I_PENALTY;
    cout << "Input ALPHA: ";
    cin >> ALPHA;
    cout << "Input TIME_STEP: ";
    cin >> TIME_STEP;

    P = problem.no_of_variables;
    Q = problem.no_of_inequality_constraints;
    R = problem.no_of_equality_constraints;

    f = problem.objective_function();
}
fn = 1; // f

gp = new long double[Q];
hp = new long double[R];
delta_x = new long double[P];

status = 0;
final_status = 0;

} // simulate operation of network

template<class domain>
int NeuralDynamics<domain>::simulate_network (const long double& _rn,
                                             const long double& _TS_factor)
{
    long double V = 0, old_V = 0; // pseudo-objective
    long double gp_sum_of_squares;
    long double hp_sum_of_squares, par;

    // get current constraint values
    g = problem.inequality_constraints (0);
    h = problem.equality_constraints (0);

    for (int i = 0; i < ITERATIONS; i++)
    {
        // find violated constraints constraint layer node outputs
        inequality_constraintsViolation();
        equality_constraintsViolation();

        gp_sum_of_squares = 0.0;
        for (int m = 0; m < Q; m++)
            gp_sum_of_squares = gp_sum_of_squares + (gp[m]*gp[m]);

        hp_sum_of_squares = 0.0;
        for (m = 0; m < R; m++)
            hp_sum_of_squares = hp_sum_of_squares + (hp[m]*hp[m]);

        // evaluate the pseudo-objective function
        V = fn + _rn * 0.5 * (gp_sum_of_squares+hp_sum_of_squares);

        if (i != 0)
if (abs((V - old_V)/old_V) <= OBJECTIVE_TOLERANCE)
{
    cout << "Convergence achieved in iteration " << i+1;
    return 1;
}
old_V = V;

// for each variable (node) calculate the updated output
for (int j = 0; j < P; j++)
{
    // Euler formula
    delta_x[j] = TIME_STEP * variable_node_input(j, _rm);
}

long double* xx = problem.current_var_state();
for (int k = 0; k < P; k++)
    xx[k] = xx[k] + delta_x[k];

// update section properties, objective function, constraints
// for the new design variables
problem.update_section_properties(xx);
fn = problem.objective_function() / f;
g = problem.inequality_constraints(0);
h = problem.equality_constraints(0);

}

cout << "Convergence not achieved! Max. iterations exhausted";
return 0;
}

// optimize the problem defined by class domain
template<class domain>
int NeuralDynamics<domain>::optimize()
{
    // rm is penalty parameter
    long double old_f, rm;
    long double* var;
    var = problem.current_var_state();
ofstream out ("n", ios::app);
for (int i = 0; i < P; i++) out << var[i] << "\n";

// run for Ni number of iterations
for (long double n = 0.0; n < Ni; n++)
{

// output the current objective function
out << n << " " << f*fn << "\n";

old_f = fn;

// update penalty parameter
m = I_PENALTY + n/ALPHA;
status = simulate_network (m);

// stop if the change in objective function is less than OT
if (abs((old_f - fn)/old_f) <= OT)
{
    final_status = 1;
    var = problem.current_var_state ();
    out << "\n";
    for (int i = 0; i < P; i++) out << i << " " << var[i] << "\n";
    out << "\n";
    for (i = 0; i < Q; i++) out << i << " " << g[i] << "\n";
    out << "\n";
    for (i = 0; i < R; i++) out << "\n" << i << " " << h[i];
    out << "\n" << "GOOD";
    out << "\n" << f*fn;
    return 1;
}

final_status = 0;
var = problem.current_var_state ();
out << "\n";
for (i = 0; i < P; i++) out << i << " " << var[i] << "\n";
out << "\n";
for (i = 0; i < Q; i++) out << i << " " << g[i] << "\n";
out << "\n";
for (i = 0; i < R; i++) out << "\n" << i << " " << h[i];
out << "\n" << "BAD" << "\n" << f*fn;
return 0;
}
// calculate input to variable layer

template<class domain>
long double NeuralDynamics<domain>::variable_node_input (const int & var_no,
const long double & _m)
{
    long double f_plus_h, delta_f;

    long double* x_plus_h = problem.current_var_state();

    // increase variable by STEP
    x_plus_h[var_no] = x_plus_h[var_no] + STEP;
    problem.update_section_properties(x_plus_h);

    // find objective function gradient
    f_plus_h = problem.objective_function ()/f;
    delta_f = (f_plus_h - fn)/(STEP);

    // find inequality constraints gradient
    long double g_sum = 0.0;
    g_plus_h = problem.inequality_constraints (1);
    for (int m = 0; m < Q; m++)
    {
        if (gp[m] == 0.0) continue;
        g_sum = g_sum + gp[m]*(g_plus_h[m] - g[m])/STEP;
    }
    g_sum = _m * g_sum;

    // find equality constraint gradient
    long double h_sum = 0.0;
    h_plus_h = problem.equality_constraints (1);
    for (m = 0; m < R; m++)
    {  
        h_sum = h_sum + hp[m]*(h_plus_h[m]-h[m])/STEP;
    }
    h_sum = _m * h_sum;

    x_plus_h[var_no] = x_plus_h[var_no] - STEP;
    problem.change_variables (x_plus_h);

    // return input to the variable node
    return (-delta_f - g_sum - h_sum);
}
// calculates output of inequality constraint nodes i.e. constraint violations
template<class domain>
void NeuralDynamics<domain>::inequality_constraintsViolation()
{
  for (int i = 0; i < Q; i++)
  {
    if (abs(g[i]) >= CONSTRAINT_TOLERANCE)
    {
      if (g[i] > 0.0)
      {
        gp[i] = g[i];
      }
      else
      {
        gp[i] = 0.0;
      }
    }
    else
    {
      gp[i] = 0.0;
    }
  }
}

// calculate output of equality constraint nodes i.e. violations
template<class domain>
void NeuralDynamics<domain>::equality_constraintsViolation()
{
  for (int i = 0; i < R; i++)
  {
    if (abs(h[i]) >= CONSTRAINT_TOLERANCE)
    {
      hp[i] = h[i];
    }
    else
    {
      hp[i] = 0.0;
    }
  }
}
// determine constraint violation within a tolerance (inequality constraints)
template <class domain>
int NeuralDynamics<domain>::inequality_constraints_tolerance ()
{
    for (int i = 0; i < Q; i++)
    {
        if (g[i] <= 0) continue;
        if ((g[i]-CT) > 0) return 0;
    }
    return 1;
}

// determine equality constraint violation within a tolerance
template <class domain>
int NeuralDynamics<domain>::equality_constraints_tolerance ()
{
    for (int i = 0; i < R; i++)
    {
        if (abs(h[i]) > CT) return 0;
    }
    return 1;
}

#endif

A.6  frac.H

// returns the fractional part of a long double value
#include "absolute.H"

long double frac (long double x)
{
    int y = x;
    long double z = x-y;
    return abs(z);
}
A.7  absolute.H

// This file defines functions that return the absolute of a value
// Syntax: abs (value); return type is same as value type

#ifndef _ABSOLUTE_H_
define _ABSOLUTE_H_

long double abs(long double x)
{
    if (x < 0.0) x = -x;
    return x;
}

float abs (float x)
{
    if (x < 0.0) x = -x;
    return x;
}
#endif

100
APPENDIX B

COMPANY CODE LISTING FOR CHAPTER 3

B.1 optimize.C

Same as Section A.1 (page 74).

B.2 ss_beam.H

#ifndef _SS_BEAM_H_
#define _SS_BEAM_H_

// class ss_beam defines a simply supported beam with udl and/or any number
// of concentrated loads on it having any lateral bracing condition

class SSBeam
{
    protected:
        int lfrd; // 1 = LRFD; 0 = ASD
        int lcases; // no. of load combinations
        long double span;
        long double udl;

    // various udl's l = live; d = dead; w = wind; s = snow; e = earthquake;
    // r = roof rain; lr = live roof
    long double udl_d, udl_l, udl_w, udl_e, udl_lr, udl_r, udl_s;

        int no_of_point_loads;
        long double* p_load;
        long double* p_load_d;
        long double* p_load_l;
        long double* p_load_location;
    // various load factors

    //...
long double lf_d[7], lf_i[7], lf_w[7], lf_e[7], lf_s[7], lf_r[7], lf_rr[7];

long double Mu[7];  // max. design moments for various load factors
long double Vu[7];   // max. design shear

int braced_throughout;  // 1 = full bracing; 0 = not fully braced
int b_points;           // no. of bracing points
long double unbraced_length[6];  // lengths of unbraced segments
long double Mmx[6][7];     // max. design moment in each ub. seg.

void input_data ();
void initialize_LFRD ();
void factored_udl (const int& if);
long double zero_shear (int aa);

public:
SSBeam ();
long double shear (long double x, int aa);
long double moment (long double x, int aa);
long double deflection_at_center (long double x, long double E);

};

#endif

B.3 ss_beam.C

#include <iostream.h>
#include <stdlib.h>
#include "ss_beam.H"
#include "absolute.H"

SSBeam::SSBeam ()
{
    input_data ();

    // initialize data
    for (int i = 0; i < 7; i++)
    {
        Mu[i] = Vu[i] = 0;
    }
if_d[i] = if_l[i] = if_w[i] = if_e[i] = if_s[i] = if_lr[i] = if_rr[i] = 0;
for (int j = 0; j < 6; j++)
    Mmx[j][i] = 0;
}

