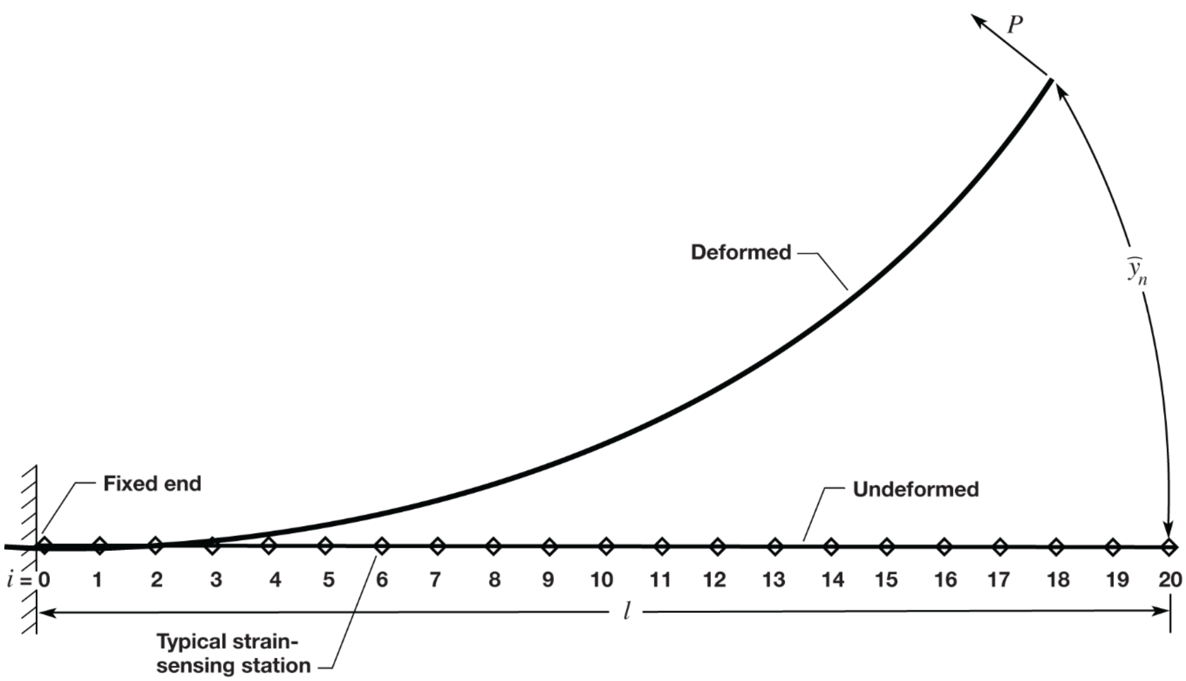
NASA/TM–2019–220221

Structure Deformation Calculation Program Based on Displacement Theory for Shape Predictions

Van Tran Fleischer, and William L. Ko
Armstrong Flight Research Center
Edwards, California 93523



PATENT PROTECTION NOTICE

The method for structure deformed shape predictions using *Displacement Theory* to transform distributed surface strains into structure deformed shapes described in this NASA technical report is protected under *Method for Real-Time Structure-Shape Sensing*, U.S. Patent No. 7,520,176, issued April 21, 2009. Therefore, those interested in using the method (with the accompanying program) should contact NASA Technology Transfer Office at NASA Armstrong Flight Research Center, Edwards, California for more information.

August 2019

NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM–2019–220221

Structure Deformation Calculation Program Based on Displacement Theory for Shape Predictions

Van Tran Fleischer, and William L. Ko
Armstrong Flight Research Center
Edwards, California 93523

National Aeronautics and
Space Administration

Armstrong Flight Research Center
Edwards, California 93523-0273

August 2019

Table of Contents

Abstract

Separated programs were written in C/C++ to validate the Displacement Transfer Functions. The Structure Deformation Calculation Program was written to combine all of the programs to calculate deformed shapes of a structure using surface strain data and structural geometrical parameters. Users do not need to know the material properties, nor the complex internal structures geometry because the Displacement Theory is purely geometrical in nature. Users only need to know the structure types as defined in this report and information such as the structure length, depth factors, number of strain sensors, and the surface strains measured at the strain-sensing stations installed on the structures. Depending on the structure type, an applicable Displacement Transfer Function will be used. This program requires two input files created by users; the recorded strain data file in comma-separated values format and the structure geometry data file in text format. The program will output the out-of-plane deflections, slopes, cross-sectional twist angles, and depth factors if applicable. All output files are created in comma-separated values format. A section in this report describes step-by-step procedures on how to use the Structure Deformation Calculation Program for structure deformed shape calculations.

Nomenclature

c	constant depth factor (vertical distance from the neutral axis to the lower surface of the uniform embedded beam), in.
c_i	lower depth factor at $x = x_i$ (distance from the embedded beam neutral axis to the *i*-th strain-sensing station on the lower surface of the embedded beam), in.
$\bar{c}_i$	upper depth factor at $x = x_i$ (distance from the embedded beam neutral axis to the *i*-th strain-sensing station on the upper surface of the embedded beam, in.
c_n	value of c_i at free end, $x = x_n = l$, in.
c_0	value of c_i at fixed end, $x = x_0 = 0$, in.
csv	comma-separated values format
d_i	chord-wise separation distance of two strain-sensing lines at $x = x_i$, in.
d_n	value of d_i at wing tip, $x = x_n = l$, in.
d_0	value of d_i at wing root, $x = x_0 = 0$, in.
h_i	depth of the front embedded beam at $x = x_i$, in.
h_n	value of h_i at free end, $x = x_n = l$, in.
h_0	value of h_i at fixed end, $x = x_0 = 0$, in.
l	length of an embedded beam, in.
n	number of domains or index for the last span-wise strain-sensing station
NASA	National Aeronautics and Space Administration
P	applied load, lb
SG	strain gauge
txt	text format
w_n	wing tip chord length (width), in.
w_0	wing root chord length (width), in.
x, y	Cartesian coordinates (x in axial direction, y in lateral direction), in.
x_i	axial coordinate of *i*-th strain sensor, in.
x_n	axial coordinate at wing tip x $= x_n = l$, in.
y_i	vertical deflection at $x = x_i$, in.
$\widehat{y}_i$	curved deflection at $x = x_i$, in.

y_n	vertical deflection at $x = x_n = l$, in.
$\hat{y}_n$	curved deflection at $x = x_n = l$, in.
y_i^B	vertical deflection at $x = x_i$ of a two-end supported embedded beam, in.
$\hat{y}_i^B$	curved deflection at $x = x_i$ of a two-end supported embedded beam, in.
$(\Delta l)_i$	$= \Delta l_i \equiv (x_i - x_{i-1})$, *i*-th domain length (distance between two adjacent strain-sensing stations), in.
Δl_n	domain length at the tip, $x_n - x_{n-1}$, in.
ε_i	lower surface bending strain at strain-sensing station *i*, in/in
$\bar{\varepsilon}_i$	upper surface bending strain at strain-sensing station *i*, in/in
ε_n	lower surface bending strain at the tip, strain-sensing station *n*, in/in
ε_0	lower surface bending strain at the root, strain-sensing station *0*, in/in
$\varepsilon(x)$	surface bending strain at axial location *x*, in/in
θ_i	slope angle of a cantilever embedded beam at $x = x_i$, rad
θ_n	slope angle of a cantilever embedded beam at $x = x_n = l$ (free end), rad
θ_0	slope angle of a cantilever embedded beam at $x = x_0 = 0$ (fixed end), rad
ϕ_i	cross-sectional twist angle of a cantilever embedded beam at $x = x_i$, rad
$(\,)'$	quantity associated with the rear strain-sensing lines

Introduction

Traditionally, the wing deflections can be measured during ground testing by using position transducers or a photogrammetry system. For in-flight deflection measurements, those methods cannot be used. One technique is to use the electro-optical flight deflection measurement systems, which are composed of on-board cameras and several wing mounted targets. Such systems can provide wing deflection information during the flight, but can be too heavy for lightweight flying vehicle applications.

After the invention of the Displacement Theory which contains different Displacement Transfer Functions (refs. 1–12), a patented technology called, "Method for Real-Time Structure Shape-Sensing," U.S. Patent Number 7,520,176 (ref. 2), was granted. The shape-sensing technology is to use the Displacement Transfer Functions to transform distributed surface strains into structure deformed shapes. This structure shape-sensing technology is quite attractive for the in-flight deformed shape monitoring of flight vehicles for flight control and maintaining flight safety. In addition, the real time wing shape monitored could then be input to the aircraft control system for aero-elastic wing shape control.

The objective of this technical memorandum is to provide users some guidance on how to use the Structure Deformation Calculation Program to calculate the deformed shape of a structure based on the Displacement Theory and Displacement Transfer Functions (refs. 1–12). Users need to prepare two files, the recorded measured surface strain data in a comma-separated values (csv) file and the required geometrical information in a text (txt) file. Depending on the structure type, the program will create several csv output files that contain the out-of-plane deflections y_i, slopes θ_i, cross-sectional twist angles ϕ_i, and depth factors c_i if applicable.

Shape Prediction Technical Background

The structure shape prediction using the Displacement Transfer Functions to transform the distributed surface strains into structure deformed shapes was reported in many National Aeronautic and Space Administration (NASA) technical reports (refs. 1–12). The following sections only cover what are related to the Structure Deformation Calculation Program. To understand more about the Displacement Theory and Displacement Transfer Functions, users can read the NASA technical reports listed in the reference section.

Key Terminologies

A surface line, along which the strain-sensing stations are to be discretely distributed, is called a *strain-sensing line*. The surface strains are to be measured at those strain-sensing stations and recorded. The region between any two adjacent strain-sensing stations is called the *domain*. The structure depth-wise cross section along the strain-sensing line is called the *embedded beam* (not to be confused with the traditional isolated Euler-Bernoulli beam). The distances from the embedded beam neutral axis to the strain-sensing stations along the lower strain-sensing line are called the *depth factors*. When the data of bending surface strains, domain lengths, depth factors, and number of strain-sensing stations are input into the appropriate Displacement Transfer Functions, the deformed shape of each embedded beam can be calculated.

Theoretical Background

In the formulations of the Displacement Transfer Functions (refs. 1–12), each embedded beam was first discretized into multiple small domains with domain junctures matching the strain-sensing stations. Such a discretization approach allowed the surface strain distribution along each strain-sensing line to be represented with a piecewise-linear function. The piecewise-linear approach enabled piecewise integrations of the embedded-beam curvature equation to yield the Displacement Transfer Functions, which geometrically relate the surface strains to the out-of plane deflections along the embedded beam.

For structure shape calculations using the Displacement Transfer Functions, surface strain data and depth factors of an embedded beam are needed. Based on the type of structure geometry and loading conditions, users can select the proper strain-sensing line system and structure type for their structures. If the depth factors are unknown, extra strain-sensing line(s) is/are required. The Structure Deformation Calculation Program covers seven structure types that have depth factors known and depth factors unknown.

Depth Factors

The depth factors of a structure are important variables in Displacement Transfer Functions. The depth factors c_i, along with strains ε_i and domain lengths Δl_i, are used in the calculations of the vertical deflections y_i, slopes θ_i, and cross-sectional twist angles ϕ_i if applicable at the *i*-th strain-sensing location. For some structures, it is difficult to know the depth factors; therefore, extra strain-sensing lines are needed.

Depth Factors Known

Structure types 1 and 6 for a one-line system applied to a cantilever beam are shown in the type 1 and type 6 sections of this report. Structure type 2 for a two-end supported tubular beam is shown in the type 2 section. Since the depth factors are known, only one strain-sensing line on the lower surface is needed for bending shape prediction analysis.

Structure type 3 for a two-line system on the lower surface for combined bending and torsion or on the side and lower surfaces for combined horizontal and vertical bending is shown in the type 3 section. The two-line system includes tapered un-swept and swept wing boxes. If the depth factor is known, only two strain-sensing lines along the lower front and lower rear edges are needed. For this type of structure, the local cross-sectional twist angles can be calculated.

Structure type 5 for a square thin plate (finite-element model) subjected to a point load at the plate center, inducing two-dimensional bending under different edge conditions (four edges clamped or simply supported) is shown in the type 5 section. Because the depth factors are known, only the multi parallel strain-sensing lines on the lower surface are needed.

Depth Factors Unknown

Structure type 4 of a four-line system with two lines on the lower surface and two lines on the upper surface for shape calculations of structures under combined bending and torsion is shown in the type 4 section. The four-line system is the most suitable sensing system for slender aircraft wings, for which the two neutral axes are unknown and are always subjected to both bending and torsion loadings. Two upper strain-sensing lines are needed for calculations of unknown depth factors. If the depth factors are known, the upper surface lines are not required.

The depths at the beam root and beam tip $\{h_0, h_n\}$ at the front of the embedded beam are known, and the local depth h_i can be calculated as shown in equation (1a).

$$h_i = h_0 - (h_0 - h_n)\frac{x_i}{l} \qquad ; \quad (i = 1,2,3,\ldots,n) \qquad \text{(1a)}$$

The depths at the beam root and beam tip $\{h'_0, h'_n\}$ at the rear of the embedded beam are known, and the local depth h'_i can be calculated as shown in equation (1b).

$$h'_i = h'_0 - (h'_0 - h'_n)\frac{x_i}{l} \qquad ; \quad (i = 1,2,3,\ldots,n) \qquad \text{(1b)}$$

The values of calculated h_i and the bending strains $\{\varepsilon_i, \bar{\varepsilon}_i\}$ where $\bar{\varepsilon}_i$ are the bending strains of the front upper surface are used to calculate c_i as shown in equation (2a).

$$c_i = \frac{|\,\varepsilon_i\,|}{|\,\varepsilon_i\,| + |\,\bar{\varepsilon}_i\,|} h_i\,; \qquad \bar{c}_i = h_i - c_i \qquad ; \quad (i = 1,2,3,\ldots,n) \qquad \text{(2a)}$$

The values of calculated h'_i at the rear and the bending strains $\{\varepsilon'_i, \bar{\varepsilon}'_i\}$ where $\bar{\varepsilon}'_i$ are the bending strains of the rear upper surface are used to calculate c'_i as shown in equation (2b).

$$c'_i = \frac{|\,\varepsilon'_i\,|}{|\,\varepsilon'_i\,| + |\,\bar{\varepsilon}'_i\,|} h'_i\,; \qquad \bar{c}'_i = h'_i - c'_i \qquad ; \quad (i = 1,2,3,\ldots,n) \qquad \text{(2b)}$$

For a nonuniform large bending structure of a two-line system on lower and upper surfaces, an extra upper strain-sensing line is required to calculate the depth factors as shown in type 7 section.

List of the Shifted Displacement Transfer Functions

Based on the piecewise-linear representations of both depth factor c_i and surface strain ε_i where $i = 1, 2, 3, \ldots, n$, the Shifted Displacement Transfer Functions (refs. 1, 3) were formulated to transform the surface strains ε_i into slopes and vertical deflections $\{tan\theta_i,\ y_i\}$ along the embedded beam. The Shifted

Displacement Transfer Functions for vertical deflections have the following different mathematical forms formulated for different types of structures (nonuniform, slightly nonuniform, and uniform).

Vertical Deflection for Cantilever Embedded Beams

There are three Shifted Displacement Transfer Functions for a cantilever embedded beam where $(y_0 = \ tan\theta_0 = 0)$.

Nonuniform Shifted Displacement Transfer Functions

The depth factors are not equal $(c_{i-1} \neq c_i)$, (refs. 1, 3). The slope equation (in recursive form) is shown as equation (3a):

$$\tan\theta_i = (\Delta l)_i \left[\frac{\varepsilon_{i-1} - \varepsilon_i}{c_{i-1} - c_i} + \frac{\varepsilon_{i-1}c_i - \varepsilon_i c_{i-1}}{(c_{i-1} - c_i)^2} \log\frac{c_i}{c_{i-1}} \right] + \tan\theta_{i-1} \quad ; \quad (i = 1,2,3, \ldots, n) \tag{3a}$$

The vertical deflection equation (in recursive form) is shown as equation (3b):

$$y_i = (\Delta l)_i^2 \left[\frac{\varepsilon_{i-1} - \varepsilon_i}{2(c_{i-1} - c_i)} - \frac{\varepsilon_{i-1}c_i - \varepsilon_i c_{i-1}}{(c_{i-1} - c_i)^3} \left(c_i \log\frac{c_i}{c_{i-1}} + (c_{i-1} - c_i) \right) \right] + y_{i-1} + (\Delta l)_i \tan\theta_{i-1} \quad ; \quad (i = 1,2,3, \ldots, n) \tag{3b}$$

Equations (3a) and (3b) are used for structure types 1, 3, and 4.

Slightly Nonuniform Shifted Displacement Transfer Functions

The depth factors are almost equal $(c_{i\text{-}1} \to c_i)$, (refs. 1, 3). The slope equation (in recursive form) is shown as equation (4a):

$$\tan\theta_i = \frac{(\Delta l)_i}{2c_{i-1}} \left[\left(2 - \frac{c_i}{c_{i-1}} \right) \varepsilon_{i-1} + \varepsilon_i \right] + \tan\theta_{i-1} \quad ; \quad (i = 1,2,3, \ldots, n) \tag{4a}$$

The vertical deflection equation (in recursive form) is shown as equation (4b):

$$y_i = \frac{(\Delta l)_i^2}{6c_{i-1}} \left[\left(3 - \frac{c_i}{c_{i-1}} \right) \varepsilon_{i-1} + \varepsilon_i \right] + y_{i-1} + (\Delta l)_i \tan\theta_{i-1} \quad ; \quad (i = 1,2,3, \ldots, n) \tag{4b}$$

Equations (4a) and (4b) are used for structure types 1, 3, and 4.

Uniform Shifted Displacement Transfer Functions

The depth factors are equal $(c_{i\text{-}1} = c_i = c)$, (ref. 1). The slope equation (in recursive form) is shown as equation (5a):

$$\tan\theta_i = \frac{(\Delta l)_i}{2c}(\varepsilon_{i-1} + \varepsilon_i) + \tan\theta_{i-1} \quad ; \quad (i = 1,2,3,\dots,n) \tag{5a}$$

The vertical deflection equation (in recursive form) is shown as equation (5b):

$$y_i = \frac{(\Delta l)_i^2}{6c}(2\varepsilon_{i-1} + \varepsilon_i) + y_{i-1} + (\Delta l)_i \tan\theta_{i-1} \quad ; \quad (i = 1,2,3,\dots,n) \tag{5b}$$

Vertical Deflection for Two-End Supported Embedded Beams

The vertical deflection y_i^B of the two-end supported embedded beam (simply supported or fixed) can be calculated from equation (6) (ref. 1):

$$y_i^B = y_i - \underbrace{\frac{x_i}{l} y_n}_{\text{Shift factor}}$$
$$= \underbrace{\frac{1}{6}\sum_{j=1}^{i} \frac{(\Delta l)_j^2}{c_{i-j}} \left\{ \left[3(2j-1) - (3j-2)\frac{c_{i-j+1}}{c_{i-j}} \right] \varepsilon_{i-j} + (3j-2)\varepsilon_{i-j+1} \right\}} - \frac{x_i}{l} y_n \quad ; \quad (i = 1,2,3,\dots,n) \tag{6}$$

In equation (6), y_i is the vertical deflection of the slightly nonuniform cantilever embedded beam (applicable to the limit case of uniform embedded beams). The mathematical expression of y_i in equation (6) was obtained by combining the slope equation (4a) and the deflection equation (4b) into a single equation. The shift factor $(x_i/l)y_n$ appearing in equation (6) is to proportionally shift the cantilever deflection curve of y_i and convert it to the deflection curve of the two-end supported beam with zero deflection $y_i^B = 0$ at the beam tip $i = n$ (second support point).