// initially assume ASD with only one load combination
cases = 1;
frd = 0;
if_d[0] = if_l[0] = 1;

// these load factors not used
if_w[0] = if_e[0] = if_s[0] = if_lr[0] = if_rr[0] = 0;

long double location = zero_shear(0);
Mu[0] = moment(location, 0);
Vu[0] = shear(0.0, 0);
long double tmp = shear(span, 0);
if (abs(tmp) >= abs(Vu[0])) Vu[0] = tmp;

// find max moment in each unbraced segment
long double dist1 = 0; long double dist2 = 0.0;
for (int j = 0; j < b_points+1; j++)
    {
        dist2 = dist2+unbraced_length[j];
        if (location < dist1 || location > dist2)
            {
                Mmx[j][0] = moment(dist1, 0);
                tmp = moment(dist2, 0);
                if (abs(tmp) >= abs(Mmx[j][0])) Mmx[j][0] = tmp;
            }
        else
            {
                Mmx[j][0] = Mu[0];
            }
        dist1 = dist2;
    }

// Finds moments and shear for all applicable load combinations
void SSBeam::initialize_LFRD()
{
    long double tmp, location;
    ...
cases = 7; // 7 load combinations

for (int i = 1; i < cases; i++)
{
    location = zero_shear(i);
    Mu[i] = moment(location, i);
    long double dist1 = 0; long double dist2 = 0.0;
    for (int j = 0; j < b_points+1; j++)
    {
        dist2 = dist2+unbraced_length[j];
        if (location < dist1 || location > dist2)
        {
            Mmx[j][i] = moment(dist1, i);
            tmp = moment(dist2, i);
            if (abs(tmp) >= abs(Mmx[j][i])) Mmx[j][i] = tmp;
        }
        else
        {
            Mmx[j][i] = Mu[i];
        }
        dist1 = dist2;
    }

    Vu[i] = shear(0.0, i);
    tmp = shear(span, i);
    if (abs(tmp) >= abs(Vu[i])) Vu[i] = tmp;
}

// input loading and bracing data for the beam
void SSBeam::input_data()
{
    cout << "Enter beam length: "; cin >> span;
    cout << "Is it braced throughout (1) or not (0): "; cin >> braced_throughout;

    if (braced_throughout != 1)
    {
        cout << "Enter no. of bracing points (max 5): "; cin >> b_points;
        for (int j = 0; j < b_points+1; j++)
        {
            cout << "From left, enter length of unbraced span " << j+1 << " ";
            cin >> unbraced_length[j];
        }
    }
}
else
{
    b_points = -1;
    for (int l = 0; l < 6; l++) unbraced_length[l] = 0;
}

cout << "Enter dead UDL on beam: "; cin >> udl_d;
cout << "Enter live UDL on beam: "; cin >> udl_l;
cout << "Enter snow UDL on beam: "; cin >> udl_s;
cout << "Enter wind UDL on beam: "; cin >> udl_w;
cout << "Enter earthquake UDL on beam: "; cin >> udl_e;
cout << "Enter roof live UDL on beam: "; cin >> udl_lr;
cout << "Enter roof rain UDL on beam: "; cin >> uld_rr;
cout << "Enter no. of point loads: ";
cin >> no_of_pointLoads;

if (no_of_pointLoads != 0)
{
    p_load_d = new long double[no_of_pointLoads];
    p_load_l = new long double[no_of_pointLoads];
    p_load = new long double[no_of_pointLoads];

    p_load_location = new long double[no_of_pointLoads];

    for (int i = 0; i < no_of_pointLoads; i++)
    {
        cout << "Enter dead component of load no. " << i + 1 << " : ";
        cin >> p_load_d[i];
        cout << "Enter live component of load no. " << i+1 << " : ";
        cin >> p_load_l[i];

        cout << "Enter location of load no. " << i + 1 << ": ";
        cin >> p_load_location[i];
        p_load[i] = p_load_d[i]+p_load_l[i];
    }
}
else
{
    p_load_d = p_load_l = p_load = 0;
    p_load_location = 0;
}
}
// Calculates the factored udl for load combination If
void SSBeam::factored_udl (const int & If)
{
    switch (If)
    {
    case 0:
        udl = If_d[If]*udl_d+If_l[If]*udl_l;
        break;
    case 1:
        udl = If_d[If]*udl_d+If_l[If]*udl_l;
        break;
    case 2:
        { long double tmp;
          if (udl_s >= udl_lr)
            { tmp = udl_s; }
          else
            { tmp = udl_lr; }
          if (tmp <= udl_rr) tmp = udl_rr;
          udl = If_d[If]*udl_d+If_l[If]*udl_l+If_s[If]*tmp;
          break;
        }
    case 3:
        { long double tmp, tmp1;
          if (If_s[If]*udl_s >= If_lr[If]*udl_lr)
            { tmp = If_s[If]*udl_s; }
          else
            { tmp = If_lr[If]*udl_lr; }
          if (tmp <= If_rr[If]*udl_rr) tmp = If_rr[If]*udl_rr;

          if (If_l[If]*udl_l) >= If_w[If]*udl_w)
            { tmp1 = If_l[If]*udl_l; }
          else
            { tmp1 = If_w[If]*udl_w; }

          udl = If_d[If]*udl_d+tmp+tmp1;
          break;
        }
    case 4:
        { long double tmp;
          if (udl_s >= udl_lr)
            { tmp = udl_s; }

106
else
    { tmp = udl_lr; }
if (tmp <= udl_rr) tmp = udl_rr;
udl = if_d[if]*udl_d+if_l[lf]*udl_l+if_w[lf]*udl_w+if_s[lf]*tmp;
break;
}
case 5:
{
long double tmp;
if (if_l[lf]*udl_l >= if_s[lf]*udl_s)
    { tmp = if_l[lf]*udl_l; }
else
    { tmp = if_s[lf]*udl_s; }
udl = if_d[lf]*udl_d+if_c[lf]*udl_c+tmp;
brea
}
case 6:
{
long double tmp;
if (abs(if_w[lf]*udl_w) >= abs(if_e[lf]*udl_e))
    { tmp = if_w[lf]*udl_w; }
else
    { tmp = if_e[lf]*udl_e; }
udl = if_d[lf]*udl_d+tmp;
brea
}
default:
    cout << "Error in assigning load factors!";
    exit (0);
}

// Calculates shear at loc (from left support) subjected to load combination aa
long double SSBeam::shear (long double loc, int aa)
{
    if (loc < 0.0 || loc > span)
    {
        cout << "Invalid location!"
        exit (1);
    }

    factored_udl(aa);
if (no_of_point_loads != 0)
{
    for (int i = 0; i < no_of_point_loads; i++)
    {
        p_load[i] = lf_d[aa]*p_load_d[i]+lf_1[aa]*p_load_1[i];
    }
}

long double sh = (udl*span*0.5 - udl*loc);
if (no_of_point_loads == 0) return sh;

long double r_reaction = 0.0, l_reaction = 0.0, sigma_y = 0.0;

for (int i = 0; i < no_of_point_loads; i++)
{
    sigma_y = sigma_y + p_load[i];
    r_reaction = r_reaction + p_load[i]*p_load_location[i];
}

r_reaction = r_reaction/span;
l_reaction = sigma_y - r_reaction;

long double a = 0.0, b = p_load_location[0];
long double ld = 0.0;
i = 0;

while (1)
{
    l_reaction = l_reaction - ld;
    if (loc >= a && loc <= b)
    {
        if (loc == b)
        {
            if (abs(l_reaction+sh) >= abs(l_reaction - p_load[i]+sh))
            {
                return (l_reaction + sh);
            }
            else
            {
                return (l_reaction - p_load[i] +sh); }
        }
    
return (l_reaction + sh);
    }
    if (i == no_of_point_loads - 1) break;
i++;
a = b;
b = p_load_location[i];
ld = p_load[i-1];
}

return (-r_reaction - (udl*span/2.0 - udl*(span-loc)));
}

// Calculates moment at loc (from left support) under load combination aa
long double SSBeam::moment (long double loc, int aa)
{
  if (loc < 0.0 || loc > span)
  {
    cout << "Invalid location!"
    exit (1);
  }

  factored_udl(aa);
  if (no_of_point_loads != 0)
  {
    for (int i = 0; i < no_of_point_loads; i++)
    {
      p_load[i] = if_d[aa]*p_load_d[i]+if_l[aa]*p_load_l[i];
    }
  }

  long double M1 = udl*span*loc*0.5 - udl*loc*loc*0.5;
  if (no_of_point_loads == 0) return M1;

  long double r_reaction = 0.0, l_reaction = 0.0, sigma_y = 0.0;

  for (int i = 0; i < no_of_point_loads; i++)
  {
    sigma_y = sigma_y + p_load[i];
    r_reaction = r_reaction + p_load[i]*p_load_location[i];
  }

  r_reaction = r_reaction/span;
  l_reaction = sigma_y - r_reaction;

  long double a = 0.0, b = p_load_location[0];
  long double M2 = 0.0, ld = 0.0;
  i = 0;
while (1)
{
    _reaction = _reaction - ld;
    if (loc >= a & & loc <= b)
    {
        M2 = M2 + _reaction*(loc - a);
        return (M1 + M2);
    }
    if (i == no_of_point_loads - 1) break;
    i++;