List of Curved Displacement Transfer Functions

For large bending deformations of highly flexible slender structures, one must understand that the actual (true) deflection $\widehat{y}_i$ of a material point at $x = x_i$ is a curved distance traced by the same material point from its initial un-deformed position to its final deformed position. Thus, the conventional vertical deflection y_i is merely the vertical component of the curved true deflection $\widehat{y}_i$ (refs. 3, 11). The Curved Displacement Transfer Functions have the following different mathematical forms for different types of structures (nonuniform, slightly nonuniform, and uniform).

Large Deflection for Cantilever Embedded Beam

Just like the small bending deformations, the large bending deformations have three Curved Displacement Transfer Functions for a cantilever embedded beam for which $\widehat{y}_i = \theta_0 = 0$.

Nonuniform Curved Displacement Transfer Functions

The depth factors are not equal $(c_{i-1} \neq c_i)$, (ref. 11). The slope equation (in recursive form) is shown in equation (7a):

$$\theta_i = (\Delta l)_i \left[\frac{\varepsilon_{i-1} - \varepsilon_i}{c_{i-1} - c_i} + \frac{\varepsilon_{i-1} c_i - \varepsilon_i c_{i-1}}{(c_{i-1} - c_i)^2} \log \frac{c_i}{c_{i-1}} \right] + \theta_{i-1} \tag{7a}$$

$; \quad (i = 1,2,3, \dots, n)$

The curved deflection equation (in recursive form) is shown in equation (7b):

$$\widehat{y}_i = (\Delta l)_i^2 \left[\frac{\varepsilon_{i-1} - \varepsilon_i}{2(c_{i-1} - c_i)} - \frac{\varepsilon_{i-1} c_i - \varepsilon_i c_{i-1}}{(c_{i-1} - c_i)^3} \left(c_i \log \frac{c_i}{c_{i-1}} + (c_{i-1} - c_i) \right) \right] + \widehat{y}_{i-1} + (\Delta l)_i \theta_{i-1} \tag{7b}$$

$; \quad (i = 1,2,3, \dots, n)$

Slightly Nonuniform Curved Displacement Transfer Functions

The depth factors are almost equal $(c_{i-1} \to c_i)$, (ref. 3). The slope equation (in recursive form) is shown in equation (8a):

$$\theta_i = \frac{(\Delta l)_i}{2 c_{i-1}} \left[\left(2 - \frac{c_i}{c_{i-1}} \right) \varepsilon_{i-1} + \varepsilon_i \right] + \theta_{i-1} \tag{8a}$$

$; \quad (i = 1,2,3, \dots, n)$

The curved deflection equation (in recursive form) is shown in equation (8b):

$$\widehat{y}_i = \frac{(\Delta l)_i^2}{6 c_{i-1}} \left[\left(3 - \frac{c_i}{c_{i-1}} \right) \varepsilon_{i-1} + \varepsilon_i \right] + \widehat{y}_{i-1} + (\Delta l)_i \theta_{i-1} \tag{8b}$$

$; \quad (i = 1,2,3, \dots, n)$

Uniform Curved Displacement Transfer Functions

The depth factors are equal $(c_{i-1} = c_i = c)$, (ref. 11). The slope equation (in recursive form) is shown in equation (9a):

$$\theta_i = \frac{(\Delta l)_i}{2c}(\varepsilon_{i-1} + \varepsilon_i) + \theta_{i-1} \quad ; \quad (i = 1,2,3,\dots,n) \tag{9a}$$

The curved deflection equation (in recursive form) is shown in equation (9b):

$$\widehat{y_i} = \frac{(\Delta l)_i^2}{6c}(2\varepsilon_{i-1} + \varepsilon_i) + \widehat{y_{i-1}} + (\Delta l)_i \theta_{i-1} \quad ; \quad (i = 1,2,3,\dots,n) \tag{9b}$$

Large Deflection for Two-end Supported Embedded Beam

The curved deflection $\widehat{y}_i^B$ of the two-end supported embedded beam (simply supported or fixed) can be calculated from equation (10), which enforces zero deflection at the beam tip ($i = n$) of the cantilever embedded beam using shifting factor $(x_i/l)\widehat{y_n}$ (ref. 1):

$$\begin{aligned} \widehat{y}_i^B &= \widehat{y}_i - \frac{x_i}{l}\widehat{y}_n \\ &= \frac{1}{6}\sum_{j=1}^{i}\frac{(\Delta l)_j^2}{c_{j-1}}\left\{\left[3(2j-1) - (3j-2)\frac{c_{i-j+1}}{c_{i-j}}\right]\varepsilon_{i-j} + (3j-2)\varepsilon_{i-j+1}\right\} - \frac{x_i}{l}\widehat{y}_n \end{aligned} \tag{10}$$

$$; \quad (i = 1,2,3,\dots,n)$$

In equation (10), $\widehat{y}_i^B$ is the curved deflection of a slightly nonuniform cantilever embedded beam. Equation (10) was obtained by combining the slope angle equation (8a) and the curved deflection equation (8b) into a single equation, and is applicable to the limit case of uniform embedded beams.

It is important to mention that, if $\{\tan q_i, y_i\}$ in equations (3) − (5) are replaced respectively with $\{\theta_i, \widehat{y}_i\}$, then equations (3) − (5) become equations (7) − (9) for the shape calculations of structures under geometrical nonlinear large deformations (ref. 11).

Cross-Sectional Twist Angle

For structure types 3 and 4 that have front and rear strain-sensing lines, the cross-sectional twist angle at the strain-sensing station i, $x = x_i$, is calculated using equation (11).

$$\phi_i = \sin^{-1}\left\{\frac{y_i - y_i'}{d_i} \text{ or } \frac{\widehat{y_i} - \widehat{y_i'}}{d_i}\right\} \quad ; \quad (i = 1,2,3,\dots,n) \tag{11}$$

Procedure to Use the Program

In order to use the Structure Deformation Calculation Program, users are required to have Microsoft Visual Studio software (Microsoft Corporation, Redmond, Washington) or any server that can compile C/C++ to compile this program. Users need to create two input files, a strain data file and a geometry file in the required formats. The arrangements of the data in these files are different depending on the structure type as defined in the next section.

Preparation of the Strain Data File

The strain data file must be in csv format with the first line containing the header of "time" and names of the strain-sensing stations. The second line to the last line should contain the time and the measured surface strains ε_i of each strain-sensing station i on each strain-sensing line from the fixed end ε_0 to the free end ε_n. The time format in this file will be copied to the output files. If there are multiple strain-sensing lines, strain data on one line must finish before starting strain data on the next line.

Preparation of the Geometry file

The geometry file must be prepared in txt format with spaces or tab delimiters between two values. This file contains the geometry data that the program will use to calculate deformations. The distances in this report are measured in inches, but users can use any units they want as long as they are consistent. This file has some or all of the following elements.

1. Structure type from 1 to 7.
2. Total length of the structure in inches l.
3. Domain length in inches. The domain length, $(\Delta l)_i = \Delta l_i$, is the distance between two adjacent strain sensors i-1 and i on a strain-sensing line; Δl_i can be constant or variable.
4. Total number of strain-sensing stations installed on the structure. If the structure has multiple strain-sensing lines, the number of stations installed on each strain-sensing line must be the same. The domain lengths Δl_i between two adjacent sensors i-1 and i on each strain-sensing line must also be the same; for example, Δl_i on line 1 = Δl_i on line 2 = Δl_i on line k: $\Delta l_{1i} = \Delta l_{2i} = \Delta l_{ki}$.
5. Depth factors in inches (c_i can be known or unknown).
6. Chord-wise distances in inches (d_i for structures that have front and rear strain-sensing lines).
7. Depths in inches (h_0 at the fixed end and h_n at the free end for structures that have lower and upper strain-sensing lines).

Running the program

After creating two required input files, users can run this the Structure Deformation Calculation Program. The program will prompt the user for three following inputs.

$ Enter strain data filename:
$ Enter geometry filename:
$ Enter structure type:

Users must enter inputs to the above prompts in order to run the program. The program will always calculate vertical deflections y_i, slopes θ_i, and determine the maximum and minimum deflections for each strain-sensing station. Different structure configurations in the formulations of the Displacement Theory and Displacement Transfer Functions (refs. 1-12) are categorized into seven structure types in this program.

Depending on the structure type, the program will also calculate the depth factors c_n and/or the cross-sectional twist angles ϕ_i.

Output files created by the program

The program will use the name of the strain data file to create the names of the output files in csv format by appending it with _Deflections, _Slopes, _Deflections_MaxMin, _DepthFactors, and _TwistedAngles. For example, if the strain data filename is N13.csv, the output files are N13_Deflections.csv, N13_Slopes.csv, N13_Deflections_MaxMin.csv, N13_DepthFactors.csv, and N13_TwistAngles.csv. The first row in the deflection file and slope file is labeled exactly the same as the first row in the strain data file. The first column in the deflection file and slope file is exactly the same as the first column in the strain data file. All structure types will have the deflection and maximum minimum deflection files. The deflections are measured in inches and the slopes and twist angles are measured in degrees. When finishing, the program will print out a complete message and also the names of the output files.

Structure Types

With the intention to make the Structure Deformation Calculation Program easy to use, one-line, two-line, and four-line systems, with known and unknown depth factors, with vertical and curved deflections, and with short and long lengths are categorized into seven structure types. The structure types cover the range from the simplest one-line uniform cantilever beam with known depth factors to a complicated four-line doubly tapered wing with unknown depth factors. Each structure type requires different formats of the strain and geometry files and has different output files. Dependent on the structure type, a correct Transfer Function is used in the program to calculate deflections, slopes, cross-sectional twist angles, and depth factors if applicable.

Type 1 – Cantilever Embedded Beam

For a cantilever embedded beam with strain-sensing stations distributed along the bottom strain-sensing line, the depth factors are known, and no torsion is involved. The one-line system can be used for shape prediction analysis. The cantilever embedded beam is the simplest structure type.

Type 1 Structure

Figure 1(a) shows a uniform cantilever beam with $c_0 = c_n$, and figure 1(b) shows a tapered cantilever or nonuniform beam with $c_0 > c_n$.

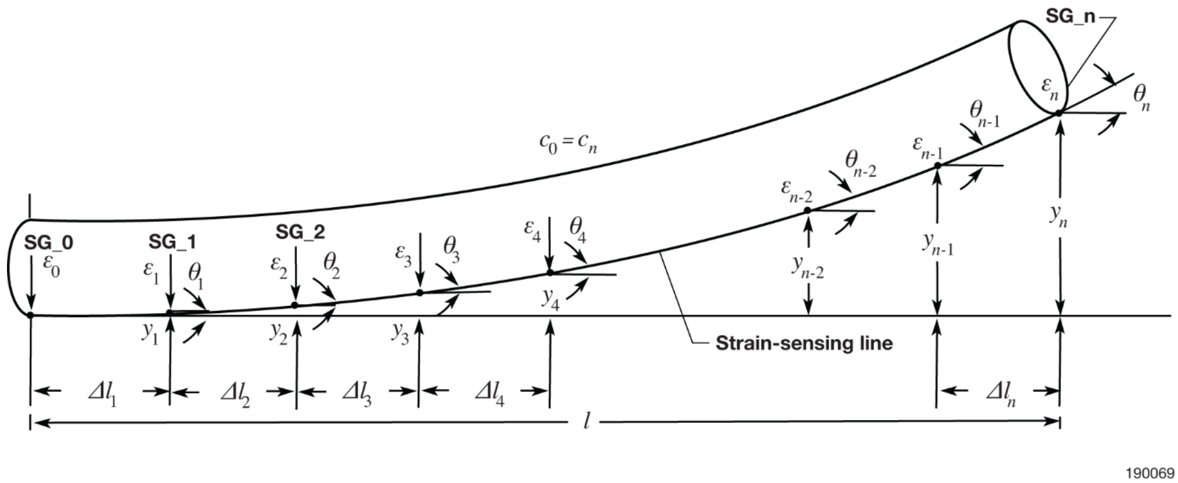


Figure 1(a). Type 1 structure of a uniform cantilever beam.

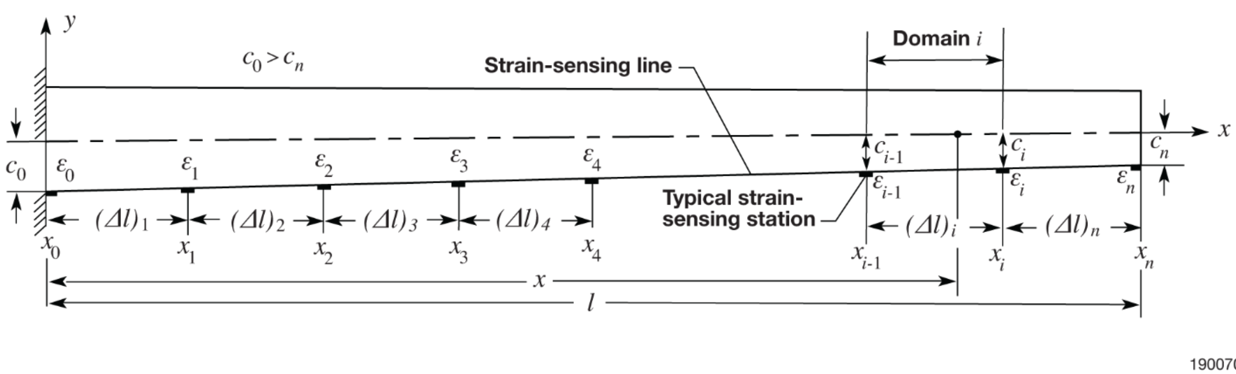
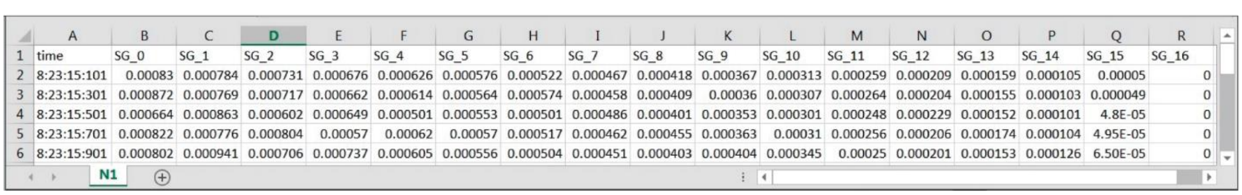


Figure 1(b). Type 1 structure of a tapered cantilever or nonuniform beam.

Type 1 Strain Data File

For type 1 strain data file, recorded strains must be arranged as shown in figure 1(c). The SG_0 is always the strain-sensing station at the fixed end, and SG_n is always the strain-sensing station at the free-end. In figure 1(c), SG_n is SG_16.

- The first line is the header containing the title "time" and names of the strain-sensing station starting from the fixed end.
- The second line to the last line must contain the times and measured strains at stations SG_0, SG_1, …, SG_n.
- The first column contains the times that can be in any time format.
- The columns after the first column contain the measured strains at stations SG_0, SG_1, …, SG_n.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	time	SG_0	SG_1	SG_2	SG_3	SG_4	SG_5	SG_6	SG_7	SG_8	SG_9	SG_10	SG_11	SG_12	SG_13	SG_14	SG_15	SG_16
2	8:23:15:101	0.00083	0.000784	0.000731	0.000676	0.000626	0.000576	0.000522	0.000467	0.000418	0.000367	0.000313	0.000259	0.000209	0.000159	0.000105	0.00005	0
3	8:23:15:301	0.000872	0.000769	0.000717	0.000662	0.000614	0.000564	0.000574	0.000458	0.000409	0.00036	0.000307	0.000264	0.000204	0.000155	0.000103	0.000049	0
4	8:23:15:501	0.000664	0.000863	0.000602	0.000649	0.000501	0.000553	0.000501	0.000486	0.000401	0.000353	0.000301	0.000248	0.000229	0.000152	0.000101	4.8E-05	0
5	8:23:15:701	0.000822	0.000776	0.000804	0.00057	0.00062	0.00057	0.000517	0.000462	0.000455	0.000363	0.00031	0.000256	0.000206	0.000174	0.000104	4.95E-05	0
6	8:23:15:901	0.000802	0.000941	0.000706	0.000737	0.000605	0.000556	0.000504	0.000451	0.000403	0.000404	0.000345	0.00025	0.000201	0.000153	0.000126	6.50E-05	0

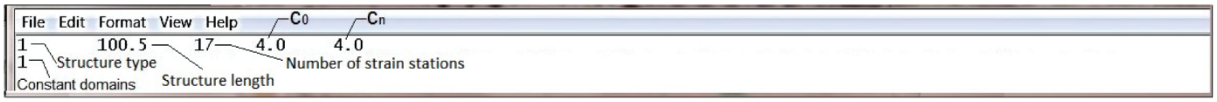
N1

190071

Figure 1(c). Type 1 strain data file.

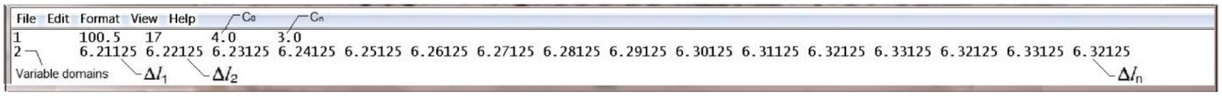
Type 1 Geometry File

For type 1 geometry files, users must prepare the geometry file in txt format as shown in figure 1(d) or figure 1(e).



190072

Figure 1(d). Type 1 geometry file of a uniform cantilever beam constant domains.



190073

Figure 1(e). Type 1 geometry file of a tapered cantilever beam variable domains.

This file has two lines:

Line 1:

- The first field is the structure type.
- The second field is the structure length.
- The third field is the number of strain-sensing stations counting from the fixed end.
- The fourth field is the depth factor c_0 at the fixed end.
- The fifth field is the depth factor c_n at the free end.

Line 2 for Δl_i domain:

- 1 is for constant domain; after 1 is nothing as shown in figure 1(d).
- 2 is for variable domain; after 2 are Δl_1, Δl_2, …, Δl_n as shown in figure 1(e).

Type 1 Deflection File

After starting the program, users need to enter the strain data filename, the geometry filename, and structure type as 1. The program will compare the entered structure type 1 with the structure type programmed in the geometry file. If they are equal to 1, the program will calculate the deflections and save the results in a deflection file as shown in figure 1(f).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	time	SG_0	SG_1	SG_2	SG_3	SG_4	SG_5	SG_6	SG_7	SG_8	SG_9	SG_10	SG_11	SG_12	SG_13	SG_14	SG_15	SG_16
2	8:23:15:101	0	0.00402	0.015764	0.034716	0.060343	0.092144	0.129619	0.17224	0.219477	0.270832	0.325803	0.383862	0.44448	0.507155	0.571388	0.636653	0.702419
3	8:23:15:301	0	0.004131	0.015927	0.034786	0.060187	0.091639	0.128754	0.171324	0.218521	0.269753	0.324529	0.382349	0.442744	0.505172	0.569127	0.634093	0.69955
4	8:23:15:501	0	0.003603	0.01496	0.032763	0.056647	0.085804	0.120245	0.159689	0.203812	0.25195	0.303561	0.358139	0.415221	0.474471	0.535264	0.597049	0.659314
5	8:23:15:701	0	0.00398	0.015739	0.034999	0.06035	0.091651	0.128568	0.17058	0.21723	0.268232	0.322882	0.38059	0.440831	0.503138	0.567101	0.632114	0.697623
6	8:23:15:901	0	0.004184	0.017035	0.037286	0.064534	0.097883	0.13671	0.180506	0.228759	0.281069	0.337264	0.3968	0.458875	0.522936	0.588542	0.655334	0.722759

N1_Deflections

190074

Figure 1(f). Type 1 deflection file.