    M2 = M2 + _reaction*(b-a);
    ld = p_load[i-1];
    a = b;
    b = p_load_location[i];
}

return (r_reaction*(span-loc));

// returns location (from left support) of zero shear under load combination aa
long double SSBeam::zero_shear (int aa)
{
    if (no_of_point_loads == 0) return (span*0.5);

    factored_udl(aa);
    if (no_of_point_loads != 0)
    {
        for (int i = 0; i < no_of_point_loads; i++)
        {
            p_load[i] = if_d[aa]*p_load_d[i]+lf_l[aa]*p_load_l[i];
        }
    }

    long double l_sh, r_sh;
    for (int i = 0; i < no_of_point_loads; i++)
    {
        l_sh = shear (p_load_location[i] - 0.01, aa) - udl*0.01;
        r_sh = shear (p_load_location[i] + 0.01, aa) + udl*0.01;
        if (l_sh >= 0.0 && r_sh <= 0.0) return p_load_location[i];
    }
}
long double a = p_load_location[0], b = p_load_location[1];

for (i = 0; i < (no_of_point_loads - 1); i++)
{
    l_sh = shear (p_load_location[i] + 0.001, aa) + udl*0.001;
    r_sh = shear (p_load_location[i+1] - 0.001, aa) - udl*0.001;
    if (l_sh > 0.0 && & r_sh < 0.0)
    {
        return (p_load_location[i] + l_sh/udl);
    }
    cout << "Error!";
    exit (1);
}

// returns deflection at midspan. I = Moment of inertia; E = md of Elasticity
long double SSBcam::deflection_at_center (long double I, long double E)
{
    udl = udh_d+udl_l;
    if (no_of_point_loads != 0)
    {
        for (int i = 0; i < no_of_point_loads; i++)
            p_load[i] = p_load_d[i]+p_load_l[i];
    }

    long double d1 = 5*udl*span*span*span*span/(384*E*I);
    if (no_of_point_loads == 0) return d1;

    long double d2 = 0.0;
    long double l_c = span*0.5;
    for (int i = 0; i < no_of_point_loads; i++)
    {
        if (p_load_location[i] < l_c)
        {
            d2 = d2 + p_load[i]*p_load_location[i]*l_c/(6*E*I*span)*span*span
                - p_load_location[i]*p_load_location[i]-l_c*l_c;
            continue;
        }
        if (p_load_location[i] == l_c)
        {
            d2 = d2 + p_load[i]*span*span*span/(48*E*I);
            continue;
        }
        if (p_load_location[i] > l_c)
        {
            d2 = d2 + p_load[i]*(span-p_load_location[i])*l_c/(6*E*I*span)*
            // Further code...
        }
    }

    return d2;
}
(span*span-(span-p_load_location[i])*span-p_load_location[i]-l_c*l_c); continue;
}
}
return d1 + d2;
}

B.4 sec_opt.H

#ifdef_SEC_OPT_H_
#define_SEC_OPT_H_

#ifdef TRUE
#define TRUE 1
#define FALSE 0
#endif

#include "ss_beam.H"
#include "aisi.H"
#include "prop.H"

// optimum design problem for cold-formed steel beams
class OptimumDesign
 : public SSBeam
{
public:
  AISI code;         // code routines access
  int type;          // section shape
  int cot;           // 1 means compression of top
  enum {NOV = 4};    // no. of variables

  // x[0] = flat portion of flange (top)
  // x[1] = flat portion of flange (bottom)
  // x[2] = flat portion of webs
  // x[3] = thickness of section

  long double x[4];   // variables
  long double x0, x1; // top and bottom flanges widths (single)
  long double tl, tu, hl, hu; // lower and upper limits on thickness and height
  long double Fy;
long double E;
long double lim_deflection;
long double rad;       // inner radius
long double N;         // bearing plate width at reactions and load
long double cf;        // amount of flange curling allowed
long double fl_sl_ratio1;
long double fl_sl_ratio0;
long double web_sl_ratio;
long double pf, mf;

Properties full_section;       // properties of a full unreduced section

long double Ma1, Ma0;          // allowable moment
long double Mab1[6], Mab0[6];  
long double Va;                // allowable shear
long double wf;
long double delta;

long double Pa[4];

long double loc[7];           // location of max moment and shear interaction
long double M_and_S[7];       // moment and shear interaction

Properties full_section_properties();

long double moment_capacity();
void unbraced_moment_capacity();
long double shear_capacity();
long double actual_deflection();
long double combined_bending_and_shear(int If, int index);
long double web_crippling_strength(int i_condition);
long double flange_curling_limitation();
long double fl_slenderness_ratio0();
long double fl_slenderness_ratio1();
long double web_slenderness_ratio();

void copy_max (long double* _a, long double* _b);
void update_section_properties (long double* _x); // update/change section
void update_section_properties();
void change_variables (long double* _x);
long double* current_var_state();
int no_of_inequality_constraints;
int no_of_equality_constraints;
int no_of_variables;

long double* gg;
long double* gg_plus_h;
long double* hh;
long double* hh_plus_h;

long double objective_function();
long double* inequality_constraints (int plus);
long double* equality_constraints (int plus);
OptimumDesign ();
);

#endif

B.5  sec_opt.C

#include "prop.H"
#include "sec_opt.H"
#include <iostream.h>
#include <math.h>

OptimumDesign::OptimumDesign ()
: SSBeam ()
{
  cout << "Which code LFRD (1) or ASD (2): ";
  cin >> lfrd;
  cout << "Enter type of section (Hat = 1, I = 2, Z = 3): ";
  cin >> type;

  no_of_variables = NOV;
  if (no_of_point_loads == 0)
  {
    no_of_inequality_constraints = 23;
  }
  else
  {
    no_of_inequality_constraints = 23+2*no_of_point_loads;
  }
  no_of_equality_constraints = 1;
gg = new long double[no_of_inequality_constraints];
gh = new long double[no_of_inequality_constraints];
hh = new long double[no_of_inequality_constraints];
ghh = new long double[no_of_inequality_constraints];

cout << "Enter dimensions of sections\n";
cout << "Flat width of top flange? ";
cin >> x[0];
cout << "Flat width of bottom flange? ";
cin >> x[1];
cout << "Flat height of webs? ";
cin >> x[2];
cout << "Thickness? ";
cin >> x[3];

cout << "Lower limit? "; cin >> tl;

cout << "Upper limit? "; cin >> tu;
cout << "Inner radius of corner elements? ";
cin >> rad;
cout << "Width of plate used at conc. loads and reactions? ";
cin >> N;
cout << "Allowable amount of flange curling? ";
cin >> cf;
cout << "Limiting deflection? :\n";
cin >> lim_deflection;
cout << "Fy? ";
cin >> Fy;
cout << "E? ";
cin >> E;

// calculate max. distance between concentrated loads. Used for
// shear lag considerations
long double wd;
if (no_of_point_loads == 0) wd = 0;
if (no_of_point_loads == 1) wd = span;
if (no_of_point_loads > 1)
{
  long double dist;
  long double m_dist = 0;
  for (int i = 1; i < no_of_point_loads; i++)
  {
    dist = p_load_location[i]-p_load_location[i-1];
    if (dist >= m_dist) m_dist = dist;
  
  // calculate max. distance between concentrated loads. Used for
  // shear lag considerations
  long double wd;
  if (no_of_point_loads == 0) wd = 0;
  if (no_of_point_loads == 1) wd = span;
  if (no_of_point_loads > 1)
  {
    long double dist;
    long double m_dist = 0;
    for (int i = 1; i < no_of_point_loads; i++)
    {
      dist = p_load_location[i]-p_load_location[i-1];
      if (dist >= m_dist) m_dist = dist;
```
wd = m_dist;

// pass info to code
code.pass_const_info (lfrd, type, braced_throughout, span, wd, Mu[0], Vu[0],
                     rad, N, cf, Fy, E);

// if (lfrd == 1)
// {
//   for (int i = 1; i < 7; i++)
//   {
//     code.load_factors (i, if_d[i], if_l[i], if_w[i], if_e[i], if_s[i],
//                        if_lr[i], if_rr[i]);
//   }
// }

SSBeam::initialize_LFRD();

//

web_sl_ratio = code.web_slenderness_ratio();
code.combined_bending_and_crippling (pf, mf);

cot = 0;
code.bending_sense (cot);
fl_sl_ratio0 = code.fl_slenderness_ratio();

cot = 1;
code.bending_sense (cot);
fl_sl_ratio1 = code.fl_slenderness_ratio();

if (braced_throughout)
{
  for (int k = 0; k < 6; k++) Mab0[k] = Mab1[k] = 0;
}

x0 = x[0];
x1 = x[1];
if (type == 2) x0 = x[0]*0.5, x1 = x[1]*0.5;
code.pass_info (x0, x1, x[2], x[3]);

full_section = full_section_properties();
Mab1 = moment_capacity();
unbraced_moment_capacity();
wf = flange_curling_limitation();
Va = shear_capacity();

116
Pa[0] = web_rippling_strength (1);
Pa[1] = web_rippling_strength (2);
Pa[2] = web_rippling_strength (3);
Pa[3] = web_rippling_strength (4);

delta = actual_deflection ();

if (/*if r <= 1 && */Mu[6] < 0)
{
    cot = 0;
    code.bending_sense (cot);
    full_section = full_section_properties ();
    M_0 = moment_capacity ();
    unbraced_moment_capacity ();
}
else
{
    M_0 = 0;
}

for (int j = 0; j < Icases; j++)
    M_and_S[j] = combined_bending_and_shear (j, 0);

}

long double OptimumDesign::objective_function ()
{
    long double fac = 2.0;
    if (type == 3) fac = 1.0;
    return (x[0]+x[1]+fac*x[2]+fac*PI*rad*rad*0.5)*x[3]*Span*7.7e-5; //DENSITY
}

long double* OptimumDesign::inequality_constraints (int plus)
{
    long double* a;
    if (plus == 0)
    {
        a = gg;
    }
    else
    {
        a = gg_plus_h;
    }
int wind = 0;
if (Mu[6] < 0) wind = 1;
long double fac = 1;
if (type == 1) fac = 0.5;

a[0] = (x0*fac+rad)/wf-1.0;
a[1] = (x0/x[3])/fl_sl_ratio1-1.0;
if (wind == 1)
{
a[2] = (x1/x[3])/fl_sl_ratio0-1.0;
}
else
{
a[2] = 0;
}

a[3] = (x[2]/x[3])/web_sl_ratio-1.0;
a[4] = delta/lsl_deflection - 1.0;
a[5] = -x[0];
a[6] = -x[1];
a[7] = -x[2];
a[8] = 1.1*(1-x[3]/tl);
a[9] = 1.1*(x[3]/tu-1);
a[10] = x[0]/x[1]-1;
a[11] = x[1]/x[0]-1;

long double b[50], Md;
long double* Mdb;
for (int ii = 0; ii < 50; ii++) b[ii] = 0.0;
long double lm = 1;
if (frd == 1) lm = 1;

long double* c;
for (int i = lm; i < 7/*icases*/; i++)
{
c = b;
if (i == lm) c = a;
Md = Ma1;
Mdb = Mab1;
if (i == 6 & & wind == 1) {Md = Ma0, Mdb = Mab0;}
c[12] = 1.2*(abs(Mu[i])/Md-1.0);
c[13] = abs(Vu[i])/Va-1.0;
c[14] = (M_and_S[i] - 1.0);
for (int j = 0; j < 6; j++)
{
    if (!!(braced_throughout != 1))
    {
        if (!(j > b_points))
        {
            c[15+j] = abs(Mmax[i][j]/Mdb[j]-1.0);
        } else
        {
            c[15+j] = 0;
        }
    } else
    {
        c[15+j] = 0.0;
    }
}

if (no_of_point_loads == 0)
{
    c[21] = 1.2*(abs(SSBeam::shear (0.0, i))/Pa[0]-1.0);
    c[22] = 1.2*(abs(SSBeam::shear (span, i))/Pa[0]-1.0);
    if (!@cases == -1) return a;
    if (i == 6m)
    {
        continue;
    } else
    {
        copy_max (a, b);
        if (i == 6) return a;
        continue;
    }
}

if (((p_load_location[0]-N) < (1.5*x[21]))
{
    c[21] = abs(SSBeam::shear(0.0, i))/Pa[2]-1.0;
    c[23] = abs(SSBeam::shear(p_load_location[0], i))/Pa[3]-1.0;
    c[24] = pf*p_load[0]/Pa[3]+mf*abs(SSBeam::moment (p_load_location[0], i))/Md -1.0;
else
{
  c[21] = abs(SSBeam::shear(0.0, i))/Pa[0]-1.0;
  if (no_of_point_loads == 1 && (span-p_load_location[0]-N) < (1.5*x[2]))
  {
    c[22] = abs(SSBeam::shear(span, i))/Pa[2]-1.0;
    c[23] = abs(SSBeam::shear(p_load_location[0], i))/Pa[3]-1.0;
    c[24] = pf*p_load[0]/Pa[3]+mf*abs(SSBeam::moment(p_load_location[0], i))/Md -1.0;
    if (lcases == 1) return a;
    if (i == lm)
    {
      continue;
    }
    else
    {
      copy_max (a, b);
      if (i == 6) return a;
      continue;
    }
  }
  c[23] = abs(SSBeam::shear(p_load_location[0], i))/Pa[1]-1.0;
  c[24] = pf*p_load[0]/Pa[1]+mf*abs(SSBeam::moment (p_load_location[0], i))/Md -1.0;
}

if (no_of_point_loads == 1)
{
  c[22] = abs(SSBeam::shear(span, i))/Pa[0]-1.0;
  if (lcases == 1) return a;
  if (i == lm)
  {
    continue;
  }
  else
  {
    copy_max (a, b);
    if (lm == 6) return a;
    continue;
  }
}
int 1 = no_of_point_loads-1;
if ((span-p_load_location[1]-N) < (1.5*x[2]))
{
    c[22] = abs(SSBeam::shear(span, i))/Pa[2]-1.0;
    c[25] = abs(SSBeam::shear(p_load_location[1], i))/Pa[3]-1.0;
    c[26] = pf*p_load[1]/Pa[3]+mf*abs(SSBeam::moment(p_load_location[1], i))/Md
            -1.0;
}
else
{
    c[22] = abs(SSBeam::shear(span, i))/Pa[0]-1.0;
    c[25] = abs(SSBeam::shear(p_load_location[1], i))/Pa[1]-1.0;
    c[26] = pf*p_load[1]/Pa[1]+mf*abs(SSBeam::moment(p_load_location[1], i))/Md
            -1.0;
}
if (no_of_point_loads == 2)
{
    if (lcases == 1) return a;
    if (i == lm)
    {
        continue;
    }
    else
    {
        copy_max (a, b);
        if (i == 6) return a;
        continue;
    }
}
int id = 27;
for (int k = 1; k < l; k++)
{
    c[id] = abs(SSBeam::shear(p_load_location[k], i))/Pa[1]-1.0;
    c[id + 1] = pf*p_load[k]/Pa[1] + mf*abs(SSBeam::moment(p_load_location[k], i))/Md
            -1.0;
    id = id + 2;
}
if (lcases == 1) return a;
if (i == lm)
{
    continue;
}
} 
else 
{
    copy_max (a, b);
    if (i == 6) return a;
    continue;
}
}

long double* OptimumDesign::equality_constraints (int plus) 
{
    long double* a;
    if (plus == 0)
    {
        a = hh;
    }
    else
    {
        a = hh_plus_h;
    }
    a[0] = 0;
    return a;
}

void OptimumDesign::copy_max (long double* _a, long double* _b) 
{
    for (int i = 12; i < no_of_inequality_constraints; i++)
    {
        if (_b[i] >= _a[i]) _a[i] = _b[i];
    }
}

Properties OptimumDesign::full_section_properties ()
{
    return code.section_properties (x0, x1, x[2], 0.0, 0.0);
}

// AISI sec C3.1.1a
long double OptimumDesign::moment_capacity ()
{
    return code.moment_capacity();

122
void OptimumDesign::unbraced_moment_capacity()
{
    Properties fl = full_section_properties();
    long double* tmp = Mab1;
    if (cot != 1) tmp = Mab0;

    for (int i = 0; i < b_points+1; i++)
    {
        tmp[i] = code.moment_capacity_unbraced(unbraced_length[i], fl);
    }
}

// AISC sec C3.2
long double OptimumDesign::shear_capacity()
{
    return code.shear_capacity();
}

long double OptimumDesign::actual_deflection()
{
    return SSBeam::deflection_at_center(code.deflection_I(full_section), E);
}

// AISC sec C3.3
long double OptimumDesign::combined_bending_and_shear(int If, int index)
{
    long double mom, sh, com, b;
    long double max_com = 0.0;
    long double Md = Ma1;
    if (If == 6 && Mu[If] < 0) Md = Ma0;

    if (index == 0) // at start index = 0
    {
        for (long double a = 0.0; a <= (span); a += span/14)
        {
            mom = SSBeam::moment(a, If);
            sh = SSBeam::shear(a, If);
            com = code.combined_bending_and_shear(mom, Md, sh, Va);
            if (com >= max_com)
            {
                max_com = com;
                b = a;
            }
        }
    }
}
if (no_of_point_loads != 0)
{
    for (int i = 0; i < no_of_point_loads; i++)
    {
        mom = SSBem::moment (p_load_location[i], lf);
        sh = SSBem::shear (p_load_location[i], lf);
        com = code.combined_bending_and_shear (mom, Md, sh, Va);
        if (com >= max_com)
        {
            max_com = com;
            b = p_load_location[i];
        }
    }
}
loc[lf] = b;

mom = SSBem::moment (loc[lf], lf);
sh = SSBem::shear (loc[lf], lf);
com = code.combined_bending_and_shear (mom, Md, sh, Va);
return com;

// AISI sec C3.4
long double OptimumDesign::web_crippling_strength (int l_condition)
{
    return code.web_crippling_strength (l_condition);
}

long double* OptimumDesign::current_var_state ()
{
    return x;
}

void OptimumDesign::change_variables (long double* _x)
{
    x[0] = _x[0];
    x[1] = _x[1];
    x[2] = _x[2];
    x[3] = _x[3];

void OptimumDesign::update_section_properties (long double* _x)
{
    change_variables (_x);

    x0 = _x[0];
    x1 = _x[1];
    if (type == 2) x0 = x0*0.5, x1 = x1*0.5;
    code.pass_info (x0, x1, x[2], x[3]);

    cot = 1;
    code.bending_sense (cot);
    full_section = full_section_properties ();
    Ma1 = moment_capacity ();
    unbraced_moment_capacity ();
    wf = flange_curling_limitation ();
    Va = shear_capacity ();
    Pa[0] = web_crippling_strength (1);
    Pa[1] = web_crippling_strength (2);
    Pa[2] = web_crippling_strength (3);
    Pa[3] = web_crippling_strength (4);

    delta = actual_deflection ();

    if (*!frd == 1 && *!/Mu[6] < 0)
    {
        cot = 0;
        code.bending_sense (cot);
        Ma0 = moment_capacity ();
        unbraced_moment_capacity ();
    }

    for (int j = 0; j < lcases; j++)
        M_and_S[j] = combined_bending_and_shear (j, 1);
}

void OptimumDesign::update_section_properties ()
{
    cot = 1;
    code.bending_sense (cot);

    full_section = full_section_properties ();
}
Ma1 = moment_capacity();