Type 1 Slope File

Similar to the deflections, the program will calculate the slopes. The results will be saved in a slope file as shown in figure 1(g).

time	SG_0	SG_1	SG_2	SG_3	SG_4	SG_5	SG_6	SG_7	SG_8	SG_9	SG_10	SG_11	SG_12	SG_13	SG_14	SG_15	SG_16
8:23:15:101	0	0.072642	0.140815	0.204103	0.26267	0.316738	0.366114	0.410605	0.450405	0.485706	0.516315	0.542041	0.563054	0.579569	0.591414	0.598375	0.600624
8:23:15:301	0	0.073804	0.140614	0.202637	0.260032	0.313018	0.364224	0.410643	0.449647	0.484242	0.514239	0.539915	0.560973	0.577158	0.588766	0.595588	0.597792
8:23:15:501	0	0.068698	0.134598	0.190882	0.242631	0.290059	0.33748	0.381883	0.42178	0.455683	0.485081	0.509788	0.531279	0.548449	0.559825	0.566511	0.568671
8:23:15:701	0	0.071915	0.143025	0.204847	0.258376	0.311903	0.360785	0.404832	0.446112	0.482939	0.513243	0.538711	0.559514	0.576648	0.589159	0.596051	0.598277
8:23:15:901	0	0.078411	0.152507	0.217399	0.27773	0.329935	0.37761	0.420569	0.458997	0.495302	0.528973	0.555707	0.575996	0.591942	0.604483	0.613062	0.615986

N1_Slopes

190075

Figure 1(g). Type 1 slope file.

Max Min Deflection File for All Structure Types

After calculating deflections for all strain-sensing stations, the max min deflections are determined and written in the output max min deflection file. Users should center the data columns so that the data are more readable. This file is always created for all structure types with the format as shown in figure 1(h).

SG Name	Time at Max Deflection	Max Deflection	Time at Min Deflection	Min Deflection
SG_0	8:23:15:101	0	8:23:15:101	0
SG_1	8:23:15:901	0.00418357	8:23:15:501	0.00360261
SG_2	8:23:15:901	0.0170345	8:23:15:501	0.0149603
SG_3	8:23:15:901	0.0372856	8:23:15:501	0.0327629
SG_4	8:23:15:901	0.0645343	8:23:15:501	0.056647
SG_5	8:23:15:901	0.0978834	8:23:15:501	0.0858039
SG_6	8:23:15:901	0.13671	8:23:15:501	0.120245
SG_7	8:23:15:901	0.180506	8:23:15:501	0.159689
SG_8	8:23:15:901	0.228759	8:23:15:501	0.203812
SG_9	8:23:15:901	0.281069	8:23:15:501	0.25195
SG_10	8:23:15:901	0.337264	8:23:15:501	0.303561
SG_11	8:23:15:901	0.3968	8:23:15:501	0.358139
SG_12	8:23:15:901	0.458875	8:23:15:501	0.415221
SG_13	8:23:15:901	0.522936	8:23:15:501	0.474471
SG_14	8:23:15:901	0.588542	8:23:15:501	0.535264
SG_15	8:23:15:901	0.655334	8:23:15:501	0.597049
SG_16	8:23:15:901	0.722759	8:23:15:501	0.659314

N1_Deflections_MaxMin

190076

Figure 1(h). Maximum and minimum deflection file.

Type 1 Output Files

Type 1 does not have depth factors nor twist angles. Type 1 output files are a deflection file, a slope file, and a max min deflection file.

Type 2 – Two-end Supported Beam

A cantilever beam with a two-end supported beam is installed with strain-sensing stations distributed along the bottom strain-sensing line. In this case, the load P is applied in the middle of the structure. The slopes will not be calculated.

Type 2 Structure

Figure 2(a) shows a two-end simply supported beam and figure 2(b) shows a two-end fixed beam. An additional case is one end fixed and other end simply supported.

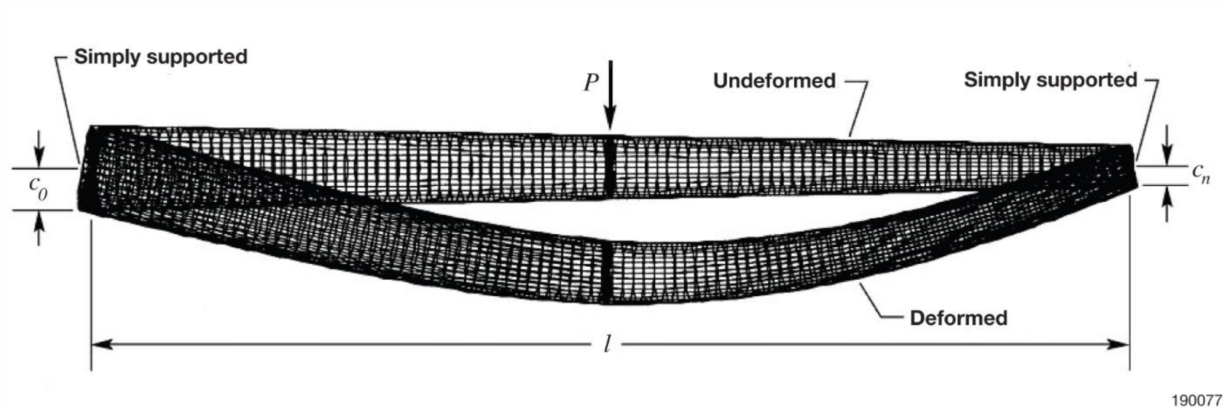


Figure 2(a). Type 2 structure of a beam with two-end simply supported.

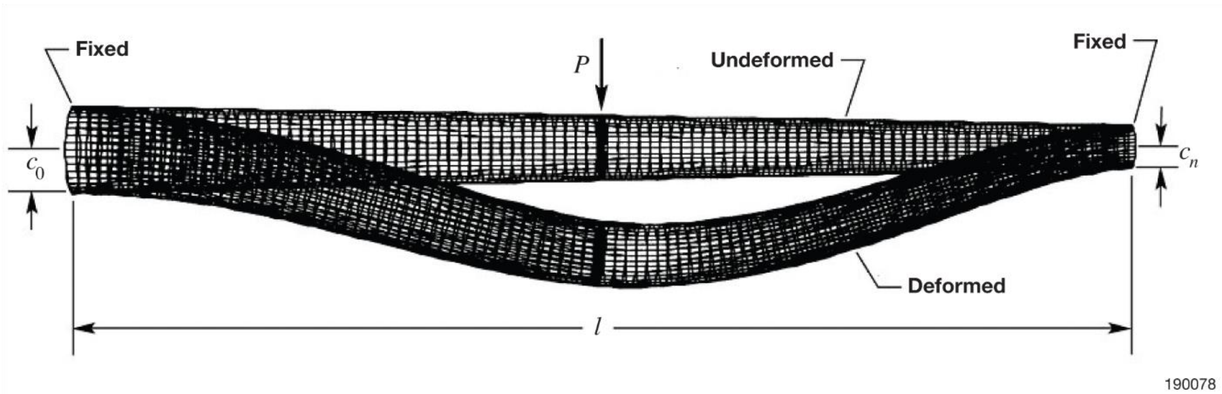
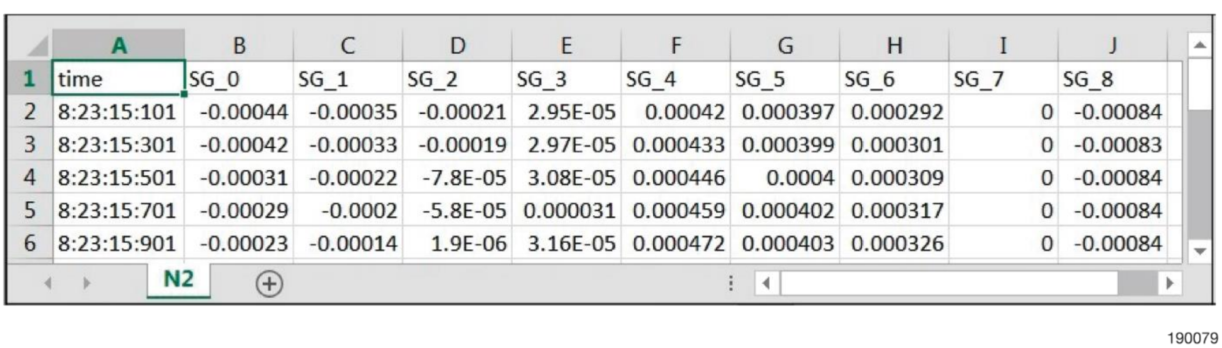


Figure 2(b). Type 2 structure of a beam with two-end fixed beam.

Type 2 Strain Data File

The strain data file is prepared in csv format similar to type 1 as shown in figure 2(c).



	A	B	C	D	E	F	G	H	I	J
1	time	SG_0	SG_1	SG_2	SG_3	SG_4	SG_5	SG_6	SG_7	SG_8
2	8:23:15:101	-0.00044	-0.00035	-0.00021	2.95E-05	0.00042	0.000397	0.000292	0	-0.00084
3	8:23:15:301	-0.00042	-0.00033	-0.00019	2.97E-05	0.000433	0.000399	0.000301	0	-0.00083
4	8:23:15:501	-0.00031	-0.00022	-7.8E-05	3.08E-05	0.000446	0.0004	0.000309	0	-0.00084
5	8:23:15:701	-0.00029	-0.0002	-5.8E-05	0.000031	0.000459	0.000402	0.000317	0	-0.00084
6	8:23:15:901	-0.00023	-0.00014	1.9E-06	3.16E-05	0.000472	0.000403	0.000326	0	-0.00084

Figure 2(c). Type 2 strain data file.

Type 2 Geometry File

The geometry file is prepared in txt format similar to the type 1 above. The geometry file is shown in figures 1(d) and 1(e).

Type 2 Deflection File

After starting the program, users need to enter the strain data filename, the geometry filename, and the structure type as 2. The program will compare the entered structure type 2 with the structure type programmed in the geometry file. If they are equal to 2, the program will calculate the deflections and save the results in a deflection file as shown in figure 2(d).

	A	B	C	D	E	F	G	H	I	J
1	time	SG_0	SG_1	SG_2	SG_3	SG_4	SG_5	SG_6	SG_7	SG_8
2	8:23:15:101	0	-0.00776	-0.0305	-0.06269	-0.09205	-0.09973	-0.07899	-0.03464	0
3	8:23:15:301	0	-0.0099	-0.03391	-0.06655	-0.09605	-0.1033	-0.08179	-0.03616	0
4	8:23:15:501	0	-0.01877	-0.04687	-0.07915	-0.10702	-0.11209	-0.08803	-0.03934	0
5	8:23:15:701	0	-0.0209	-0.05026	-0.08299	-0.111	-0.11563	-0.09079	-0.04081	0
6	8:23:15:901	0	-0.02603	-0.05791	-0.09074	-0.11809	-0.12151	-0.09511	-0.04305	0

N2_Deflections

190080

Figure 2(d). Type 2 deflection file.

Type 2 Output Files

Type 2 only have deflections. Type 2 output files are a deflection file and a max min deflection file.

Type 3 – Tapered Wing Box and Two-line System

A tapered wing box with two strain-sensing lines where strain-sensing stations are distributed along front and rear bottom lines. Twist angles will be calculated in this case. Any two strain-sensing lines can be used as long as they are in the same vertical or horizontal plane. The domain lengths for strain-sensing station *i* on two strain-sensing lines must be the same; for example, $\Delta l_i = \Delta l_{1i}$ on line 1 = Δl_{2i} on line 2.

Type 3 Structure

Figure 3(a) shows a wing box with two lower strain-sensing lines.

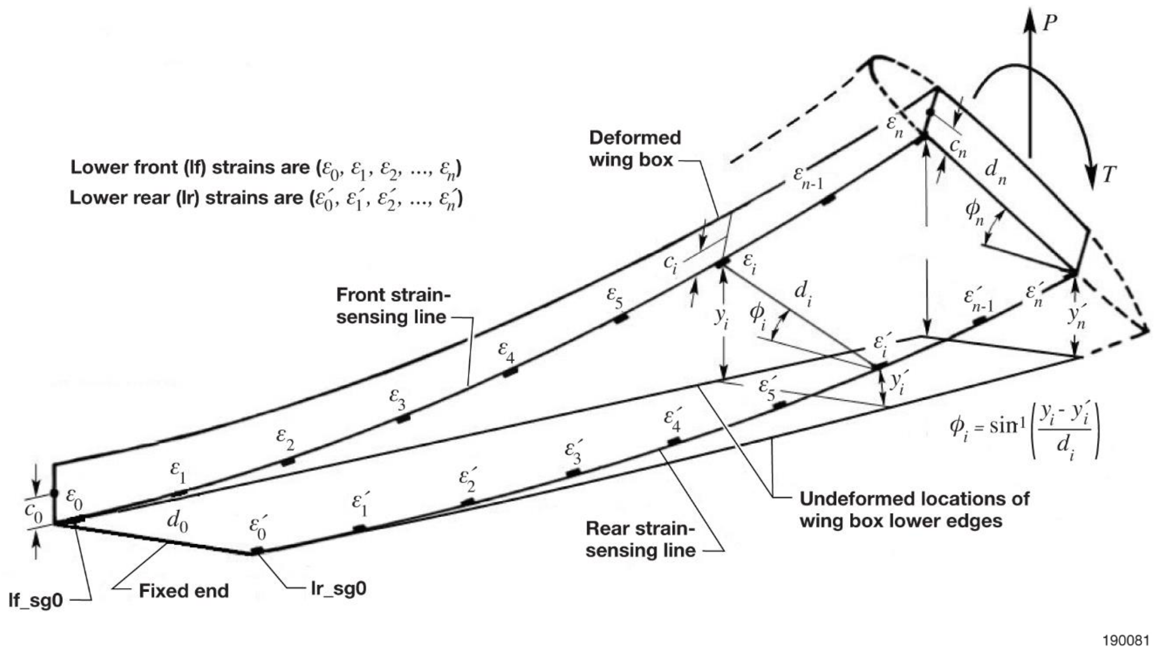
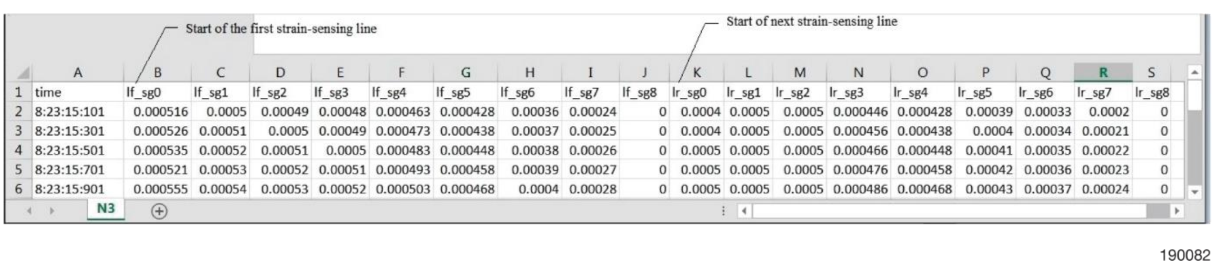


Figure 3(a). Type 3 Structure of a tapered wing box two-line system.

Type 3 Strain Data File

Users need to prepare the strain data file in csv format as shown in figure 3(b). The strain values on one line must be completed before starting on the other line.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	lr_sg0	lr_sg1	lr_sg2	lr_sg3	lr_sg4	lr_sg5	lr_sg6	lr_sg7	lr_sg8
2	8:23:15:101	0.000516	0.0005	0.00049	0.00048	0.000463	0.000428	0.00036	0.00024	0	0.0004	0.0005	0.0005	0.000446	0.000428	0.00039	0.00033	0.0002	0
3	8:23:15:301	0.000526	0.00051	0.0005	0.00049	0.000473	0.000438	0.00037	0.00025	0	0.0004	0.0005	0.0005	0.000456	0.000438	0.0004	0.00034	0.00021	0
4	8:23:15:501	0.000535	0.00052	0.00051	0.0005	0.000483	0.000448	0.00038	0.00026	0	0.0005	0.0005	0.0005	0.000466	0.000448	0.00041	0.00035	0.00022	0
5	8:23:15:701	0.000521	0.00053	0.00052	0.00051	0.000493	0.000458	0.00039	0.00027	0	0.0005	0.0005	0.0005	0.000476	0.000458	0.00042	0.00036	0.00023	0
6	8:23:15:901	0.000555	0.00054	0.00053	0.00052	0.000503	0.000468	0.0004	0.00028	0	0.0005	0.0005	0.0005	0.000486	0.000468	0.00043	0.00037	0.00024	0

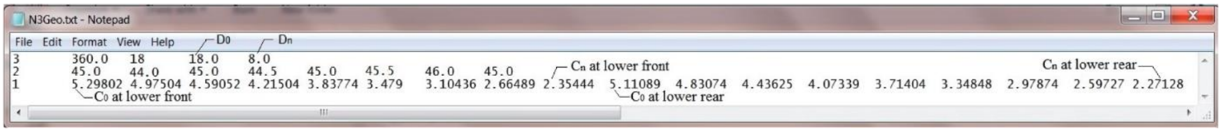
Figure 3(b). Type 3 strain data file.

Type 3 Geometry File

Users need to prepare the geometry file as shown in figures 3(c) and 3(d). This file has three lines.


```
3       360.0   18      18.0    8.0
1
1       5.29802  2.35444  5.11089  2.27128
```

Figure 3(c). Type 3 geometry file constant domains.



N3Geo.txt - Notepad

File Edit Format View Help

```
3       360.0   18      18.0    8.0
2       45.0    44.0    45.0    44.5    45.0    45.5    46.0    45.0
1       5.29802 4.97504 4.59052 4.21504 3.83774 3.479   3.10436 2.66489 2.35444 5.11089 4.83074 4.43625 4.07339 3.71404 3.34848 2.97874 2.59727 2.27128
```


Figure 3(d). Type 3 geometry file variable domains.

Line 1:

- The first field is the structure type.
- The second field is the structure length.
- The third field is the total number of strain-sensing stations.
- The fourth field is the chore-wise distance d_0 at the fixed end.
- The fifth field is the chore-wise distance d_n at the free end.

Line 2 for Δl_i domain:

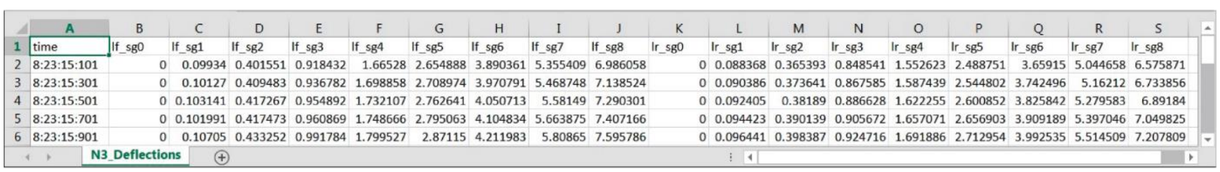
- 1 is for constant domain; after 1 is nothing as shown in figure 3(c).
- 2 is for variable domain; after 2 are Δl_1, Δl_2, …, Δl_n as shown in figure 3(d).