unbraced(moment_capacity());
wf = flange_curling_limitation();
V = shear_capacity();

Pa[0] = web_crippling_strength(1);
Pa[1] = web_crippling_strength(2);
Pa[2] = web_crippling_strength(3);
Pa[3] = web_crippling_strength(4);

delta = actual_deflection();

if (**flrd == 1 && */Mu[6] < 0)
{
    cot = 0;
    code.bending_sense(cot);
    Ma0 = moment_capacity();
    unbraced(moment_capacity());
}

for (int j = 0; j < icases; j++)
    M_and_S[j] = combined_bending_and_shear(j, 1);

// AISI sec B1.1b
long double OptimumDesign::flange_curling_limitation()
{
    return code.flange_curling_limitation();
}

B.6 aisi.H

#ifndef _AISI_H_
define _AISI_H_

#include "prop.H"
#include "absolute.H"
#ifndef PI
#define PI 3.141592654
#endif
// class defining the AISI ASD and LRFD codes for design of
// cold-formed steel beams
class AISI {
private:
    int LFRD; // 1 = LRFD; 0 = ASD
    int cot; // cot = 1 = bending with compression on top
        // cot = 0 = bending with compression on bottom
int is_braced; // 1 = fully braced; 0 = otherwise
int sec_type; // 1 = hat; 2 = I; 3 = Z
long double L; // span
long double bw_loads_dist; // max. dist. b/w conc. loads (for dshear lag)
long double now; // no. of webs

long double phi_b; // resistance factors - bending
long double phi_wv; // resistance factor - web shear
long double phi_wc; // resistance factor - web crippling

long double Fy;
long double E;
long double w; // flat width of compression flange
long double wt; // flat width of tension flange
long double dp; // flat depth of web
long double t; // thickness of section
long double rd; // radius of corner elements
long double A_ce; // area of one corner element
long double y_ce; // cg dist. from outer edge of corner element
long double I_ce;
long double N; // length of bearing plate at conc. loads
long double cf; // allowable flange curling
long double Mapp; // applied max. moment--no load factors
long double Vapp; // applied max. shear
long double sl_ratio; // limiting slenderness ratio for shear lag

// general function for calculating effective widths
long double eff_widths (const long double& bf, const long double& f,
    const long double& k, const int& typ);

long double inelastic_reserve_capacity (const long double& _l_com,
    const long double& _M);

long double web_rippling_strength_LFRD (int l_condition);
void resistance_factors ();
void corner_element_properties ();
long double shear_lag ();
long double moment_capacity_backling ();
long double moment_capacity_shearlag ();

public:
void load_factors (int no, long double& d, long double& l, long double& wi,
    long double& e, long double& s, long double& lr, long double& rr);
void bending_sense (int a);
void pass_info (const long double& _w, const long double& _wt,
    const long double& _d, const long double& _t);
void pass_const_info (const int& ct, const int& type, int is_braced,
    const long double& _L, const long double& _bw_loads_dist,
    const long double& _Mapp, const long double& _Vapp,
    const long double& _r, const long double& _N,
    const long double& _cf, const long double& _Fy, const long double& _E);

Properties section_properties (long double x0, long double x1,
    const long double& x2a,
    const long double& x2b, const long double& y_ass);

long double eff_width_of_uces (const long double& _f, const int& _typ);
long double eff_width_of_uces (const long double& _f, const int& _typ);
void eff_height_of_web (long double& _b1, long double& _b2,
    const long double& _f1, const long double& _f2, const int& _typ);

long double moment_capacity ();
long double moment_capacity_unbraced (const long double& Lu, Properties full);
long double shear_capacity ();
long double deflection_I (Properties pr);
long double combined_bending_and_shear (const long double& mom,
    const long double& _Ma, const long double& sh, const long double& _Va);
long double web_crippling_strength (int I_condition);
void combined_bending_and_crippling (long double& pf, long double& mf);
long double flange_curling_limitation ();
long double fl_slenderness_ratio ();
long double web_slenderness_ratio ();

AISI ();

);
B.7 aisi.C

#include "aisi.H"
#include <iostream.h>
#include <stdlib.h>
#include <math.h>

AISI::AISI ()
{
}

// changes value of cot i.e. the bending sense
void AISI::bending_sense (int k)
{
    cot = k;
}

// pass section dimensions to AISI
void AISI::pass_info (const long double& _w, const long double& _wt,
            const long double& _d, const long double& _t)
{
    w = _w;
    wt = _wt;
    dp = _d;
    t = _t;
    sl_ratio = shear_lag ();
    corner_element_properties ();
}

// pass const. info to AISI
void AISI::pass_const_info (const int& ct, const int& type, int _is_braced,
            const long double& _L, const long double& _bw_loads_dist,
            const long double& _Mapp, const long double& _Vapp,
            const long double& _r, const long double& _N,
            const long double& _cf, const long double& _Fy,
            const long double& _E)
{
    LFRD = ct;
    sec_type = type;
    if (sec_type == 1 || sec_type == 2) now = 2;
    if (sec_type == 3) now = 1;
    is_braced = _is_braced;
    L = _L;

bw_loads_dist = _bw_loads_dist;
Mapp = _Mapp;
Vapp = _Vapp;
rd = _r;
N = _N;
cf = _cf;
Fy = _Fy;
E = _E;

cot = 1;
resistance_factors();
}

// initialize load factors for the appropriate specification
void AISI::load_factors (int no, long double& d, long double& l,
                        long double& wi,
                        long double& e, long double& s, long double& lr, long double& rr)
{
    // d = dead; l = live; wi = wind; s = snow; rr = roof rain; lr = live roof

    if (LFRD == 1)
    {
        switch (no)
        {
            case 1:
                d = 1.4;
                l = 1.0; wi = e = s = lr = rr = 0;
                break;
            case 2:
                d = 1.2; l = 1.6; lr = s = rr = 0.5;
                wi = e = 0;
                break;
            case 3:
                d = 1.2; l = 0.5; wi = 0.8; lr = 1.4; s = 1.6; rr = 1.6; e = 0;
                break;
            case 4:
                d = 1.2; wi = 1.3; l = 0.5; lr = rr = s = 0.5; e = 0;
                break;
            case 5:
                d = 1.2; e = 1.5; l = 0.5; s = 0.2; wi = rr = lr = 0;
                break;
            case 6:
                d = 0.9; wi = -1.3; e = -1.5; l = s = lr = rr = 0;
                break;
        }
    }
}
break;
default:
cout << "Error in AISI load factors!";
exit (0);
}
}
else
{
if (no == 1)
{
    d = l = 1.0; wi = e = s = lr = rr = 0;
}
if (no == 2)
{
    d = l = wi = s = 0.75;
    e = lr = rr = 0;
}
if (no == 6)
{
    d = 0.9; wi = -0.75;
    l = s = e = lr = rr = 0;
}
if (no == 3 || no == 4 || no == 5)
{
    d = l = s = e = wi = lr = rr = 0;
}
}

// initialize resistance factors
void AISI::resistance_factors()
{
    // resistance factors for beams only

    if (LFRD == 1)
    {
        if (sec_type == 1 && cot == 1)
        {
            phi_b = 0.95; // hat sections; stiffened flanges
        }
        else
        {
            phi_b = 0.9; // I and Z sections; unstiffened flanges
        }
    }
    }
phi_wv = 0.9;
if (sec_type == 1 || sec_type == 3)
{
  phi_wc = 0.75;  // hat and Z sections
}
else
{
  phi_wc = 0.8;  // I-sections
}
else
{
  phi_b = 0.6;
  phi_wv = 1.0;
  phi_wc = 1.0;
}

// calculate section properties
Properties AISI::section_properties (long double x0, long double x1,
const long double& x2a, const long double& x2b,
const long double& y_ass)
{

  long double Ay = 0.0;  // sigma A*y
  long double Ayy = 0.0;  // sigma A*y*y
  if (sec_type == 2) x0 = 2*x0, x1 = 2*x1;

  // for top flange
  long double area1 = x0*t;
  Ay = Ay + area1 * t * 0.5;
  Ayy = Ayy + area1 * (t * 0.5)*(t * 0.5);

  // for bottom flange
  long double area3 = x1*t;
  Ay = Ay + area3 * (dp + 2*rd + 1.5*t);
  Ayy = Ayy + area3 * (dp + 2*rd + 1.5*t)*(dp + 2*rd + 1.5*t);

  // for corner elements
  long double noce = 2;
  if (sec_type == 3) noce = 1;
  long double area4 = noce*A_ce;  // the upper corner(s)
\[Ay = Ay + area4 \times y_{ce};\]
\[Ayy = Ayy + area4 \times y_{ce} \times y_{ce};\]

\[Ay = Ay + area4 \times (rd + t + dp + (rd-(y_{ce}-t)));\]
\[Ayy = Ayy + area4 \times (rd + t + dp + (rd-(y_{ce}-t))) \times (rd + t + dp + (rd-(y_{ce}-t)));\]

Properties prop;
long double y_{tg}, ymax;
long double d = dp+2*rd+2*t;

// if web is fully effective
if (x2a == dp)
{
    long double area2 = now \times x2a \times t;
    Ay = Ay + area2 \times (x2a \times 0.5 + rd + t);
    Ayy = Ayy + area2 \times (x2a \times 0.5 + rd + t) \times (x2a \times 0.5 + rd + t);
    prop.A = area1 + area2 + area3 + 2*area4;
    prop.y_{cg} = Ay/prop.A;
    prop.I = Ayy + now \times (t \times x2a \times x2a \times x2a)/12 - prop.A \times prop.y_{cg} \times prop.y_{cg};
    if (cot != 1) prop.y_{cg} = d-prop.y_{cg};
    y_{tg} = d-prop.y_{cg};
    ymax = prop.y_{cg};
    if (y_{tg} >= prop.y_{cg}) ymax = y_{tg};
    prop.S = prop.I/ymax;
    prop.Sc = prop.I/prop.y_{cg};
    return prop;
}

// if web is not fully effective
long double l_{com} = y_{ass} - rd - t;
long double y_{t} = dp - l_{com};

if ((x2a+x2b) > l_{com})
{
    cout << "Dimensions not right!";
    exit (0);
}

long double area2a, area2b;
if (cot == 1)
{
    

133
area2a = now*x2a*t;
Ay = Ay + area2a * (x2a*0.5 + rd + t);
Ayy = Ayy + area2a * (x2a*0.5 + rd + t)*(x2a*0.5 + rd + t);

area2b = now*(y_t+x2b)*t;
Ay = Ay + area2b * (t+rd+dp-0.5*(y_t+x2b));
Ayy = Ayy + area2b * (t+rd+dp-0.5*(y_t+x2b))*(t+rd+dp-0.5*(y_t+x2b));
}
else
{
area2a = now*x2a*t;
Ay = Ay + area2a * (d-(x2a*0.5 + rd + t));
Ayy = Ayy + area2a * (d-(x2a*0.5 + rd + t))*(d-(x2a*0.5 + rd + t));

area2b = now*(y_t+x2b)*t;
Ay = Ay + area2b * (t+rd+0.5*(y_t+x2b));
Ayy = Ayy + area2b * (t+rd+0.5*(y_t+x2b))*(t+rd+0.5*(y_t+x2b));
}

prop.A = area1 + area2a + area2b + area3 + 2*area4;
prop.y_cg = Ay/prop.A;
prop.I = Ayy + now*(t*x2a*x2a*x2a/12 + t*x2b*x2b*x2b/12)
-prop.A*prop.y_cg*prop.y_cg;
if (cot != 1) prop.y_cg = d-prop.y_cg;

y_tg = d-prop.y_cg;
ymax = prop.y_cg;
if (y_tg >= prop.y_cg) ymax = y_tg;
prop.S = prop.I/ymax;
prop.Sc = prop.I/prop.y_cg;
return prop;

// calculates corner element properties
void AISI:corner_element_properties ()
{
  long double a1 = PI*rd*rd*0.25;
  long double a2 = PI*(rd+t)*(rd+t)*0.25;
  A_ce = a2 - a1;
  y_ce = (a2*((rd+t)-4*(rd+t))/(3*PI))
    - a1*(rd - 4*rd/(3*PI) + t)) / A_ce;
  // find the I value

134
\[ I_{ce} = \pi \cdot 0.0625 \cdot (rd + t)(rd + t)(rd + t)(rd + t) - rd \cdot rd \cdot rd \cdot rd \]
\[ - A_{ce} \cdot (rd + t - y_{ce}) \cdot (rd + t - y_{ce}) \]

// general function for calculating effective widths of elements
long double AISI::eff_widths (const long double& bf, const long double& f,
const long double& k, const int& typ)
{
    long double lambda = (1.052/sqrt(k)) * (bf/t) * sqrt(f/E);
    long double rho;

    // typ = 1 means for load determination. typ != 1 deflection determination
    if (typ == 1) // for load capacity determination
    {
        if (lambda <= 0.673)
        {
            return bf;
        }
        else
        {
            rho = (1-0.22/lambda)/lambda;
            if (rho >= 1.0) rho = 1.0;
            return rho*bf;
        }
    }

    if (sec_type != 1)
    {
        if (lambda <= 0.673)
        {
            return bf;
        }
        else
        {
            rho = (1-0.22/lambda)/lambda;
            if (rho >= 1.0) rho = 1.0;
            return rho*bf;
        }
    }

    long double lambda_c = 0.256 + 0.328*(bf/t)*sqrt(Fy/E);
    if (lambda <= 0.673) return bf;
if (lambda > 0.673 && lambda <= lambda_c)
{
    rho = (1.358-0.461/lambda)/lambda;
}
if (lambda >= lambda_c)
{
    rho = (0.41+0.59*sqrt(Fy/f)-0.22/lambda)/lambda;
}
if (rho >= 1.0) rho = 1.0;
return rho*bf;

// ucse = uniformly compressed stiffened element
long double AISI::eff_width_of_ucse (const long double& _f, const int& _typ)
{
    long double _bf = w;
    if (cot != 1) _bf = wt;
    return eff_widths (_bf, _f, 4.0, _typ);
}

// ucue = uniformly compressed unstiffened element
long double AISI::eff_width_of_ucue (const long double& _f, const int& _typ)
{
    long double _bf = w;
    if (cot != 1) _bf = wt;
    return eff_widths (_bf, _f, 0.43, _typ);
}

// effective height of web
void AISI::eff_height_of_web (long double& _b1, long double& _b2,
const long double& _f1, const long double& _f2, const int& _typ)
{
    long double psi = _f2/_f1;
    long double _k = 4+2*(1-psi)*(1-psi)*(_f1-_f2)*(_f1+_f2);  
    long double be = eff_widths (dp, _f1, _k, _typ);
    _b1 = be/(3-pi);
    if (psi <= -0.236)
    {
        _b2 = be*0.5;
    }
}
else
{
    _b2 = be-_b1;
}
}

// calculate moment capacity
long double AISI::moment_capacity ()
{
    resistance_factors ();
    long double Mb = moment_capacity_buckling ();
    if (sl_ratio == 1) return Mb;

    long double Ms = moment_capacity_shearlag ();
    if (Ms < Mb) Mb = Ms;
    return Mb;
}

// lateral bending strength of the unbraced segments
long double AISI::moment_capacity_unbraced (const long double & Lu,
                                          Properties full)
{
    long double Mc, Me, My, lyc, d, dw, bf;
    d = dp+2*rd+2*t;
    bf = w;
    if (cot != 1) bf = wt;
    dw = full.y_cg-rd-t;

    if (sec_type == 2)
    {
        lyc = 2*(t*bf*bf*bf/12+bf*t*(0.5*bf+rd+t)*(0.5*bf+rd+t))
            + 2*(I_ce+A_ce*y_ce*y_ce)
            + 2*(t*dw*0.5*t*0.5*t);
    }
    if (sec_type == 3)
    {
        // compression flange
        lyc = t*bf*bf*bf/12+t*bf*(0.5*bf+rd+0.5*t)*(0.5*bf+rd+0.5*t)
            + I_ce+A_ce*(y_ce-0.5*t)*(y_ce-0.5*t)
            + t*dw*0.5*0.25*t*0.25*t;
    }

    Me = PI*PI*E*1*d*lyc/(Lu*Lu);


137
if (sec_type == 3) Me = Me*0.5;
My = full.Sc*Fy;

if (Me >= 2.78*My)
{
    Mc = My;
}
if ((Me < 2.78*My) && (Me > 0.56*My))
{
    Mc = 10*My/9*(1-10*My/(36*Me));
}
if (Me <= 0.56*My)
{
    Mc = Me;
}

Properties pr;
long double f = Mc/full.Sc;
long double b = eff_width_of_ucue (f, 1);
if (cot == 1)
    {pr = section_properties (b, wt, dp, 0.0, 0.0);}
else
    {pr = section_properties (w, b, dp, 0.0, 0.0);}

long double b1, b2 = 0.0;
long double f1, f2 = 0.0;

long double l_com, old_y;

l_com = pr.y_cg-rd-t;
f1 = f/pr.y_cg*l_com;
f2 = f/pr.y_cg*(dp-l_com);

eff_height_of_web (b1, b2, f1, f2, 1);

long double rf = 0.6;
if (LFRD == 1) rf = 0.9;

if (((b1+b2) >= l_com)
{
    return rf*pr.Sc*Mc/full.Sc;
}

old_y = pr.y_cg;
if (cot == 1)
  {pr = section_properties (b, wt, b1, b2, old_y);}  
else 
  {pr = section_properties (w, b, b1, b2, old_y);}  

return rf*pr.Sc*Mc/full.Sc;

// bending strength based on local buckling considerations 
long double AISI::moment_capacity_buckling ()
{
  Properties pr;

  // find b assuming NA at mid-depth or below 
  long double b;
  if (sec_type == 1 && cot == 1)
    {
      b = eff_width_of_ucse (Fy, 1);
    }
  else
    {
      b = eff_width_of_ucue (Fy, 1);
    }
  if (cot == 1)
    {pr = section_properties (b, wt, dp, 0.0, 0.0);}  
else
    {pr = section_properties (w, b, dp, 0.0, 0.0);}  

long double b1, b2 = 0.0;
long double f, f1, f2 = 0.0;

long double depth = 2*t+2*rd+dp;
long double l_com, old_y, Ma;

if (pr.y_cg >= depth*0.5)
  {
    l_com = pr.y_cg-rd-t;
    f! = Fy/pr.y_cg*l_com;
    f2 = Fy/pr.y_cg*(dp-l_com);
  }
else
{
 for (int i = 0; i < 10; i++)
{
 f = Fy/(depth-pr.y_cg)*pr.y_cg;
 old_y = pr.y_cg;
 if (sec_type == 1 && cot == 1)
{
 b = eff_width_of_ucse (f, 1);
 }
 else
{
 b = eff_width_of_ucue (f, 1);
 }
 if (cot == 1)
 {pr = section_properties (b, wt, dp, 0.0, 0.0);}
 else
 {pr = section_properties (w, b, dp, 0.0, 0.0);}

 if ((old_y - pr.y_cg) <= 0.001) break;
 }
 l_com = pr.y_cg-rd-t;
 f1 = Fy/(depth-pr.y_cg)*l_com;
 f2 = Fy/(depth-pr.y_cg)*(dp-l_com);
}

 eff_height_of_web (b1, b2, f1, f2, 1);

 if ((b1+b2) >= l_com)
{
 Ma = pr.S*Fy*phi_b;
 return inelastic_reserve_capacity (l_com, Ma);
 }

 old_y = pr.y_cg;

 for (int i = 0; i < 10; i++)
{
 if (cot == 1)
 {pr = section_properties (b, wt, b1, b2, old_y);}
 else
 {pr = section_properties (w, b, b1, b2, old_y);}
 if ((pr.y_cg-old_y) <= 0.001) break;
 if (pr.y_cg >= depth*0.5)
\{ 
  l_com = pr.y_cg-rd-t;
  f1 = Fy/pr.y_cg*l_com;
  f2 = Fy/pr.y_cg*(dp-l_com);
\}
else
\{
  f = Fy/(depth-pr.y_cg)*pr.y_cg;
  if (sec_type == 1 && cot == 1)
    \{ b = eff_width_of.ucse (f, 1); \}
  else
    \{ b = eff_width_of.ucse (f, 1); \}
  l_com = pr.y_cg-rd-t;
  f1 = Fy/(depth-pr.y_cg)*l_com;
  f2 = Fy/(depth-pr.y_cg)*(dp-l_com);
\}

  eff_height_of_web (b1, b2, f1, f2, 1);
  old_y = pr.y_cg;
\}
  Ma = pr.S*Fy*phi_b;
return inelastic_reserve_capacity (l_com, Ma);
\}

// bending strength based on shear lag effect considerations
long double AISI::moment_capacity_shearlag ()
{ 
  Properties pr;
  // find bc and bt based on shear lag considerations
  long double bc = sl_ratio*w;
  long double bt = sl_ratio*wr;
  // properties assuming full web effectiveness
  pr = section_properties (bc, bt, dp, 0.0, 0.0);

  long double b1, b2 = 0.0;
  long double f, f1, f2 = 0.0;

  long double depth = 2*t+2*rd+dp;
  long double l_com, old_y, Ma;

141
if \( (pr.y_{cg} \geq depth \times 0.5) \)
{
    l_{com} = pr.y_{cg} - r_{d} - t;
    f1 = Fy/pr.y_{cg} \times l_{com};
    f2 = Fy/pr.y_{cg} \times (dp - l_{com});
}
else
{
    l_{com} = pr.y_{cg} - r_{d} - t;
    f1 = Fy/(depth - pr.y_{cg}) \times l_{com};
    f2 = Fy/(depth - pr.y_{cg}) \times (dp - l_{com});
}

eff\_height\_of\_web (b1, b2, f1, f2, 1);
if ((b1 + b2) \geq l_{com})
{
    return pr.S \times Fy \times \phi_{b};
}
old\_y = pr.y_{cg};

for (int i = 0; i < 10; i++)
{
    pr = section\_properties (bc, bt, b1, b2, old\_y);
    if ((pr.y_{cg} - old\_y) \leq 0.001) break;
    if (pr.y_{cg} \geq depth \times 0.5)
    {
        l_{com} = pr.y_{cg} - r_{d} - t;
        f1 = Fy/pr.y_{cg} \times l_{com};
        f2 = Fy/pr.y_{cg} \times (dp - l_{com});
    }
    else
    {
        l_{com} = pr.y_{cg} - r_{d} - t;
        f1 = Fy/(depth - pr.y_{cg}) \times l_{com};
        f2 = Fy/(depth - pr.y_{cg}) \times (dp - l_{com});
    }
    eff\_height\_of\_web (b1, b2, f1, f2, 1);
    old\_y = pr.y_{cg};
}
return pr.S \times Fy \times \phi_{b};
// bending strength considering inelastic reserve capacity (partial
// plastification)
long double AIS1::inelastic_reserve_capacity (const long double& _l_com,
    const long double& _M)
{
  long double ey = Fy/E, cy;
  long double lambda_1 = 1.11/sqrt(ey);
  long double lambda_2 = 1.28/sqrt(ey);

  if (!is_braced == TRUE && (_l_com/t) <= lambda_1
      && Vapp <= (0.35*Fy*now*dp*t))
    {
      return _M;
    }

  long double bc = w;
  long double bt = wt;
  if (cot != 1) bc = wt, bt = w;
  if (sec_type == 1 && cot == 1)
    {
      if (bc/t) <= lambda_1)
        {
          cy = 3.0;
        }
      else
        {
          cy = 1.0;
        }
    }
  else
    {
      cy = 1.0;
    }

  long double d, M;
  d = dp+2*rd+t;

  if (sec_type == 1)
```c
} else if (sec_type == 2)
{
    bc = eff_width_of_ucee (Fy, 1);
    bc = bc+2*rd+t;
    bt = bt+2*rd+t;
}
else
{
    bc = eff_width_of_ucee (Fy, 1);
    bc = bc+rd+t*0.5;
    bt = bt+rd+t*0.5;
}