Line 3 for depth factors:

- If the beam depth tapers down linearly from the fixed end to the free end, enter 1. After 1, enter the depth factor c_0 and c_n for the front line, $c_0{}'$ and $c_n{}'$ for the rear line as shown in figure 3(c).
- If the beam depth does not taper down linearly from the fixed end to the free end, enter 2. After 2, enter the depth factors in the order of the strain sensors in the strain data file as shown in figure 3(d), c_0 for lf_sg0, …, c_n for lf_sg8, $c_0{}'$ for lr_sg0, …, $c_n{}'$ for lr_sg8.

Type 3 Deflection File

After starting the program, users need to enter the strain data filename, the geometry filename, and the structure type as 3. The program will compare the entered structure type 3 with the structure type programmed in the geometry file. If they are equal to 3, the program will calculate the deflections and save the results in a deflection file as shown in figure 3(e).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	lr_sg0	lr_sg1	lr_sg2	lr_sg3	lr_sg4	lr_sg5	lr_sg6	lr_sg7	lr_sg8
2	8:23:15:101	0	0.09934	0.401551	0.918432	1.66528	2.654888	3.890361	5.355409	6.986058	0	0.088368	0.365393	0.848541	1.552623	2.488751	3.65915	5.044658	6.575871
3	8:23:15:301	0	0.10127	0.409483	0.936782	1.698858	2.708974	3.970791	5.468748	7.138524	0	0.090386	0.373641	0.867585	1.587439	2.544802	3.742496	5.16212	6.733856
4	8:23:15:501	0	0.103141	0.417267	0.954892	1.732107	2.762641	4.050713	5.58149	7.290301	0	0.092405	0.38189	0.886628	1.622255	2.600852	3.825842	5.279583	6.89184
5	8:23:15:701	0	0.101991	0.417473	0.960869	1.748666	2.795063	4.104834	5.663875	7.407166	0	0.094423	0.390139	0.905672	1.657071	2.656903	3.909189	5.397046	7.049825
6	8:23:15:901	0	0.10705	0.433252	0.991784	1.799527	2.87115	4.211983	5.80865	7.595786	0	0.096441	0.398387	0.924716	1.691886	2.712954	3.992535	5.514509	7.207809

Figure 3(e). Type 3 deflection file.

Type 3 Slope File

Similar to the deflections, the program will calculate the slopes. The results will be saved in a slope file as shown in figure 3(f).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	lr_sg0	lr_sg1	lr_sg2	lr_sg3	lr_sg4	lr_sg5	lr_sg6	lr_sg7	lr_sg8
2	8:23:15:101	0	0.253873	0.518375	0.801251	1.103337	1.417189	1.72506	1.994052	2.117184	0	0.228823	0.480439	0.752939	1.042369	1.341897	1.634618	1.879095	1.984902
3	8:23:15:301	0	0.258854	0.52875	0.817482	1.125972	1.446867	1.762563	2.040488	2.168643	0	0.234011	0.491194	0.769755	1.065807	1.372634	1.673499	1.927217	2.038197
4	8:23:15:501	0	0.263721	0.53901	0.833599	1.148492	1.476429	1.79995	2.086806	2.219983	0	0.2392	0.50195	0.786572	1.089245	1.40337	1.712378	1.975337	2.091488
5	8:23:15:701	0	0.262788	0.543469	0.843917	1.165213	1.500195	1.831541	2.12733	2.265529	0	0.244388	0.512705	0.803388	1.112682	1.434105	1.751256	2.023453	2.144776
6	8:23:15:901	0	0.273776	0.55985	0.866154	1.193851	1.535873	1.87504	2.179756	2.322974	0	0.249576	0.52346	0.820204	1.136119	1.464839	1.790131	2.071567	2.19806

N3_Slopes

190086

Figure 3(f). Type 3 slope file.

Type 3 Cross-sectional Twist Angle File

Similar to the deflections and slopes, the program will calculate the twist angles. The results will be saved in a twist angle file as shown in figure 3(g).

	A	B	C	D	E	F	G	H	I	J
1	time	Station_0	Station_1	Station_2	Station_3	Station_4	Station_5	Station_6	Station_7	Station_8
2	8:23:15:101	0	0.034924	0.115095	0.22247	0.358602	0.528836	0.735988	0.989201	1.305778
3	8:23:15:301	0	0.034644	0.114089	0.220261	0.354661	0.522582	0.726705	0.976075	1.288206
4	8:23:15:501	0	0.034175	0.112608	0.217291	0.349672	0.514996	0.715805	0.961045	1.268445
5	8:23:15:701	0	0.024088	0.087007	0.175699	0.29156	0.439782	0.622771	0.849373	1.137528
6	8:23:15:901	0	0.033768	0.110978	0.213484	0.342631	0.50356	0.698541	0.936324	1.235064

N3_TwistAngles

190087

Figure 3(g). Type 3 twist angle file.

Type 3 Output Files

Type 3 does not have depth factors. Type 3 output files are a deflection file, a slope file, a twist angle file, and a max min deflection file.

Type 4 – Doubly Tapered Wing and Four-line System

A doubly tapered wing with four strain-sensing lines where strain-sensing stations are distributed along two front lines and two rear lines. Depth factors and twist angles will be calculated in this case. The domain lengths for strain-sensing station i on four strain-sensing lines must be the same; for example, $\Delta l_i = \Delta l_{1i}$ on line 1 = Δl_{2i} on line 2 = Δl_{3i} on line 3 = Δl_{4i} on line 4. Type 4 is the most complicated type; users need to prepare the strain data file and the geometry file carefully.

Type 4 Structure

Figure 4(a) shows a doubly tapered wing with a four-line system. The two extra lines must be added to determine depth factors c_i. After running this structure type one time, users have the depth factors c_i created by this program. Then, users can use structure type 3 with a two-line system.

The lower front strains are $(\varepsilon_0, \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$.
The upper front strains are $(\bar{\varepsilon}_0, \bar{\varepsilon}_1, \bar{\varepsilon}_2, \ldots, \bar{\varepsilon}_n)$.
The lower rear strains are $(\varepsilon'_0, \varepsilon'_1, \varepsilon'_2, \ldots, \varepsilon'_n)$.
The upper rear strains are $(\bar{\varepsilon}'_0, \bar{\varepsilon}'_1, \bar{\varepsilon}'_2, \ldots, \bar{\varepsilon}'_n)$.

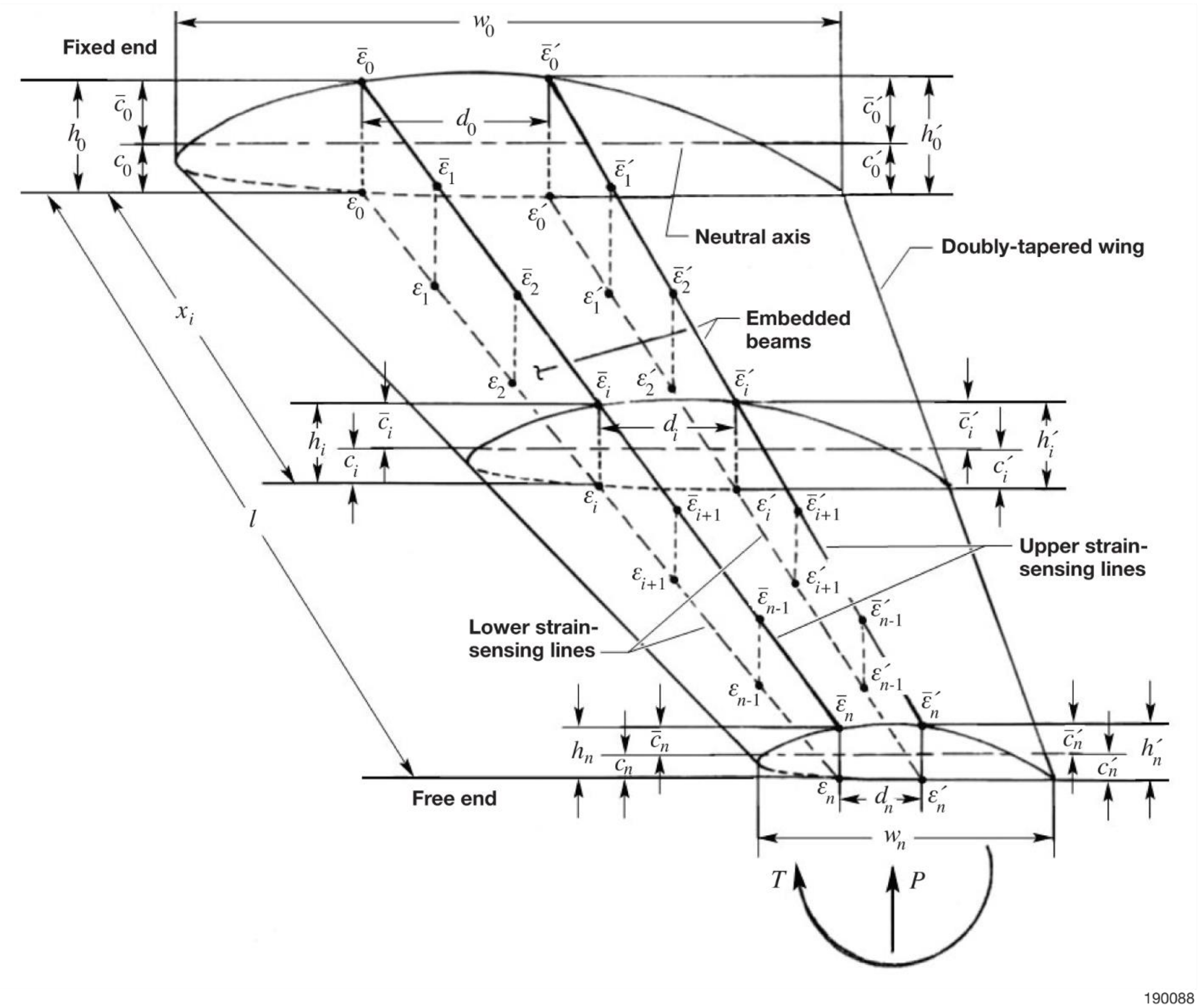


Figure 4(a). Type 4 structure of a doubly tapered wing four-line system.

Type 4 Strain Data File

Users need to prepare a single strain data file in csv format as shown in figure 4(b). The first strain sensor on each line must always be located at the fixed end. The strain values on one line must be completed before starting on the next line. The order of strains need to be exactly as shown in figure 4(b). The top half containing strain data for the front starts from column B and the bottom half containing strain data for the rear starts from column T.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	uf_sg0	uf_sg1	uf_sg2	uf_sg3	uf_sg4	uf_sg5	uf_sg6	uf_sg7	uf_sg8
2	8:23:15:101	0.000516	0.000495	0.000486	0.00048	0.000463	0.000428	0.00036	0.000245	0	-0.00064	-0.00061	-0.0006	-0.00059	-0.00057	-0.00053	-0.00044	-0.00032	0
3	8:23:15:301	0.000526	0.000505	0.000496	0.00049	0.000473	0.000438	0.00037	0.000255	0	-0.00055	-0.00059	-0.00061	-0.0006	-0.00058	-0.00054	-0.00045	-0.00033	0
4	8:23:15:501	0.000536	0.000515	0.000506	0.0005	0.000483	0.000448	0.00038	0.000265	0	-0.00056	-0.0006	-0.00062	-0.00061	-0.00059	-0.00055	-0.00046	-0.00034	0
5	8:23:15:701	0.000546	0.000525	0.000516	0.00051	0.000493	0.000458	0.00039	0.000275	0	-0.00057	-0.00061	-0.00063	-0.00062	-0.0006	-0.00056	-0.00047	-0.00035	0
6	8:23:15:901	0.000556	0.000535	0.000526	0.00052	0.000503	0.000468	0.0004	0.000285	0	-0.00058	-0.00062	-0.00064	-0.00063	-0.00061	-0.00057	-0.00048	-0.00036	0

N4

	A	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK
1	time	lr_sg0	lr_sg1	lr_sg2	lr_sg3	lr_sg4	lr_sg5	lr_sg6	lr_sg7	lr_sg8	ur_sg0	ur_sg1	ur_sg2	ur_sg3	ur_sg4	ur_sg5	ur_sg6	ur_sg7	ur_sg8
2	8:23:15:101	0.000431	0.00045	0.000453	0.000446	0.000428	0.000392	0.000327	0.000204	0	-0.00062	-0.00063	-0.00065	-0.00064	-0.00061	-0.00056	-0.00047	-0.0003	0
3	8:23:15:301	0.000441	0.00046	0.000463	0.000456	0.000438	0.000402	0.000337	0.000214	0	-0.00062	-0.00064	-0.00066	-0.00065	-0.00062	-0.00057	-0.00048	-0.00031	0
4	8:23:15:501	0.000451	0.00047	0.000473	0.000466	0.000448	0.000412	0.000347	0.000224	0	-0.00063	-0.00065	-0.00067	-0.00066	-0.00063	-0.00058	-0.00049	-0.00032	0
5	8:23:15:701	0.000461	0.00048	0.000483	0.000476	0.000458	0.000422	0.000357	0.000234	0	-0.00064	-0.00066	-0.00068	-0.00067	-0.00064	-0.00059	-0.0005	-0.00033	0
6	8:23:15:901	0.000471	0.00049	0.000493	0.000486	0.000468	0.000432	0.000367	0.000244	0	-0.00065	-0.00067	-0.00069	-0.00068	-0.00065	-0.0006	-0.00051	-0.00034	0

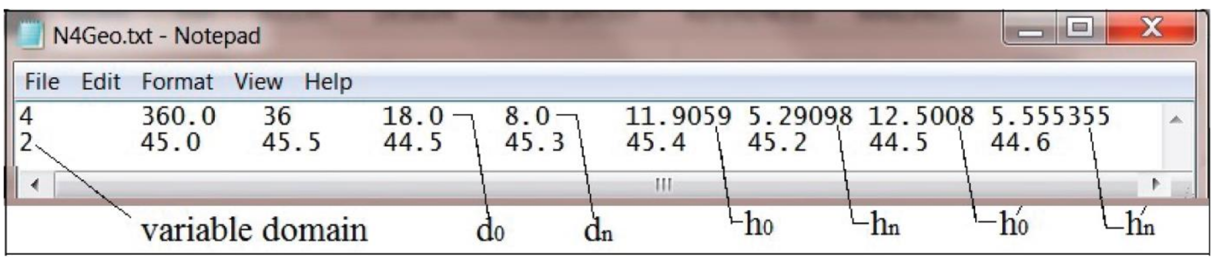
N4

190089

Figure 4(b). Type 4 strain data file.

Type 4 Geometry File

Users need to prepare the Geometry file in txt format as shown in figure 4(c).



190090

Figure 4(c). Type 4 geometry File.

Line 1:

- The first field is the structure type.
- The second field is the structure length.
- The third field is the total number of strain-sensing stations.
- The fourth field is the separation distance from the front and the rear at the fixed end, d_0.
- The fifth field is the separation distance from the front and the rear at the free end, d_n.
- The sixth field is the beam depth at the front fixed end, h_0.
- The seventh field is the beam depth at the front free end, h_n.
- The eighth field is the beam depth at the rear fixed end, h_0'.
- The ninth field is the beam depth at the rear free end, h_n'.

Line 2 for Δl_i domain:

- 1 is for constant domain; nothing after 1 as shown in figure 1(d).
- 2 is for variable domain; after 2 are Δl_1, Δl_2, …, Δl_n as shown in figure 4(c).

Type 4 Deflection File

After starting the program, users need to enter the strain data filename, the geometry filename, and the structure type as 4. The program will compare the entered structure type 4 with the structure type programmed in the geometry file. If they are equal to 4, the program will calculate the deflections and save the results in a deflection file as shown in figure 4(d).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	uf_sg0	uf_sg1	uf_sg2	uf_sg3	uf_sg4	uf_sg5	uf_sg6	uf_sg7	uf_sg8
2	8:23:15:101	0	0.09934	0.406064	0.918278	1.670733	2.671672	3.916812	5.369012	6.986921	0	-0.09933	-0.40605	-0.91827	-1.67073	-2.67168	-3.91683	-5.36886	-6.98697
3	8:23:15:301	0	0.101291	0.414139	0.936707	1.704535	2.726251	3.997975	5.482817	7.139541	0	-0.08847	-0.37615	-0.87183	-1.61137	-2.60371	-3.84505	-5.29852	-6.92285
4	8:23:15:501	0	0.103242	0.422214	0.955136	1.738337	2.78083	4.079139	5.596623	7.292161	0	-0.09004	-0.38268	-0.88676	-1.63875	-2.6479	-3.91076	-5.39061	-7.04614
5	8:23:15:701	0	0.105193	0.430289	0.973565	1.772139	2.835408	4.160303	5.710429	7.44478	0	-0.09162	-0.38922	-0.90169	-1.66613	-2.6921	-3.97647	-5.48269	-7.16943
6	8:23:15:901	0	0.107144	0.438364	0.991994	1.805942	2.889987	4.241467	5.824235	7.5974	0	-0.09319	-0.39575	-0.91661	-1.69351	-2.73629	-4.04219	-5.57478	-7.29272
7	8:23:16:101	0	0.109095	0.446439	1.010422	1.839744	2.944565	4.322631	5.93804	7.75002	0	-0.09476	-0.40229	-0.93154	-1.72089	-2.78049	-4.1079	-5.66687	-7.41601

	A	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK
1	time	lr_sg0	lr_sg1	lr_sg2	lr_sg3	lr_sg4	lr_sg5	lr_sg6	lr_sg7	lr_sg8	ur_sg0	ur_sg1	ur_sg2	ur_sg3	ur_sg4	ur_sg5	ur_sg6	ur_sg7	ur_sg8
2	8:23:15:101	0	0.088368	0.369564	0.848353	1.557702	2.504549	3.684109	5.057507	6.576918	0	-0.08833	-0.36947	-0.84822	-1.55753	-2.50435	-3.68387	-5.05713	-6.57655
3	8:23:15:301	0	0.090387	0.377906	0.867393	1.592631	2.560958	3.76803	5.175235	6.734814	0	-0.08849	-0.37189	-0.85595	-1.57416	-2.53377	-3.7303	-5.12494	-6.67007
4	8:23:15:501	0	0.092405	0.386248	0.886432	1.627561	2.617367	3.851952	5.292964	6.892709	0	-0.0899	-0.37775	-0.86932	-1.59868	-2.57335	-3.78913	-5.20735	-6.78041
5	8:23:15:701	0	0.094423	0.39459	0.905471	1.66249	2.673776	3.935874	5.410693	7.050605	0	-0.09131	-0.3836	-0.88269	-1.62321	-2.61293	-3.84796	-5.28977	-6.89076
6	8:23:15:901	0	0.096441	0.402932	0.924511	1.69742	2.730185	4.019796	5.528422	7.2085	0	-0.09272	-0.38945	-0.89606	-1.64773	-2.65251	-3.90678	-5.37218	-7.0011
7	8:23:16:101	0	0.09846	0.411274	0.94355	1.73235	2.786594	4.103717	5.646151	7.366396	0	-0.09412	-0.39531	-0.90943	-1.67225	-2.69208	-3.96561	-5.4546	-7.11145

N4_Deflections

190091

Figure 4(d). Type 4 deflection file.