long double A = 2-1/cy-cy;
long double B = bc+2*cy*d+cy*bt;
long double C = cy*d*d+cy*bt*d;
long double yc = (-B+sqrt(B*B+4*A*C))/2*A;
long double yt = d-yc;
long double yp = yc/cy;
long double ycp = yc-yp;
if (yp > yt)
{
    M = Fy*t*(bc*yc+2*ycp*(yp+ycp/2)+2/3*yp*yp+2/3*yt*yt*cy*yt/yc + bt*yt*cy*yt/yc);
}
else
{
    yc = (bt-bc+2*d)/4;
    yt = d-yc;
    yp = yc/cy;
    ycp = yc-yp;
    long double yp = yt-yp;
    M = Fy*t*(bc*yc+2*ycp*(yp+ycp*0.5)+4/3*yp*yp+2*yp*yp*0.5)+bt*yt;
}
if (phi_b*M <= 1.25*_M) return phi_b*M;
return 1.25*_M;
```
// shear strength
long double AISI::shear_capacity ()
{
    long double Va;
    long double kv = 5.34; // webs without transverse stiffeners
    if (LFRD != 1)
    {
        long double par = 1.38*sqrt(E*kv/Fy);
        if (dp/t <= par)
        {
            Va = (0.38*t*t*sqrt(kv*Fy*E));
            if (Va >= 0.4*Fy*dp*t) Va = 0.4*Fy*dp*t;
            return now*Va;
        }
        else
        {
            return Va = now*(0.53*E*kv*t*t*t/dp);
        }
    }
    long double par1 = sqrt(E*kv/Fy);
    if (dp/t <= par1)
    {
        return Va = now*1.0*0.577*Fy*dp*t;
    }
    if (dp/t > par1 && dp/t < (1.415*par1))
    {
        return Va = now*phi_wv*0.64*t*t*sqrt(kv*Fy*E);
    }
    return Va = now*phi_wv*0.905*E*kv*t*t*t/dp;
}

// moment of inertia (I) for deflection calculation
long double AISI::deflection_I (Properties pr)
{
    long double f, f1, f2, l_com, old_y, bd;
    long double b1 = 0;
    long double b2 = 0;
long double depth = pd+2*rd+2*t;

for (int i = 0; i < 10; i++)
{
    l_com = pr.y_cg-t_rd;
    f = Mapp*pr.y_cg/pr.I;
    f1 = Mapp*l_com/pr.I;
    f2 = Mapp*(dp-l_com)/pr.I;
    if (f > Fy) f = Fy, f1 = Fy;
    if (f2 > Fy) f2 = Fy;

    old_y = pr.y_cg;
    if (sec_type == 1)
    {
        bd = eff_width_of_ucse (f, 0);
    }
    else
    {
        bd = eff_width_of_UcE (f, 0);
    }
    eff_height_of_web (b1, b2, f1, f2, 0);
    if ((b1+b2) >= l_com)
    {
        pr = section_properties (bd, wt, dp, 0.0, 0.0);
    }
    else
    {
        pr = section_properties (bd, wt, b1, b2, old_y);
    }
    if ((pr.y_cg-old_y) <= 0.001) break;
}

return pr.I;
}

long double AISI::combined_bending_and_shear(const long double& mom,
    const long double& _Ma, const long double& sh, const long double& _Va)
{
    return (mom/_Ma)*(mom/_Ma) + (sh/_Va)*(sh/_Va);
}
// web crippling strength
long double AISI::web_crippling_strength (int l_condition)
{
    if (LFRD == 1) return web_crippling_strength_LFRD (l_condition);

    long double k = Fy/33;
    long double C1, C2, C3, C4, C5, C6, C7, C8, m;
    if ((sec_type == 1) || (sec_type == 3))
    {
        C1 = 1.22-0.22*k;
        C2 = 1.06-0.06*rd/t;
        if (C2 >= 1.0) C2 = 1.0;
        C3 = 1.33-0.33*k;
        C4 = 1.15-0.15*rd/t;
        if (C4 < 0.5) C4 = 0.5;
        if (C4 >= 1.0) C4 = 1.0;
    }
    if (sec_type == 2)
    {
        C5 = 1.49-0.33*k;
        if (C5 < 0.6) C5 = 0.6;
        if (dp/t <= 150) C6 = 1+(dp/t)/750;
        if (dp/t > 150) C6 = 1.2;
        if (dp/t <= 66.5) C7 = 1/k;
        if (dp/t > 66.5) C7 = (1.1-(dp/t)/665)/k;
        C8 = (0.98-(dp/t)/865)/k;
        m = t/0.075;
    }

    if (sec_type == 1 || sec_type == 3)
    {
        switch (l_condition)
        {
            case 1: //EOF
                {
                    long double fac = 0.71+0.015*N/t;
                    if ((N/t) <= 60.0) fac = 1+0.01*N/t;
                    if (sec_type == 1) return now*t*t*k*C3*C4*(179-0.33*dp/t)*fac;
                    if (sec_type == 3) return now*t*t*k*C3*C4*(117-0.15*dp/t)*fac;
                }
            case 2: //IOF
                {
                    long double fac = 1+0.007*N/t;
                    if ((N/t) > 60.0) fac = 0.75+0.011*N/t;
                }
return now*t*t*k*C1*C2*(291-0.4*dp/t)*fac;
}
case 3:    //ETF
    return now*t*t*k*C3*C4*(132-0.31*dp/t)*(1+0.01*N/t);
case 4:    //ITF
    return now*t*t*k*C1*C2*(417-1.22*dp/t)*(1+0.0013*N/t);
default:
    cout << "Error in web crippling!";
    exit (1);
}
}

if (sec_type == 2)
{
switch (l_condition)
{
    case 1:
    {
        long double fac = 1+0.007*N/t;
        if (N/t > 60) fac = 0.75+0.011*N/t;
        return now*t*t*Fy*C6*(5+0.63*sqrt(N/t));
    }
    case 2:
        return now*t*t*Fy*C5*(0.88+0.12*m)*(7.5+1.63*sqrt(N/t));
    case 3:
        return now*t*t*Fy*C8*(0.64+0.31*m)*(5+0.63*sqrt(N/t));
    case 4:
        return now*t*t*Fy*C7*(0.82+0.15*m)*(7.5+1.63*sqrt(N/t));
default:
    cout << "Error in web crippling!";
    exit (1);
}
}

// web crippling strength. For LFRD only
long double AISI::web_crippling_strength_LFRD (int l_condition)
{
    long double k = Fy/33;
    long double C1, C2, C3, C4, C5, C6, C7, C8, m;
    if (sec_type == 1 || sec_type == 3)
    {
        C1 = 1.22-0.22*k;
        }
C2 = 1.06-0.06*rd/t;
if (C2 >= 1.0) C2 = 1.0;
C3 = 1.33-0.33*k;
C4 = 1.15-0.15*rd/t;
if (C4 < 0.5) C4 = 0.5;
if (C4 >= 1.0) C4 = 1.0;

if (sec_type == 2)
{
C5 = 1.49-0.53*k;
if (C5 < 0.6) C5 = 0.6;
C6 = 1+(dp/t)/750;
if (dp/t > 150) C6 = 1.2;
C7 = 1/k;
if (dp/t > 66.5) C7 = (1.1-(dp/t)/665)/k;
C8 = (0.98-(dp/t)/865)/k;
m = t/0.075;
}

if (sec_type == 1 || sec_type == 3)
{
switch (l_condition)
{
case 1:  //EOF
{
    long double fac = 0.71+0.015*N/t;
    if (((N/t) <= 60.0)) fac = 1+0.01*N/t;
    if (sec_type == 1) return phi_wc*now*t*t*k*C3*C4*(331-0.61*dp/t)*fac;
    if (sec_type == 3) return phi_wc*now*t*t*k*C3*C4*(217-0.28*dp/t)*fac;
}
case 2:  //IOF
{
    long double fac = 1+0.007*N/t;
    if ((N/t) > 60.0) fac = 0.75+0.011*N/t;
    return phi_wc*now*t*t*k*C1*C2*(538-0.74*dp/t)*fac;
}
case 3:  //ETF
    return phi_wc*now*t*t*k*C3*C4*(244-0.57*dp/t)*(1+0.01*N/t);
case 4:  //TF
    return now*t*t*k*C1*C2*(771-2.26*dp/t)*(1+0.0013*N/t);
default:
    cout << "Error in web crippling!";
    exit (1);
}
if (sec_type == 2)
{
    switch (l_condition)
    {
    case 1:
    {
        long double fac = 1+0.007*N/t;
        if (N/t > 60) fac = 0.75+0.011*N/t;
        return phi_wc*nrow*t*t*Fy*C6*(10+1.25*sqrt(N/t));
    }
    case 2:
    return phi_wc*nrow*t*t*Fy*C5*(0.88+0.12*m)*(15+3.25*sqrt(N/t));
    case 3:
    return phi_wc*nrow*t*t*Fy*C8*(0.64+0.31*m)*(10+1.25*sqrt(N/t));
    case 4:
    return phi_wc*nrow*t*t*Fy*C7*(0.82+0.15*m)*(15+3.25*sqrt(N/t));
    default:
        cout << "Error in web crippling!";
        exit (1);
    }
}

void AISI::combined_bending_and_crippling (long double& _pf,
    long double& _mf)
{
    if (LFRD == 1)
    {
        if (sec_type == 1 || sec_type == 3)
        {
            _pf = 0.753521;
            _mf = 0.704225;
        }
    if (sec_type == 2)
    {
        _pf = 0.621212;
        _mf = 0.757575;
    }
    }
else

150
```c
if (sec_type == 1 || sec_type == 3)
{
    _pf = 0.8;
    _mf = 0.66667;
}
if (sec_type == 2)
{
    _pf = 0.733333;
    _mf = 0.666667;
}
}

// calculate limiting flange width for curling
long double AISI::flange_curling_limitation ()
{
    long double fav;
    long double d = 2*t + 2*rd + dp;
    if (sec_type == 1)
    {
        fav = Fy*eff_width_of_ucse (Fy, 1)*phi_b/(w);
    }
    else
    {
        fav = Fy*eff_width_of_ucse (Fy, 1)*phi_b/w;
    }
    return sqrt(0.061*t*d*E/fav)*sqrt(sqrt(100*cf/d));
}

long double AISI::fl_slenderness_ratio ()
{
    if (sec_type == 1 && cot == 1) return 500;
    return 60;
}

long double AISI::web_slenderness_ratio ()
{
    return 200;
}
```
/ shear lag effect calculation of effective flange width
long double AISI::shear_lag ()
{
    long double wf;
    if (wt >= w)
        { wf = wt; }
    else
        { wf = w; }
    if (sec_type == 1)
        {
            wf = wf*0.5+rd;
        }
    else
        {
            wf = wf+rd;
        }
    if (bw_loads_dist < 2*wf) return 1.0;
}

long double Lwf[] = {30. 25, 20, 18, 16, 14, 12, 10, 8, 6};
long double ratio[] = {1, 0.96, 0.91, 0.89, 0.86, 0.82, 0.78, 0.73, 0.67, 0.55};

long double par = L/wf;
if (par > Lwf[0]) return 1.0;

for (int i = 1; i < 10; i++)
{
    if (par > Lwf[i] && par <= Lwf[i-1]) break;
    if (i == 9) cout << "Error in shear lag!", exit (0);
}
return ratio[i]+(par-Lwf[i])/(Lwf[i-1]-Lwf[i])*(ratio[i-1]-ratio[i]);

B.8 nd.H

Same as Section A.4 (page 91).

B.9 nd.C

Same as Section A.5 (page 93).
B.10  absolute.H

Same as Section A.7 (page 100).

B.11  prop.H

#ifndef _PROP_H_
define _PROP_H_

#define PI 3.141592654L
#define TRUE 1
#define FALSE 0

// Properties declare the important cross-section properties of a shape
struct Properties
{
    long double A;   // area
    long double I;   // moment of inertia about x-axis
    long double S;   // section modulus Sx
    long double Sc;  // section modulus in compression
    long double y_cg; // distance to c.g from outermost compression fiber

};

#endif