Type 4 Slope File

Similar to the deflections, the program will calculate the slopes. The results will be saved in a slope file as shown in figure 4(e).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	uf_sg0	uf_sg1	uf_sg2	uf_sg3	uf_sg4	uf_sg5	uf_sg6	uf_sg7	uf_sg8
2	8:23:15:101	0	0.253873	0.521314	0.801046	1.105146	1.421786	1.731025	1.997027	2.119064	0	-0.25386	-0.52131	-0.80105	-1.10516	-1.4218	-1.73103	-1.99656	-2.12009
3	8:23:15:301	0	0.258894	0.531788	0.817312	1.127858	1.451604	1.768702	2.043537	2.170553	0	-0.23138	-0.49771	-0.78215	-1.09147	-1.41385	-1.72946	-2.00198	-2.1294
4	8:23:15:501	0	0.263915	0.542262	0.833578	1.150569	1.481421	1.806378	2.090045	2.222038	0	-0.23544	-0.5062	-0.79533	-1.10986	-1.43799	-1.75995	-2.03945	-2.17076
5	8:23:15:701	0	0.268936	0.552736	0.849844	1.17328	1.511237	1.844052	2.13655	2.273519	0	-0.2395	-0.51469	-0.80851	-1.12824	-1.46212	-1.79044	-2.07692	-2.21212
6	8:23:15:901	0	0.273957	0.56321	0.86611	1.195991	1.541052	1.881725	2.183053	2.324997	0	-0.24355	-0.52319	-0.82169	-1.14663	-1.48625	-1.82093	-2.11439	-2.25347
7	8:23:16:101	0	0.278978	0.573683	0.882375	1.218701	1.570867	1.919396	2.229552	2.376471	0	-0.24761	-0.53168	-0.83488	-1.16502	-1.51038	-1.85142	-2.15186	-2.29483

	A	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK
1	time	lr_sg0	lr_sg1	lr_sg2	lr_sg3	lr_sg4	lr_sg5	lr_sg6	lr_sg7	lr_sg8	ur_sg0	ur_sg1	ur_sg2	ur_sg3	ur_sg4	ur_sg5	ur_sg6	ur_sg7	ur_sg8
2	8:23:15:101	0	0.228823	0.483235	0.752707	1.044066	1.346256	1.640277	1.882036	1.986902	0	-0.22872	-0.48318	-0.75266	-1.04403	-1.34622	-1.64021	-1.88175	-1.98724
3	8:23:15:301	0	0.234012	0.494052	0.769518	1.067543	1.377096	1.679297	1.930195	2.040188	0	-0.22982	-0.48794	-0.76162	-1.05764	-1.36497	-1.66464	-1.91243	-2.02142
4	8:23:15:501	0	0.2392	0.504869	0.786329	1.091019	1.407936	1.718315	1.978351	2.09347	0	-0.23345	-0.49555	-0.77343	-1.07411	-1.38657	-1.69191	-1.94596	-2.05845
5	8:23:15:701	0	0.244388	0.515686	0.80314	1.114495	1.438774	1.757332	2.026505	2.146749	0	-0.23709	-0.50316	-0.78523	-1.09057	-1.40818	-1.71919	-1.97949	-2.09548
6	8:23:15:901	0	0.249576	0.526503	0.81995	1.137971	1.469612	1.796348	2.074655	2.200024	0	-0.24072	-0.51077	-0.79704	-1.10704	-1.42978	-1.74646	-2.01302	-2.13251
7	8:23:16:101	0	0.254764	0.537321	0.836761	1.161446	1.500448	1.835362	2.122803	2.253295	0	-0.24436	-0.51837	-0.80884	-1.12351	-1.45138	-1.77373	-2.04655	-2.16954

N4_Slopes

190092

Figure 4(e). Type 4 slope file.

Type 4 Depth Factor File

The program will calculate the depth factors for type 4. The results will be saved in a depth factor file as shown in figure 4(f). Structure type 4 does not have depth factors c_i; therefore, a four-line system is used. After the c_i are calculated from this program, users can use structure type 3 with a two-line system.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	uf_sg0	uf_sg1	uf_sg2	uf_sg3	uf_sg4	uf_sg5	uf_sg6	uf_sg7	uf_sg8
2	5.29802	4.97504	4.59052	4.21504	3.83774	3.479	3.10436	2.66489	2.35444	6.60788	6.10399	5.66165	5.21026	4.7607	4.29257	3.84035	3.45295	2.93654

	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ
1	lr_sg0	lr_sg1	lr_sg2	lr_sg3	lr_sg4	lr_sg5	lr_sg6	lr_sg7	lr_sg8	ur_sg0	ur_sg1	ur_sg2	ur_sg3	ur_sg4	ur_sg5	ur_sg6	ur_sg7	ur_sg8
2	5.11089	4.83074	4.43625	4.07339	3.71404	3.34848	2.97874	2.59727	2.27128	7.38991	6.80188	6.32819	5.82287	5.31404	4.81141	4.31298	3.82627	3.28408

N4_DepthFactors

190093

Figure 4(f). Type 4 depth factor file.

Type 4 Twist Angle File

The program will also calculate the twist angles for type 4. The results will be saved in a twist angle file as shown in figure 4(g). Users can change the title names LwrStation_0, …, UprStation_8 to whatever names that make sense to them.

	A	B	C	D	E	F	G	H	I	J
1	time	LwrStation_0	LwrStation_1	LwrStation_2	LwrStation_3	LwrStation_4	LwrStation_5	LwrStation_6	LwrStation_7	LwrStation_8
2	8:23:15:101	0	0.034923	0.115094	0.222468	0.358599	0.528832	0.735983	0.989195	1.305771
3	8:23:15:301	0	0.03471	0.114253	0.220524	0.355022	0.523041	0.727263	0.976731	1.288961
4	8:23:15:501	0	0.034496	0.113412	0.218579	0.351444	0.517251	0.718543	0.964267	1.272151
5	8:23:15:701	0	0.034282	0.112572	0.216634	0.347866	0.51146	0.709823	0.951803	1.255341
6	8:23:15:901	0	0.034069	0.111731	0.21469	0.344289	0.50567	0.701103	0.939339	1.238532
7	8:23:16:101	0	0.033855	0.110891	0.212745	0.340711	0.499879	0.692383	0.926875	1.221722

	K	L	M	N	O	P	Q	R	S
1	UprStation_0	UprStation_1	UprStation_2	UprStation_3	UprStation_4	UprStation_5	UprStation_6	UprStation_7	UprStation_8
2	0	0.035029	0.11537	0.222871	0.35914	0.52951	0.736809	0.989891	1.30708
3	0	0.000068	0.013298	0.050638	0.118045	0.220974	0.362442	0.55148	0.806163
4	0	0.000457	0.01544	0.055593	0.127114	0.235582	0.384204	0.582198	0.847397
5	0	0.000981	0.017583	0.060548	0.136184	0.250189	0.405967	0.612916	0.888632
6	0	0.001506	0.019725	0.065504	0.145254	0.264797	0.42773	0.643634	0.929867
7	0	0.00203	0.021867	0.070459	0.154323	0.279405	0.449493	0.674352	0.971103

N4_TwistAngles

190094

Figure 4(g). Type 4 twist angle file.

Type 4 Output Files

The program creates the most output files for this type. Type 4 output files are a deflection file, a slope file, a depth factor file, a twist angle file, and a max min deflection file.

Type 5 – Thin Uniform Plate

For a uniform plate ($c_0 = c_n$), the strain sensor system requires multiple parallel strain-sensing lines across the two opposite edges as shown in figure 5(a). The four edges of the plate can be either fixed or simply supported. The load is applied somewhere in the center of the plate. The plate must be very thin and the depth factor is very small compared to the length. The domain lengths for strain-sensing station i on every strain-sensing line must be the same; for example, $\Delta l_i = \Delta l_{1i}$ on line 1 = Δl_{2i} on line 2 = Δl_{3i} on line 3 = Δl_{ki} on line k. Similar to type 2, the slopes will not be calculated.

Type 5 Structure

Figure 5(a) shows a thin uniform plate with parallel strain-sensing lines with undeformed and deformed shapes.

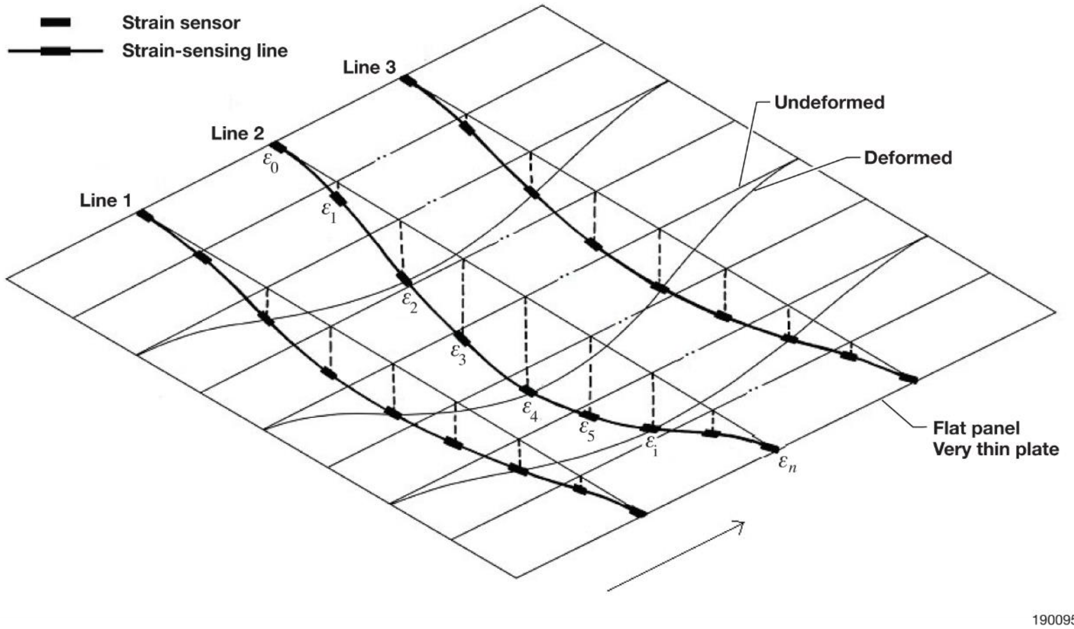


Figure 5(a). Type 5 structure of a very thin plate.

Type 5 Strain Data File

Users need to prepare the strain data file in csv format with strain values on one line which must be completed before starting on the next line. Lines must start from one end across to the other end as shown in figure 5(b); for example, line 1, line 2, …, line k. For more details of how to arrange the type 5 strain data file, users can refer to the type 1 strain data file and the type 4 strain data file in the report.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	Line1_SG0	Line1_SG1	Line1_SG2	Line1_SG3	Line1_SG4	Line1_SG5	Line1_SG6	Line1_SG7	Line1_SG8	Line2_SG0	Line2_SG1	Line2_SG2	Line2_SG3	Line2_SG4	Line2_SG5	Line2_SG6	Line2_SG7	Line2_SG8
2	8:23:15:101	0	0.0000200	0.0000520	0.0001240	0.0003480	0.0001240	0.0000520	0.0000200	0	0	0.0000201	0.0000523	0.0001248	0.0003502	0.0001248	0.0000523	0.0000201	0
3	8:23:15:301	0	0.0000201	0.0000523	0.0001247	0.0003500	0.0001247	0.0000523	0.0000201	0	0	0.0000202	0.0000526	0.0001255	0.0003523	0.0001255	0.0000526	0.0000202	0
4	8:23:15:501	0	0.0000201	0.0000524	0.0001249	0.0003504	0.0001249	0.0000524	0.0000201	0	0	0.0000203	0.0000527	0.0001257	0.0003527	0.0001257	0.0000527	0.0000203	0
5	8:23:15:701	0	0.0000203	0.0000527	0.0001256	0.0003524	0.0001256	0.0000527	0.0000203	0	0	0.0000204	0.0000530	0.0001264	0.0003548	0.0001264	0.0000530	0.0000200	0
6	8:23:15:901	0	0.0000203	0.0000527	0.0001257	0.0003528	0.0001257	0.0000527	0.0000203	0	0	0.0000204	0.0000531	0.0001265	0.0003551	0.0001265	0.0000531	0.0000200	0

	A	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL
1	time	Line3_SG0	Line3_SG1	Line3_SG2	Line3_SG3	Line3_SG4	Line3_SG5	Line3_SG6	Line3_SG7	Line3_SG8										
2	8:23:15:101	0	0.0000200	0.0000521	0.0001242	0.0003486	0.0001242	0.0000521	0.0000200	0										
3	8:23:15:301	0	0.0000200	0.0000523	0.0001247	0.0003500	0.0001247	0.0000523	0.0000201	0										
4	8:23:15:501	0	0.0000200	0.0000523	0.0001248	0.0003502	0.0001248	0.0000523	0.0000201	0										
5	8:23:15:701	0	0.0000200	0.0000525	0.0001253	0.0003516	0.0001253	0.0000525	0.0000200	0										
6	8:23:15:901	0	0.0000200	0.0000526	0.0001254	0.0003519	0.0001254	0.0000526	0.0000200	0										

N5

190096

Figure 5(b). Type 5 strain data file for three lines.

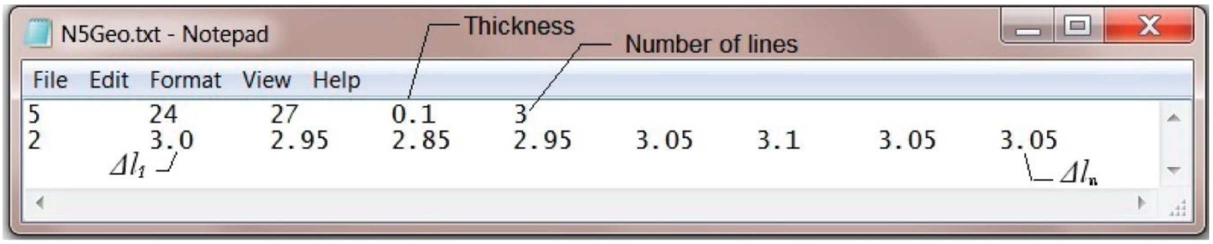
Type 5 Geometry File

The geometry file should be prepared in txt format as shown in figure 5(c). This file has two lines:
Line 1:

- The first field is the structure type.
- The second field is the length of the strain-sensing line.
- The third field is the number of strain-sensing stations.
- The fourth field is the plate thickness.
- The fifth field is the number of strain-sensing lines on the plate.

Line 2 for Δl_i domain:

- 1 is for constant domain; after 1 is nothing as shown in figure 1(d).
- 2 is for variable domain; after 2 are Δl_1, Δl_2, …, Δl_n as shown in figure 5(c).

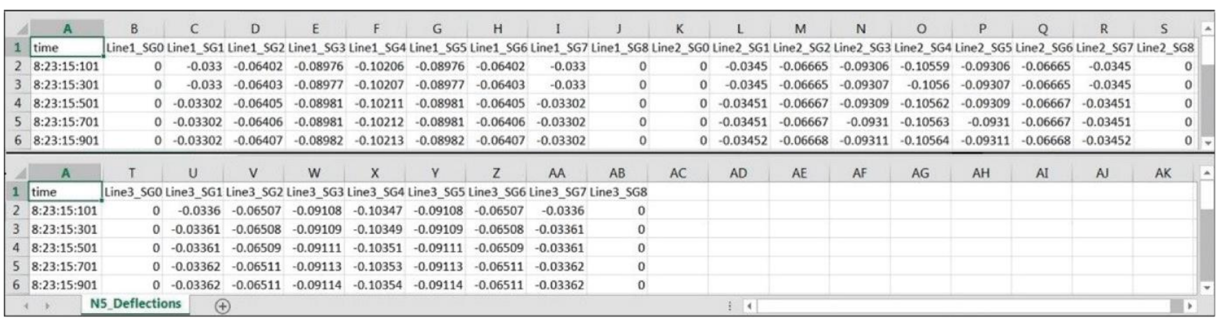


190097

Figure 5(c). Type 5 geometry file.

Type 5 Deflection File

After starting the program, users need to enter the strain data filename, the geometry filename, and the structure type as 5. The program will compare the entered structure type 5 with the structure type programmed in the geometry file. If they are equal to 5, the program will calculate the deflections and save the results in a deflection file as shown in figure 5(d).



time	Line1_SG0	Line1_SG1	Line1_SG2	Line1_SG3	Line1_SG4	Line1_SG5	Line1_SG6	Line1_SG7	Line1_SG8
8:23:15:101	0	-0.033	-0.06402	-0.08976	-0.10206	-0.08976	-0.06402	-0.033	0
8:23:15:301	0	-0.033	-0.06403	-0.08977	-0.10207	-0.08977	-0.06403	-0.033	0
8:23:15:501	0	-0.03302	-0.06405	-0.08981	-0.10211	-0.08981	-0.06405	-0.03302	0
8:23:15:701	0	-0.03302	-0.06406	-0.08981	-0.10212	-0.08981	-0.06406	-0.03302	0
8:23:15:901	0	-0.03302	-0.06407	-0.08982	-0.10213	-0.08982	-0.06407	-0.03302	0

time	Line2_SG0	Line2_SG1	Line2_SG2	Line2_SG3	Line2_SG4	Line2_SG5	Line2_SG6	Line2_SG7	Line2_SG8
8:23:15:101	0	-0.0345	-0.06665	-0.09306	-0.10559	-0.09306	-0.06665	-0.0345	0
8:23:15:301	0	-0.0345	-0.06665	-0.09307	-0.1056	-0.09307	-0.06665	-0.0345	0
8:23:15:501	0	-0.03451	-0.06667	-0.09309	-0.10562	-0.09309	-0.06667	-0.03451	0
8:23:15:701	0	-0.03451	-0.06667	-0.0931	-0.10563	-0.0931	-0.06667	-0.03451	0
8:23:15:901	0	-0.03452	-0.06668	-0.09311	-0.10564	-0.09311	-0.06668	-0.03452	0

time	Line3_SG0	Line3_SG1	Line3_SG2	Line3_SG3	Line3_SG4	Line3_SG5	Line3_SG6	Line3_SG7	Line3_SG8
8:23:15:101	0	-0.0336	-0.06507	-0.09108	-0.10347	-0.09108	-0.06507	-0.0336	0
8:23:15:301	0	-0.03361	-0.06508	-0.09109	-0.10349	-0.09109	-0.06508	-0.03361	0
8:23:15:501	0	-0.03361	-0.06509	-0.09111	-0.10351	-0.09111	-0.06509	-0.03361	0
8:23:15:701	0	-0.03362	-0.06511	-0.09113	-0.10353	-0.09113	-0.06511	-0.03362	0
8:23:15:901	0	-0.03362	-0.06511	-0.09114	-0.10354	-0.09114	-0.06511	-0.03362	0

190098

Figure 5(d). Type 5 deflection file for three lines.

Type 5 Output Files

Type 5 does not have slopes, nor twist angles. The output files are a deflection file and a max min deflection file.

Type 6 – Long Beam with Known Depth Factors

Type 6 is the same as type 1 where the length of the structure is very long compared to the width and known depth factors c_i. For large deformations, $tan\theta_i$ is replaced by θ_i in the Displacement Transfer Function used for type 1.

Type 6 Structure

Figure 6(a) shows a nonuniform long cantilever beam with undeformed and deformed shapes.

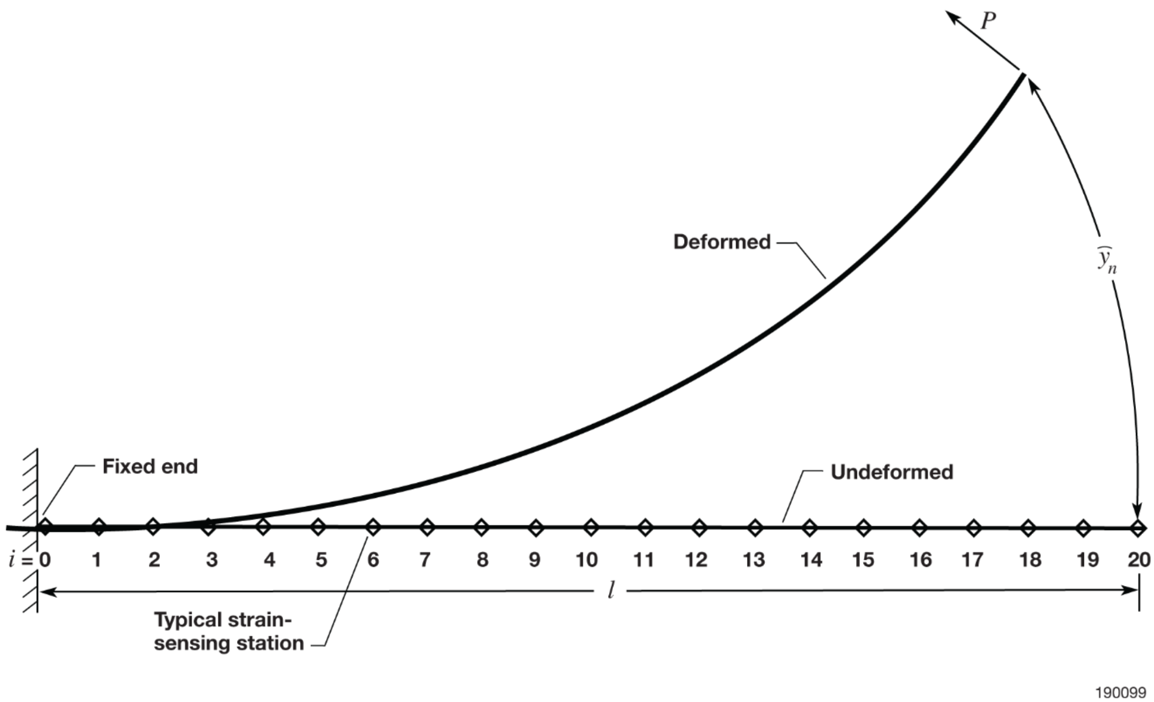


Figure 6(a). Type 6 structure of a long beam with known c_i.

Type 6 Strain Data File

For type 6, recorded strains must be arranged as type 1 and is shown in figure 6(b). For more details, refer to the type 1 strain data file in the report.

	A	B	C	D	E	F	G	H	I	J	K
1	time	SG_0	SG_1	SG_2	SG_3	SG_4	SG_5	SG_6	SG_7	SG_8	SG_9
2	8:23:15:101	0.011550	0.011670	0.011770	0.011850	0.011910	0.011950	0.011940	0.011900	0.011820	0.011680
3	8:23:15:301	0.011552	0.011672	0.011772	0.011852	0.011912	0.011952	0.011942	0.011902	0.011822	0.011682
4	8:23:15:501	0.011554	0.011674	0.011774	0.011854	0.011914	0.011954	0.011944	0.011904	0.011824	0.011684
5	8:23:15:701	0.011565	0.011685	0.011785	0.011865	0.011925	0.011966	0.011956	0.011915	0.011835	0.011695
6	8:23:15:901	0.011567	0.011687	0.011787	0.011867	0.011927	0.011967	0.011957	0.011917	0.011837	0.011697

	L	M	N	O	P	Q	R	S	T	U	V
1	SG_10	SG_11	SG_12	SG_13	SG_14	SG_15	SG_16	SG_17	SG_18	SG_19	SG_20
2	0.011480	0.011200	0.010840	0.010380	0.009800	0.009090	0.008190	0.007060	0.005590	0.003550	0.000370
3	0.011481	0.011201	0.010841	0.010381	0.009801	0.009091	0.008191	0.007061	0.005591	0.003550	0.000370
4	0.011483	0.011203	0.010843	0.010383	0.009803	0.009093	0.008192	0.007062	0.005592	0.003551	0.000370
5	0.011495	0.011215	0.010854	0.010393	0.009813	0.009102	0.008201	0.007069	0.005597	0.003555	0.000370
6	0.011497	0.011216	0.010856	0.010395	0.009814	0.009103	0.008202	0.007070	0.005598	0.003555	0.000371

N6

190100

Figure 6(b). Type 6 strain data file.

Type 6 Geometry File

For type 6, users need to prepare the geometry file in txt format similar to type 1. For more details, refer to the type 1 geometry file shown in figures 1(d) and 1(e).

Type 6 Deflection File

After starting the program, users need to enter the strain data filename, the geometry filename, and the structure type as 6. The program will compare the entered structure type 6 with the structure type programmed in the geometry file. If they are equal to 6, the program will calculate the deflections and save the results in a deflection file as shown in figure 6(c).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	time	SG_0	SG_1	SG_2	SG_3	SG_4	SG_5	SG_6	SG_7	SG_8	SG_9	SG_10	SG_11	SG_12	SG_13	SG_14	SG_15	SG_16	SG_17	SG_18	SG_19	SG_20
2	8:23:15:101	0	0.330129	1.34229	3.070225	5.54926	8.816516	12.9108	17.87167	23.74	30.55773	38.36669	47.20806	57.1209	68.14105	80.29859	93.61429	108.0946	123.7206	140.4318	158.0903	176.3968
3	8:23:15:301	0	0.330169	1.342451	3.070593	5.549926	8.817574	12.91235	17.87381	23.74285	30.56139	38.3713	47.21373	57.12776	68.14923	80.30823	93.62552	108.1075	123.7355	140.4487	158.1093	176.418
4	8:23:15:501	0	0.330231	1.342706	3.071177	5.55098	8.819249	12.91481	17.87721	23.74736	30.5672	38.37859	47.2227	57.13861	68.16218	80.32348	93.64331	108.1281	123.759	140.4754	158.1393	176.4515
5	8:23:15:701	0	0.330297	1.342975	3.071791	5.552091	8.821013	12.91739	17.88078	23.75211	30.57331	38.38626	47.23214	57.15004	68.17581	80.33955	93.66204	108.1497	123.7837	140.5035	158.171	176.4868
6	8:23:15:901	0	0.330354	1.343203	3.072313	5.553034	8.822513	12.91959	17.88382	23.75615	30.57851	38.39279	47.24017	57.15975	68.1874	80.35321	93.67796	108.1681	123.8048	140.5274	158.1979	176.5168

N6_Deflections

190101

Figure 6(c). Type 6 deflection file.

Type 6 Slope File

Similar to the deflections, the program will calculate the slopes. The results will be saved in a slope file as shown in figure 6(d).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	time	SG_0	SG_1	SG_2	SG_3	SG_4	SG_5	SG_6	SG_7	SG_8	SG_9	SG_10	SG_11	SG_12	SG_13	SG_14	SG_15	SG_16	SG_17	SG_18	SG_19	SG_20
2	8:23:15:101	0	2.542574	5.211239	8.011689	10.95036	14.03455	17.26852	20.65578	24.20184	27.9084	31.77413	35.79388	39.95771	44.24952	48.64077	53.08887	57.52154	61.81696	65.76593	68.96153	70.50141
3	8:23:15:301	0	2.542879	5.211864	8.01265	10.95167	14.03624	17.27059	20.65825	24.20474	27.91175	31.77794	35.79817	39.96251	44.25483	48.64661	53.09524	57.52844	61.82438	65.77382	68.96981	70.50987
4	8:23:15:501	0	2.543362	5.212855	8.014173	10.95375	14.03891	17.27388	20.66218	24.20934	27.91705	31.78398	35.80497	39.9701	44.26324	48.65585	53.10532	57.53937	61.83613	65.78632	68.98291	70.52326
5	8:23:15:701	0	2.543871	5.213897	8.015776	10.95594	14.04171	17.27733	20.66631	24.21418	27.92264	31.79034	35.81213	39.97809	44.27209	48.66558	53.11595	57.55088	61.84849	65.79948	68.99671	70.53737
6	8:23:15:901	0	2.544303	5.214784	8.017138	10.95781	14.0441	17.28027	20.66983	24.2183	27.92738	31.79574	35.81822	39.98489	44.27961	48.67386	53.12498	57.56066	61.85901	65.81066	69.00844	70.54936

N6_Slopes

190102

Figure 6(d). Type 6 slope file.

Type 6 Output Files

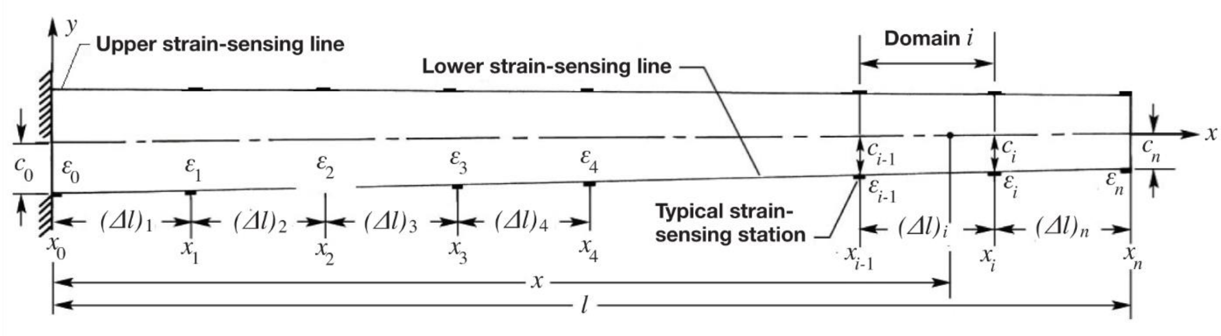
Type 6 does not have depth factors, nor twist angles. Type 6 output files are a deflection file, a slope file, and a max min deflection file.

Type 7 – Long Beam with Unknown Depth Factors

Type 7 is the same as type 1 where the length of the structure is very long compared to the width and unknown depth factors c_i. For this type, an extra strain-sensing line on the upper surface is needed to calculate the depth factors c_i. For large deformations, $tan\theta_i$ is replaced by θ_i in the Displacement Transfer Function used for type 1.

Type 7 Structure

Figure 7(a) shows a long cantilever beam with two strain-sensing lines.



190103

Figure 7(a). Type 7 Structure of a long beam with unknown c_i.

Type 7 Strain Data File

Users need to prepare the strain data file in csv format as shown in figure 7(b). The two strain sensors lf_sg0 and uf_sg0 are always at the fixed end. The strain values on one line must be completed before starting on the other line; normally the lower front strain-sensing line is first followed by the upper front strain-sensing line.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	uf_sg0	uf_sg1	uf_sg2	uf_sg3	uf_sg4	uf_sg5	uf_sg6	uf_sg7	uf_sg8
2	8:23:15:101	0.000516	0.000495	0.000486	0.00048	0.000463	0.000428	0.00036	0.000245	0	-0.00064	-0.00061	-0.0006	-0.00059	-0.00057	-0.00053	-0.00044	-0.00032	0
3	8:23:15:301	0.000526	0.000505	0.000496	0.00049	0.000473	0.000438	0.00037	0.000255	0	-0.00055	-0.00059	-0.00061	-0.0006	-0.00058	-0.00054	-0.00045	-0.00033	0
4	8:23:15:501	0.000536	0.000515	0.000506	0.0005	0.000483	0.000448	0.00038	0.000265	0	-0.00056	-0.0006	-0.00062	-0.00061	-0.00059	-0.00055	-0.00046	-0.00034	0
5	8:23:15:701	0.000546	0.000525	0.000516	0.00051	0.000493	0.000458	0.00039	0.000275	0	-0.00057	-0.00061	-0.00063	-0.00062	-0.0006	-0.00056	-0.00047	-0.00035	0
6	8:23:15:901	0.000556	0.000535	0.000526	0.00052	0.000503	0.000468	0.0004	0.000285	0	-0.00058	-0.00062	-0.00064	-0.00063	-0.00061	-0.00057	-0.00048	-0.00036	0
7	8:23:16:101	0.000566	0.000545	0.000536	0.00053	0.000513	0.000478	0.00041	0.000295	0	-0.00059	-0.00063	-0.00065	-0.00064	-0.00062	-0.00058	-0.00049	-0.00037	0

N7

190104

Figure 7(b). Type 7 strain data file.

Type 7 Geometry File

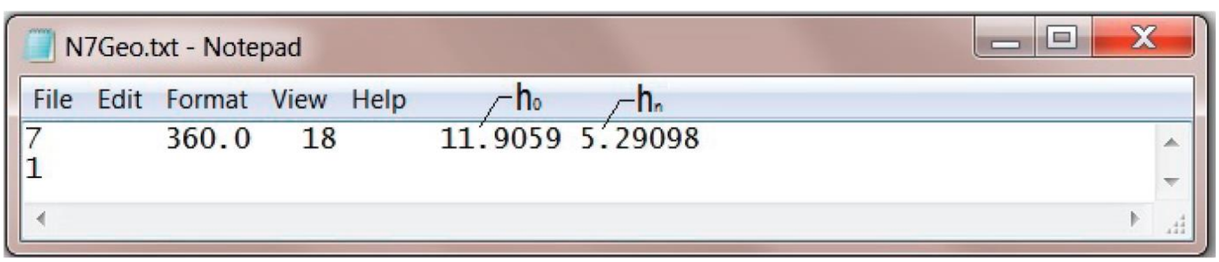
For type 7, users need to prepare the Geometry file in txt format as shown in figures 7(c) and 7(d). This file has two lines:

Line 1:

- The first field is the structure type.
- The second field is the structure length.
- The third field is the number of strain-sensing stations.
- The fourth field is the wing root depth h_0 at the front.
- The fifth field is wing tip depth h_n at the front.

Line 2 for Δl_i domain:

- 1 is for constant domain; after 1 is nothing as shown in figure 7(c).
- 2 is for variable domain; after 2 are Δl_1, Δl_2, …, Δl_n as shown in figure 7(d).



190105

Figure 7(c). Type 7 geometry file constant domains.

N7GeoVar.txt - Notepad

```
7         360.0    18        11.9059 5.29098
2         40.0   39.85   40.15   40.2   40.1   39.9   39.8   40.1  39.9
```

190106

Figure 7(d). Type 7 geometry file variable domains.

Type 7 Deflection File

After starting the program, users need to enter the strain data filename, the geometry filename, and the structure type as 7. The program will compare the entered structure type 7 with the structure type programmed in the geometry file. If they are equal to 7, the program will calculate the deflections and save the results in a deflection file as shown in figure 7(e).

time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	uf_sg0	uf_sg1	uf_sg2	uf_sg3	uf_sg4	uf_sg5	uf_sg6	uf_sg7	uf_sg8
8:23:15:101	0	0.09934	0.401551	0.918432	1.66528	2.654886	3.890359	5.355407	6.986056	0	-0.09933	-0.40154	-0.91842	-1.66528	-2.6549	-3.89038	-5.35525	-6.9861
8:23:15:301	0	0.101291	0.409535	0.936864	1.698972	2.709118	3.970966	5.468954	7.13876	0	-0.08847	-0.37186	-0.87205	-1.60611	-2.58719	-3.81886	-5.28518	-6.92235
8:23:15:501	0	0.103242	0.41752	0.955297	1.732664	2.763349	4.051573	5.582502	7.291465	0	-0.09004	-0.37832	-0.88698	-1.6334	-2.6311	-3.88412	-5.37706	-7.0457
8:23:15:701	0	0.105193	0.425504	0.97373	1.766356	2.817581	4.13218	5.69605	7.44417	0	-0.09162	-0.38478	-0.90191	-1.66069	-2.67502	-3.94938	-5.46893	-7.16905
8:23:15:901	0	0.107144	0.433489	0.992163	1.800048	2.871812	4.212787	5.809597	7.596875	0	-0.09319	-0.39124	-0.91684	-1.68798	-2.71893	-4.01464	-5.56081	-7.29241
8:23:16:101	0	0.109095	0.441473	1.010595	1.833739	2.926044	4.293394	5.923145	7.74958	0	-0.09476	-0.3977	-0.93177	-1.71527	-2.76285	-4.0799	-5.65269	-7.41576

N7_Deflections

190107

Figure 7(e). Type 7 deflection file.

Type 7 Slope File

Similar to the deflections, the program will calculate the slopes. The results will be saved in a slope file as shown in figure 7(f).

time	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	uf_sg0	uf_sg1	uf_sg2	uf_sg3	uf_sg4	uf_sg5	uf_sg6	uf_sg7	uf_sg8
8:23:15:101	0	0.253874	0.518389	0.801302	1.103473	1.417477	1.725581	1.994857	2.118148	0	-0.25386	-0.51839	-0.80131	-1.10349	-1.41749	-1.72559	-1.99439	-2.11918
8:23:15:301	0	0.258896	0.528804	0.817577	1.126156	1.447213	1.763158	2.04139	2.169718	0	-0.23138	-0.4948	-0.78246	-1.08982	-1.40952	-1.72396	-1.99984	-2.12857
8:23:15:501	0	0.263917	0.539219	0.833852	1.148839	1.476949	1.800736	2.087923	2.221288	0	-0.23544	-0.50324	-0.79565	-1.10818	-1.43359	-1.75437	-2.03733	-2.16999
8:23:15:701	0	0.268938	0.549634	0.850126	1.171521	1.506686	1.838313	2.134456	2.272858	0	-0.2395	-0.51168	-0.80884	-1.12655	-1.45765	-1.78479	-2.07482	-2.21142
8:23:15:901	0	0.273959	0.560049	0.866401	1.194204	1.536422	1.875891	2.180989	2.324428	0	-0.24356	-0.52013	-0.82203	-1.14491	-1.48172	-1.8152	-2.11231	-2.25284
8:23:16:101	0	0.27898	0.570464	0.882675	1.216887	1.566158	1.913468	2.227522	2.375998	0	-0.24761	-0.52857	-0.83522	-1.16328	-1.50578	-1.84561	-2.1498	-2.29426

N7_Slopes

190108

Figure 7(f). Type 7 slope file.

Type 7 Depth Factor File

Similar to the deflections and slopes, the program will calculate the depth factors. The results will be saved in a depth factor file as shown in figure 7(g). After the c_i are calculated from this program, users can use structure type 6 with only one strain-sensing line.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	lf_sg0	lf_sg1	lf_sg2	lf_sg3	lf_sg4	lf_sg5	lf_sg6	lf_sg7	lf_sg8	uf_sg0	uf_sg1	uf_sg2	uf_sg3	uf_sg4	uf_sg5	uf_sg6	uf_sg7	uf_sg8
2	5.29802	4.97504	4.59052	4.21504	3.83774	3.479	3.10436	2.66489	2.35444	6.60788	6.10399	5.66165	5.21026	4.7607	4.29257	3.84035	3.45295	2.93654

N7_DepthFactors

190109

Figure 7(g). Type 7 depth factor file.

Type 7 Output Files

Type 7 does not have twist angles. Type 7 output files are a deflection file, a slope file, a depth factor file, and a max min deflection file.

Final Remarks

There have been many NASA/TPs and NASA/TMs written and published about the Displacement Theory throughout the years. The Displacement Transfer Functions were derived for many structure types. The shape prediction accuracy of the Displacement Theory was analytically validated by finite-element analysis of the Ikhana wing (General Atomics Aeronautical Systems Inc., Poway, California) (ref. 13). The Displacement Theory was also experimentally validated using real-time strain data recorded from the ground loads tests performed in the Flight Load Laboratory at the NASA Armstrong Flight Research Center with full-scale Global Observer (AeroVironment Inc., Monrovia, California) aircraft wings (ref.14) and the GIII (Gulfstream Aerospace, Savannah, Georgia) swept wing structure (ref. 15). In order for users to apply the Displacement Transfer Functions without requiring deep knowledge of the Displacement Theory, the Structure Deformation Calculation Program was written and completed. This program will output the out-of-plane deflections, slopes, cross-sectional twist angles, and depth factors based on the structure type. The outputs of this program can be plotted for all strain-sensing stations in one time slice, one strain-sensing station in all time slices, or all strain-sensing stations in all time slices. This program is versatile and can be applied to a wide range of structures such as aircraft and spacecraft (wing, tail, and fuselage), ships (slab, plate, beam, and truss), skyscrapers, radio towers, bridges, and windmills. The data outputs by the program can be used to monitor the integrity of a structure, and appropriate actions would be made if the structure shows weakness that may cause serious safety issues.

Appendix A: Program Flowchart

The Structure Deformation Calculation Program is written for 7 structure types, it is important that users know their structure types. Each structure type requires different geometry information and different strain arrangement. The program flowchart is displayed in figures A1 and A2.

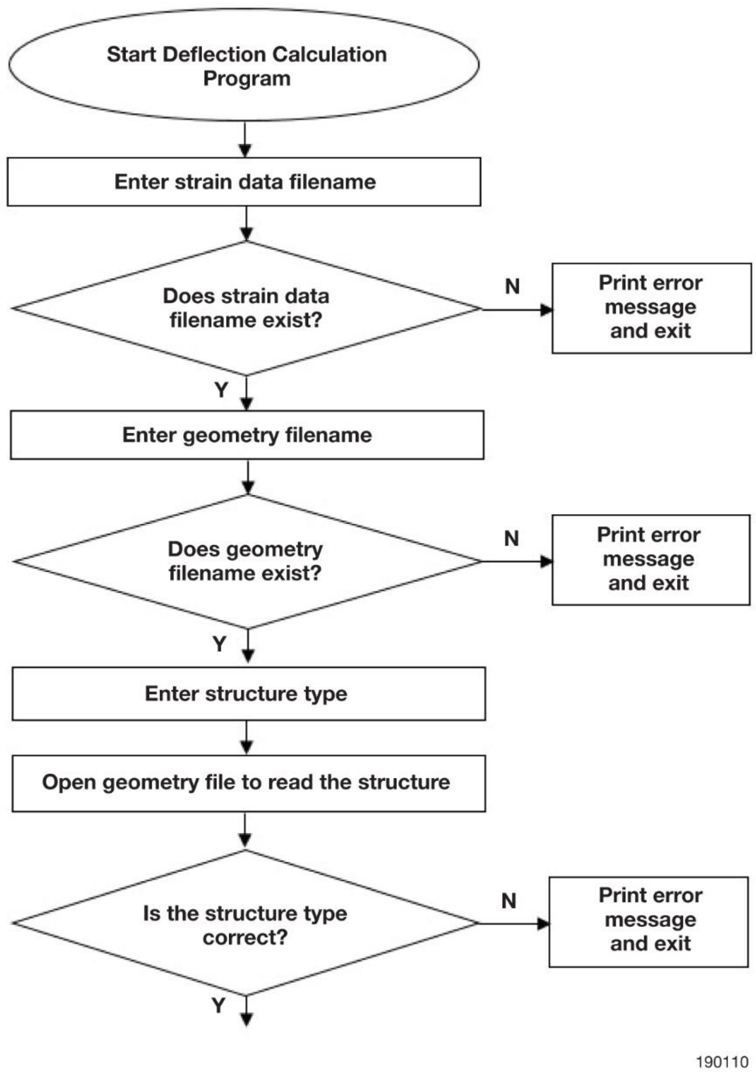


Figure A1. Flowchart of the start of the Structure Deformation Calculation Program.

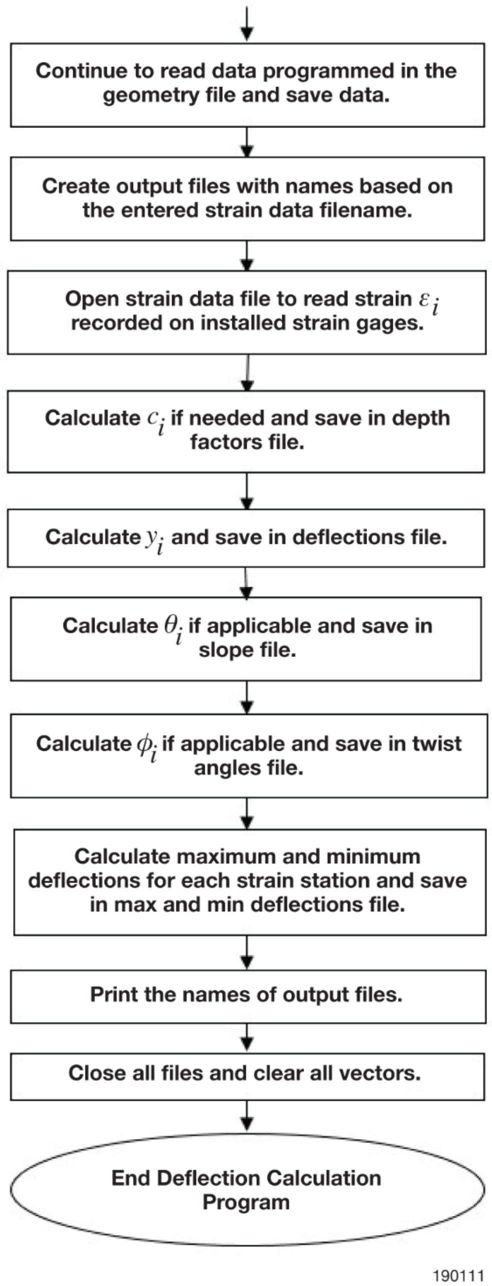


Figure A2. Flowchart of the end of the Structure Deformation Calculation Program.

Appendix B: Program Header File

```
/*******************************************************************************
* TITLE: DisplacementCalculation.h - Structure Deformation Calculation Program Header      *
*                                                                              *
* Written by:     Van Tran Fleischer                                           *
* Title:          Electronics Engineer                                         *
* Date:           September 13, 2017                                           *
* Version:        1                                                            *
* Organization: Advanced Systems Development Branch, Code 540                  *
* Center:         NASA Armstrong Flight Research Center                        *
*                                                                              *
* INTRODUCTION:                                                                *
*                                                                              *
*  This file contains C++ include files, functions, constants and variables used  *
*   in DisplacementCalculation.cpp.                                            *
*                                                                              *
*******************************************************************************/

#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
#include <iomanip>
#include <vector>
using namespace std;

double   asin(double x);
double   tan(double x);
double   atan(double x);
double   sqrt(double x);
double   log(double x);
double   pow(double x, double y);

int   GetUserInputs();
int   ReadGeometryFile();
int   ReadType1_2_6();
int   ReadType3();
int   ReadType4();
int   ReadType5();
int   ReadType7();
int   CreateOutputFiles();
int   CalcDisplacement();
void  CalculateC();
void  CalcTwistAngles();
void  DetermineMaxMin();
void  WriteMaxMinFile();
void  CloseFiles_ClearVectors();

ifstream inFile;
```

```
ifstream geoFile;
ofstream outFile;
ofstream thetaFile;
ofstream phiFile;
ofstream maxminFile;
ofstream cFile;

string inputFile;
string ingeoFile;
string outputFile;
string outthetaFile;
string outphiFile;
string outmaxminFile;
string outcFile;

vector<string> stationNames, tMax, tMin;
vector<double> epsilon, deltaL, x, y, yB, yMax, yMin;
vector<double> theta, tan_theta, phi, sin_phi;
vector<double> c, d, h;

const   int         MAX_LINE = 500000;
const   int         TRUE = 1;
const   int         FALSE = 0;
const   int         ERROR = -1;
const   int         OK = 0;
const   int         VAR_DOMAIN = 2;
const   int         TAPERED = 1;
const   double      TPR_RATIO = 0.9;
const   double      PI = 3.1415926535897932;

char    *token, *t;
char    *nextToken = NULL;
char     inBuff[MAX_LINE];

double  C, C0, Cn, C0_prime, Cn_prime, D0, Dn, H0, Hn, H0_prime, Hn_prime,
        strain, length, const_deltaL, H_ratio, Hprime_ratio;

int     calC = FALSE, checked = OK;

unsigned int i, j, k, n, structureType, structType, cType, domainType, numLines,
         nStations, numStations, noStations, first_time = 1, phiCreated = 0,
         cCreated = 0, lineNum = 0;
```

Appendix C: Program Code in C++

```
/******************************************************************************
* TITLE: DisplacementCalculation.cpp – Structure Deformation Calculation Program        *
*                                                                                        *
* Written by:    Van Tran Fleischer                                                      *
* Title:         Electronics Engineer                                                    *
* Date:          September 13, 2017                                                      *
* Version:        1                                                                      *
* Organization: Advanced Systems Development Branch, Code 540                            *
* Center:         NASA Armstrong Flight Research Center                                  *
*                                                                                        *
* INTRODUCTION:                                                                          *
*                                                                                        *
*    In order to use this program, users must prepare a .csv file that                   *
*    contains time and strain values with the following format:                          *
*                                                                                        *
*    The 1st line of the .csv file should contain the headers that contains              *
*    Time and Strain-sensing stations names.                                             *
*                                                                                        *
*    The second line and thereafter should have time and strain data for                 *
*    each strain-sensing station on the structure.                                       *
*                                                                                        *
*    -----------------------------------------------------------------------------------  *
*    | Time               | SG1 Name | SG2 Name | SG.. Name  | SGn Name |                 *
*    -----------------------------------------------------------------------------------  *
*    |076 09:04:15.313  | 0.01155    | 0.01173   |     ....    | 0.01125   |            *
*    -----------------------------------------------------------------------------------  *
*    |076 09:04:15.363  | 0.01164    | 0.01181   |     ....    | 0.01137   |            *
*    -----------------------------------------------------------------------------------  *
*    |076 09:04:15.413  | 0.01172    | 0.01187   |     ....    | 0.01146   |            *
*    -----------------------------------------------------------------------------------  *
*    |076 09:04:15.463  | 0.01187    | 0.01194   |     ....    | 0.01158   |            *
*    -----------------------------------------------------------------------------------  *
*                                                                                        *
* Users must prepare a .txt file that contains the data about structure.                 *
* The format of this file is different based on structure type.                          *
*                                                                                        *
* This program will prompt users for the names of two files and structure type.          *
*                                                                                        *
* 1. Strain data input filename in .csv extension. This file must be located in the same      *
*    directory as the DisplacementCalculation.exe.                                       *
*                                                                                        *
* 2. Geometry input filename in .txt format. This file must be located in the same       *
*     directory as the DisplacementCalculation.exe program.                              *
*                                                                                        *
* 3. Enter structure type:                                                               *
*     1 for uniform or tapered cantilever embedded beam with 1 strain line               *
*     2 for two-end supported embedded beam                                              *
*     3 for wing box with 2 strain lines & known c                                       *
```

```
*     4 for doubly wing box with four-line system & unknown c                                      *
*     5 for thin uniform plate                                                                      *
*     6 for curved deformation of long tapered cantilever beam                                      *
*     7 for curved deformation of long nonlinear beam                                               *
*                                                                                                   *
* NOMENCLATURE used in the program:                                                                 *
*                                                                                                   *
* C: depth factor of uniform beam, in.                                                              *
* c[i]: depth factors at strain-sensing station i, x=xi, in.                                        *
* C0, c[0]: value of c[i] at fixed end (beam root) strain-sensing station, in.                      *
* Cn, c[n]: value of c[i] at free end (beam tip) strain-sensing station, in.                        *
* d: chord-wise distance between two span-wise parallel strain lines, in.                           *
* d[i]: chord-wise distance between front strain-sensing stations i and rear strain-sensing         *
* stations i', in.                                                                                  *
* deltaL[i]: distance between strain-sensing stations on a same strain-sensing line i-1 & i , in.   *
* x[i]: distance from the fixed end to the i-th strain-sensing station, in.                         *
* y[i]: deflection at strain-sensing station i, in.                                                 *
* theta[i], θ[i]: slope of deformed beam at strain-sensing station i, deg                           *
* phi[i], Φ[i]: cross-sectional twist angle at strain-sensing station i, deg                        *
*                                                                                                   *
* REVISION HISTORY:                                                                                 *
*                                                                                                   *
* Initial Release: September 13, 2017                                                               *
*                                                                                                   *
* Revisions:                                                                                        *
*                                                                                                   *
*****************************************************************************************************/

#include "DisplacementCalculation.h"

int CalcDisplacement()
{
    double term1, term2, term3, term4;

    // Clear epsilon arrays
    epsilon.clear();

    // Read a line of data in the Strain input file
    while (inFile.getline(inBuff, MAX_LINE))
    {
        // Read time for the current time slice
        token = strtok_s(inBuff, " ,\t\n", &nextToken);

        // First value is time
        if (token)
        {
            // Save time
            t = token;

            // Write time to deflection output file
            outFile << t << ",";
```

```
        // Write time to slope output file
        if ((structureType != 2) && (structureType != 5))
        {
            thetaFile << t << ",";
        }

        // Write time to twist angle output file
        if ((structureType == 3) || (structureType == 4))
            phiFile << t << ",";
    }

    // Read the input strains for the current time slice
    while (token)
    {
        token = strtok_s(0, " ,\t\n", &nextToken);

        if (token)
        {
            strain = atof(token);
            epsilon.push_back(strain);
            y.push_back(0);
            yB.push_back(0);

            if ((structureType != 2) && (structureType != 5))
            {
                theta.push_back(0);
                tan_theta.push_back(0);
            }

            if ((structureType == 3) || (structureType == 4))
            {
                phi.push_back(0);
                sin_phi.push_back(0);
            }
        }
    }

    switch (structureType)
    {
    case 1: // uniform or tapered cantilever beam

        // Set deflection and slope at the fixed end
        y[0] = 0.0;
        theta[0] = 0.0;

        // Write to deflection and slope output files
        outFile << fixed << setprecision(6) << y[0];
        thetaFile << fixed << setprecision(6) << theta[0];
```

```
    // Uniform
    if (C0 == Cn)
    {
        for (i = 1; i < numStations; i++)
        {
            // Eq. (5a) in this paper or Eq. (24) in NASA/TP-2009-214643
            term1 = epsilon[i - 1] + epsilon[i];
            tan_theta[i] = (deltaL[i] / (2.0*C0)) * term1 + tan_theta[i - 1];
            theta[i] = atan(tan_theta[i]) * 180.0 / PI;

            // Eq. (5b) in this paper or Eq. (26) in NASA/TP-2009-214643
            term2 = (2.0*epsilon[i - 1]) + epsilon[i];
            y[i] = (deltaL[i] * deltaL[i] / (6.0*C0)) * term2 + y[i - 1] +
                                deltaL[i] * tan_theta[i - 1];

            // Write to deflection and slope output files
            outFile << "," << fixed << setprecision(6) << y[i];
            thetaFile << "," << fixed << setprecision(6) << theta[i];
        }
    }
    else if ((c[1] / C0 > TPR_RATIO) && (Cn / c[n-1] > TPR_RATIO))
{
    // Slightly Tapered
        for (i = 1; i < numStations; i++)
        {
            // Eq. (4a) in this paper or Eq. (14a) in NASA/TP-2015-218464
            term1 = (2.0 - (c[i] / c[i - 1])) * epsilon[i - 1] + epsilon[i];
            tan_theta[i] = (deltaL[i] / (2.0*c[i - 1])) * term1 + tan_theta[i - 1];
            theta[i] = atan(tan_theta[i]) * 180.0 / PI;

            // Eq. (4b) in this paper or Eq. (14b) in NASA/TP-2015-218464
            term2 = (3.0 - (c[i] / c[i - 1])) * epsilon[i - 1] + epsilon[i];
            y[i] = (deltaL[i] * deltaL[i] / (6.0*c[i - 1])) * term2 + y[i - 1] +
                                deltaL[i] * tan_theta[i - 1];

            // Write to deflection and slope output files
            outFile << "," << fixed << setprecision(6) << y[i];
            thetaFile << "," << fixed << setprecision(6) << theta[i];
        }
}
else
    {
        // Nonuniform
        for (i = 1; i < numStations; i++)
        {
            // Eq. (3a) in this paper or Eq. (13a) in NASA/TP-2015-218464
            term1 = (epsilon[i - 1] - epsilon[i]) / (c[i - 1] - c[i]);
            term2 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1]) *
                    log(c[i] / c[i - 1]) / pow((c[i - 1] - c[i]), 2);
            tan_theta[i] = deltaL[i] * (term1 + term2) + tan_theta[i - 1];
            theta[i] = atan(tan_theta[i]) * 180.0 / PI;
```

```
            // Eq. (3b) in this paper or Eq. (13b) in NASA/TP-2015-218464
            term3 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1]) /
                                    pow((c[i - 1] - c[i]), 3);
            term4 = c[i] * log(c[i] / c[i - 1]) + (c[i - 1] - c[i]);
            y[i] = deltaL[i] * deltaL[i] * ((term1 / 2.0) - term3*term4) +
                                      y[i - 1] + deltaL[i] * tan_theta[i - 1];

            // Write to deflection and slope output files
            outFile << "," << fixed << setprecision(6) << y[i];
            thetaFile << "," << fixed << setprecision(6) << theta[i];
        }
    }
    break;

case 2: // two-end supported

    // Set values at the selected fixed end
    y[0] = 0.0;
    yB[0] = 0.0;

    // Write to deflection output file
    outFile << fixed << setprecision(6) << yB[0];

    // Calculate deflection y
    for (i = 1; i < numStations; i++)
    {
        term1 = 0.0;
        term2 = 0.0;

        // Eq. (6) in this paper or Eq. (36) in NASA/TP-2009-214643
        for (j = 1; j <= i; j++)
        {
            term2 = (1.0 / c[i - j]) *
                ((3.0 * (2.0*j - 1.0) - ((3.0*j - 2.0)* c[i - j + 1] / c[i - j])) * epsilon[i - j]
                    + (3.0*j - 2.0) * epsilon[i - j + 1]);
            term1 += term2;
        }

        y[i] = deltaL[i] * deltaL[i] * term1 / 6.0;
    }

    // Calculate deflection yB
    for (i = 1; i < numStations; i++)
    {
        // yB = y - the correction term
        yB[i] = y[i] - (x[i] / length * y[n]);

        // Write to deflection output file
        outFile << "," << fixed << setprecision(6) << yB[i];
```

```
    }
    break;

case 3: // two-line system

    // Calculate deflections and slopes
    for (j = 0; j < 2; j++)
    {
        // Set values at the fixed end
        y[j*nStations] = 0.0;
        theta[j*nStations] = 0.0;

        // Write to deflection and slope output files
        outFile << fixed << setprecision(6) << y[j*nStations];
        thetaFile << fixed << setprecision(6) << theta[j*nStations];

        for (i = j*nStations + 1; i < (j*nStations + nStations); i++)
        {
            // Eq. (3a) in this paper or Eq. (13a) in NASA/TP-2015-218464
            term1 = (epsilon[i - 1] - epsilon[i]) / (c[i - 1] - c[i]);
            term2 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1])*log(c[i] / c[i - 1]) /
                                                        pow((c[i - 1] - c[i]), 2);
            tan_theta[i] = deltaL[i - j*nStations] * (term1 + term2) + tan_theta[i - 1];
            theta[i] = atan(tan_theta[i]) * 180.0 / PI;

        // Eq. (3b) in this paper or Eq. (13b) in NASA/TP-2015-218464
            term3 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1]) / pow((c[i - 1] - c[i]), 3);
            term4 = c[i] * log(c[i] / c[i - 1]) + (c[i - 1] - c[i]);
            y[i] = deltaL[i - j*nStations] * deltaL[i - j*nStations] * ((term1 / 2.0) -
            term3*term4) + y[i - 1] + deltaL[i - j*nStations] * tan_theta[i - 1];

            // Write to deflection and slope output files
            outFile << "," << fixed << setprecision(6) << y[i];
            thetaFile << "," << fixed << setprecision(6) << theta[i];
        }

        // Write "," in output files
        outFile << ",";
        thetaFile << ",";
    }

    // Calculate twist angles
    CalcTwistAngles();
    break;

case 4: // 4-line system

    // Calculate c[i]
    if (calC == FALSE)
    {
        CalculateC();
```

```
        }

        // Calculate deflections and slopes
        for (j = 0; j < 4; j++)
        {
            // Set values at the fixed end
            y[j*nStations] = 0.0;
            theta[j*nStations] = 0.0;

            // Write to deflection and slope output files
            outFile << fixed << setprecision(6) << y[j*nStations];
            thetaFile << fixed << setprecision(6) << theta[j*nStations];

            for (i = j*nStations + 1; i < (j*nStations + nStations); i++)
            {
                // Eq. (3a) in this paper or Eq. (13a) in NASA/TP-2015-218464
                term1 = (epsilon[i - 1] - epsilon[i]) / (c[i - 1] - c[i]);
                term2 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1])*log(c[i] / c[i - 1]) /
                                                            pow((c[i - 1] - c[i]), 2);
                tan_theta[i] = deltaL[i - j*nStations] * (term1 + term2) + tan_theta[i - 1];
                theta[i] = atan(tan_theta[i]) * 180.0 / PI;

                // Eq. (3b) in this paper or Eq. (13b) in NASA/TP-2015-218464
                term3 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1]) / pow((c[i - 1] - c[i]), 3);
                term4 = c[i] * log(c[i] / c[i - 1]) + (c[i - 1] - c[i]);
                y[i] = deltaL[i - j*nStations] * deltaL[i - j*nStations] * ((term1 / 2.0) -
                term3*term4) + y[i - 1] + deltaL[i - j*nStations] * tan_theta[i - 1];

                // Write to deflection and slope output files
                outFile << "," << fixed << setprecision(6) << y[i];
                thetaFile << "," << fixed << setprecision(6) << theta[i];
            }

            // Write "," in output files
            outFile << ",";
            thetaFile << ",";
        }

        // Calculate twist angles
        CalcTwistAngles();
        break;

    case 5: // Thin uniform plate

        // Calculate deflections
        for (j = 0; j < numLines; j++)
        {
            // Set values at the selected fixed end
            y[j*nStations] = 0.0;
            yB[j*nStations] = 0.0;
```

```
            // Write to deflection output file
            outFile << fixed << setprecision(6) << yB[j*nStations];

            // Calculate y deflections
            for (i = j*nStations + 1; i < (j*nStations + nStations); i++)
            {
                term1 = 0.0;
                term2 = 0.0;

                // Eq. (6) in this paper or Eq. (36) in NASA/TP-2009-214643
                for (k = 1; k <= (i-j*nStations); k++)
                {
                    term2 = (1.0 / C) *
                        ((3.0 * (2.0*k - 1.0) - (3.0*k - 2.0)) * epsilon[i - k]
                            + (3.0*k - 2.0) * epsilon[i - k + 1]);
                    term1 += term2;
                }

                y[i] = deltaL[i - j*nStations] * deltaL[i - j*nStations] * term1 / 6.0;
            }

            // Calculate yB deflections
            for (i = j*nStations + 1; i < (j*nStations + nStations); i++)
            {
                // yB = y - the correction term
                yB[i] = y[i] - (x[i-j*nStations] / length * y[n + j*nStations]);

                // Write to deflection output file
                outFile << "," << fixed << setprecision(6) << yB[i];
            }

            // Write "," in output file
            outFile << ",";
        }
        break;

    case 6: // curved deformation of 1 line long tapered beam

        // Set deflection and slope at the fixed end
        y[0] = 0.0;
        theta[0] = 0.0;

        // Write to deflection and slope output files
        outFile << fixed << setprecision(6) << y[0];
        thetaFile << fixed << setprecision(6) << theta[0];

        // Calculate curved deflections of tapered beam

    // Uniform
        if (C0 == Cn)
```

```
{
    for (i = 1; i < numStations; i++)
    {
        // Eq. (9a) in this paper or Eq. (18a) in NASA/TP-2017-219406
        term1 = epsilon[i - 1] + epsilon[i];
        theta[i] = (deltaL[i] / (2.0*C0)) * term1 + theta[i - 1];

        // Eq. (9b) in this paper or Eq. (18b) in NASA/TP-2017-219406
        term2 = (2.0*epsilon[i - 1]) + epsilon[i];
        y[i] = (deltaL[i] * deltaL[i] / (6.0*C0)) * term2 + y[i - 1] +
                                                    deltaL[i] * theta[i - 1];

        // Write to deflection and slope output files
        outFile << "," << fixed << setprecision(6) << y[i];
        thetaFile << "," << fixed << setprecision(6) << theta[i] * 180.0 / PI;
    }
}
else if ((c[1] / C0 > TPR_RATIO) && (Cn / c[n - 1] > TPR_RATIO))
{
    // Slightly Nonuniform Tapered
    for (i = 1; i < numStations; i++)
    {
        // Eq. (8a) in this paper or Eq. (18) in NASA/TP-2009-214643
        term1 = (2.0 - (c[i] / c[i - 1])) * epsilon[i - 1] + epsilon[i];
        tan_theta[i] = (deltaL[i] / (2.0*c[i - 1])) * term1 + theta[i - 1];

        // Eq. (8b) in this paper or Eq. (21) in NASA/TP-2009-214643
        term2 = (3.0 - (c[i] / c[i - 1])) * epsilon[i - 1] + epsilon[i];
        y[i] = (deltaL[i] * deltaL[i] / (6.0*c[i - 1])) * term2 + y[i - 1] +
            deltaL[i] * theta[i - 1];

        // Write to deflection and slope output files
        outFile << "," << fixed << setprecision(6) << y[i];
        thetaFile << "," << fixed << setprecision(6) << theta[i] * 180.0 / PI;
    }
}
else
{
    // Nonuniform Curved Deformation
    for (i = 1; i < numStations; i++)
    {
        // Eq. (7a) in this paper or Eq. (18a) in NASA/TP-2017-219406
        term1 = (epsilon[i - 1] - epsilon[i]) / (c[i - 1] - c[i]);
        term2 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1])*log(c[i] / c[i - 1]) /
                                                            pow((c[i - 1] - c[i]), 2);
        theta[i] = deltaL[i] * (term1 + term2) + theta[i - 1];

        // Eq. (7b) in this paper or Eq. (18b) in NASA/TP-2017-219406
        term3 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1]) / pow((c[i - 1] - c[i]), 3);
        term4 = c[i] * log(c[i] / c[i - 1]) + (c[i - 1] - c[i]);
```

```
                y[i] = deltaL[i] * deltaL[i] * ((term1 / 2.0) - term3*term4) + y[i - 1] +
                                                                      deltaL[i] * theta[i - 1];

                // Write to deflection and slope output files
                outFile << "," << fixed << setprecision(6) << y[i];
                thetaFile << "," << fixed << setprecision(6) << theta[i] * 180.0 / PI;
            }
        }
        break;

    case 7: // curved deformation for 2 lines, lower and upper; unknown c

        // Calculate c[i]
        if (calC == FALSE)
        {
            CalculateC();
        }

        // Calculate nonlinear large deflections and slopes for a long nonuniform structure
        for (j = 0; j < 2; j++)
        {
            // Set values at the fixed end
            y[j*nStations] = 0.0;
            theta[j*nStations] = 0.0;

            // Write to deflection and slope output files
            outFile << fixed << setprecision(6) << y[j*nStations];
            thetaFile << fixed << setprecision(6) << theta[j*nStations];

            for (i = j*nStations + 1; i < (j*nStations + nStations); i++)
            {
                // Eq. (7a) in this paper or Eq. (18a) in NASA/TP-2017-219406
                term1 = (epsilon[i - 1] - epsilon[i]) / (c[i - 1] - c[i]);
                term2 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1])*log(c[i] / c[i - 1]) /
                                                                      pow((c[i - 1] - c[i]), 2);
                theta[i] = deltaL[i - j*nStations] * (term1 + term2) + theta[i - 1];

                // Eq. (7b) in this paper or Eq. (18b) in NASA/TP-2017-219406
                term3 = (epsilon[i - 1] * c[i] - epsilon[i] * c[i - 1]) / pow((c[i - 1] - c[i]), 3);
                term4 = c[i] * log(c[i] / c[i - 1]) + (c[i - 1] - c[i]);
                y[i] = deltaL[i - j*nStations] * deltaL[i - j*nStations] * ((term1 / 2.0) –
                           term3*term4) + deltaL[i - j*nStations] * theta[i - 1] + y[i - 1];

                // Write to deflection and slope output files
                outFile << "," << fixed << setprecision(6) << y[i];
                thetaFile << "," << fixed << setprecision(6) << theta[i] * 180.0 / PI;
            }

            // Write "," in output files
            outFile << ",";
            thetaFile << ",";
```

```
    }
    break;

} // switch

// Put the end of line to output files
outFile << endl;

if ((structureType != 2) && (structureType != 5))
{
    thetaFile << endl;
}

if ((structureType == 3) || (structureType == 4))
    phiFile << endl;

// Initialize vectors yMax, yMin, tMax and tMin for the first time
if (first_time == 1)
{
    if ((structureType != 2) && (structureType != 5))
    {
        for (i = 0; i < numStations; i++)
        {
            yMax[i] = y[i];
            yMin[i] = y[i];

            tMax[i] = t;
            tMin[i] = t;
        }
    }
    else
    {
        for (i = 0; i < numStations; i++)
        {
            yMax[i] = yB[i];
            yMin[i] = yB[i];

            tMax[i] = t;
            tMin[i] = t;
        }
    }

    first_time = 0;
}

// Determin max and min deflections
DetermineMaxMin();

// Clear vectors
epsilon.clear();
```

```
        y.clear();
        yB.clear();
        theta.clear();
        phi.clear();
    }
    return(OK);
}

int GetUserInputs()
{
    // Get required data from user
    cout << "$ Enter Strain Data filename: ";
    cin >> inputFile;

    // Open input file
    inFile.open(inputFile.c_str(), ios::in);
    if (inFile.fail())
    {
        cerr << "Could not open " << inputFile << "!\n";
        cerr << "Please check your Strain Data filename!\n\n";
        checked = ERROR;
        return(ERROR);
    }

    cout << "$ Enter Geometry filename: ";
    cin >> ingeoFile;

    // Open geometry input file
    geoFile.open(ingeoFile.c_str(), ios::in);
    if (geoFile.fail())
    {
        cerr << "Could not open " << ingeoFile << "!\n";
        cerr << "Please check your Geometry filename!\n\n";
        checked = ERROR;
        return(ERROR);
    }

    cout << "\nStructure Type:\n";
    cout << "1 for uniform or tapered beam with 1-line system.\n";
    cout << "2 for two-end supported.\n";
    cout << "3 for 2-line system.\n";
    cout << "4 for 4-line system.\n";
    cout << "5 for thin uniform plate.\n";
    cout << "6 for curved deformation of long tapered beam.\n";
cout << "7 for curved deformation of long nonlinear beam.\n";

// Prompt for structure type from user
    cout << "\n$ Enter structure type: ";
    cin >> structureType;

    if ((structureType < 1) || (structureType > 7))
```

```
    {
        cout << "\nThe entered structure type is not valid! Must be from 1 to 7!\n";
        cerr << "Please rerun the program and enter a valid structure type!\n\n";
        checked = ERROR;
        return(ERROR);
    }
    return(OK);
}

int ReadGeometryFile()
{
    // Check structureType to load geometry data correctly
    switch (structureType)
    {
    case 1: // uniform & tapered
    case 2: // two-end supported
    case 6: // curved deformation of long tapered beam

        ReadType1_2_6();
        break;

    case 3: // 2 lines and known c

        ReadType3();
        break;

    case 4: // 4 lines and unknown c

        ReadType4();
        break;

    case 5: // 2-point supported

        ReadType5();
        break;

    case 7: // nonuniform curved deflection, 2 lines, lower and upper, unknown c

        ReadType7();
        break;

    } // switch structureType

    // Check structure type
    if (structType != structureType)
    {
        cerr << "\nThe entered structure type is different from the one in the Geometry file!\n";
        cerr << "Please Check Geometry File " << ingeoFile << "!\n\n";
        checked = ERROR;
        return(ERROR);
```

```
    }

    // Initialize vectors yMax, yMin, tMax, and tMin
    for (i = 0; i < numStations; i++)
    {
        yMax.push_back(0);
        yMin.push_back(0);
        tMax.push_back("0");
        tMin.push_back("0");
    }
    return(OK);
}

int ReadType1_2_6()
{
    lineNum = 0;

    // Read the 1st line of geometry file
    while (geoFile.getline(inBuff, MAX_LINE))
    {
        lineNum++;

        switch (lineNum)
        {
        // Read data
        case 1:   // Read line No. 1

            // Read structure type
            token = strtok_s(inBuff, " ,\t\n", &nextToken);
            structType = atoi(token);

            // Read structure length
            token = strtok_s(0, " ,\t\n", &nextToken);
            length = atof(token);

            // Read number of strain-sensing stations
            token = strtok_s(0, " ,\t\n", &nextToken);
            numStations = atoi(token);

            // Read depth factor at the fixed end C0
            token = strtok_s(0, " ,\t\n", &nextToken);
            C0 = atof(token);

            // Read depth factor at the free end Cn
            token = strtok_s(0, " ,\t\n", &nextToken);
            Cn = atof(token);

            // Calculate n
            n = numStations - 1;
            break;
```

```
case 2:   // Read line No. 2

    // Read domain type
    token = strtok_s(inBuff, " ,\t\n", &nextToken);
    domainType = atoi(token);

    // Push the index of c, deltaL, and x
    c.push_back(C0);
    deltaL.push_back(0);
    x.push_back(0);

    if (domainType == VAR_DOMAIN)
    {
        // Read deltaL
        token = strtok_s(0, " ,\t\n", &nextToken);
        while (token)
        {
            deltaL.push_back(atof(token));
            c.push_back(0);
            x.push_back(0);

            // read next deltaL
            token = strtok_s(0, " ,\t\n", &nextToken);
        }

        // Calculate Xi and Ci for variable domain
        for (i = 1; i < numStations; i++)
        {
            x[i] = x[i - 1] + deltaL[i];
            c[i] = C0 - (C0 - Cn)* (x[i] / length);
        }
    }
    else
    {
        const_deltaL = length / double(n);

        for (i = 1; i < numStations; i++)
        {
            deltaL.push_back(const_deltaL);
            x.push_back(0);
            c.push_back(0);
        }

        // Calculate Xi and Ci for constant domain
        for (i = 1; i < numStations; i++)
        {
            x[i] = (double) i * const_deltaL;
            c[i] = C0 - (C0 - Cn)* (x[i] / length);
        }
    }
```

```
                break;

            } // switch (lineNum)
        } // while

        // Close geoFile
        geoFile.close();
        return(OK);
}

int ReadType3()
{
        lineNum = 0;

        // read the 1st line of geometry file
        while (geoFile.getline(inBuff, MAX_LINE))
        {
            lineNum++;

            switch (lineNum)
            {
            // Read data
            case 1:  // Read line No. 1

                // Read structure type
                token = strtok_s(inBuff, " ,\t\n", &nextToken);
                structType = atoi(token);

                // Read structure length
                token = strtok_s(0, " ,\t\n", &nextToken);
                length = atof(token);

                // Read number of strain-sensing stations
                token = strtok_s(0, " ,\t\n", &nextToken);
                numStations = atoi(token);

                // Read chord-wise distant at root D0
                token = strtok_s(0, " ,\t\n", &nextToken);
                D0 = atof(token);

                // Read chord-wise distant at tip Dn
                token = strtok_s(0, " ,\t\n", &nextToken);
                Dn = atof(token);

                // Calculate n
                nStations = numStations / 2;
                n = nStations - 1;
                break;

            case 2:   // Read line No. 2
```

```
        // Read domain type
        token = strtok_s(inBuff, " ,\t\n", &nextToken);
        domainType = atoi(token);

        // Push the index of deltaL and x
        deltaL.push_back(0);
        x.push_back(0);

        if (domainType == VAR_DOMAIN)
        {
            // Read deltaL
            token = strtok_s(0, " ,\t\n", &nextToken);
            while (token)
            {
                deltaL.push_back(atof(token));
                x.push_back(0);

                // read next deltaL
                token = strtok_s(0, " ,\t\n", &nextToken);
            }

            // Calculate Xi for variable domain
            x[0] = 0.0;
            for (i = 1; i < nStations; i++)
            {
                x[i] = x[i - 1] + deltaL[i];
            }
        }
        else
        {
            const_deltaL = length / double(n);
            for (i = 1; i < nStations; i++)
            {
                deltaL.push_back(const_deltaL);
                x.push_back(0);
            }

            // Calculate Xi for constant domain
            x[0] = 0.0;
            for (i = 1; i < nStations; i++)
            {
                x[i] = ((double)i) * const_deltaL;
            }
        }
        break;

    case 3:   // Read line No. 3

        // Read cType
        token = strtok_s(inBuff, " ,\t\n", &nextToken);
```

```
cType = atoi(token);

// Check if tapered
if (cType == TAPERED)
{
    // Read C0
    token = strtok_s(0, " ,\t\n", &nextToken);
    C0 = atof(token);

    // Read Cn
    token = strtok_s(0, " ,\t\n", &nextToken);
    Cn = atof(token);

    // Read C0_prime
    token = strtok_s(0, " ,\t\n", &nextToken);
    C0_prime = atof(token);

    // Read Cn_prime
    token = strtok_s(0, " ,\t\n", &nextToken);
    Cn_prime = atof(token);

    // Initialize c, & x vectors
    for (i = 0; i < numStations; i++)
    {
        c.push_back(0);
        x.push_back(0);
    }

    // Calculate Ci for the front
    c[0] = C0;
    for (i = 1; i < nStations; i++)
    {
        c[i] = C0 - (C0 - Cn)* (x[i] / length);
    }

    // Calculate Ci for the rear
    c[nStations] = C0_prime;
    for (i = nStations + 1; i < numStations; i++)
    {
        c[i] = C0_prime - (C0_prime - Cn_prime)* (x[i-nStations] / length);
    }
}
else
{
    // Read c[i]
    token = strtok_s(0, " ,\t\n", &nextToken);
    while (token)
    {
        c.push_back(atof(token));

        // read next c
```

```
                    token = strtok_s(0, " ,\t\n", &nextToken);
                }
            }
            break;

        } // switch (lineNum)
    } // while getline

    // Close geoFile
    geoFile.close();

    // Initialize vectors d & phi
    for (i = 0; i < nStations; i++)
    {
        d.push_back(0);
        phi.push_back(0);
    }
    return(OK);
}

int ReadType4()
{
    lineNum = 0;

    // Read the 1st line of geometry file
    while (geoFile.getline(inBuff, MAX_LINE))
    {
        lineNum++;

        switch (lineNum)
        {
        // Read data
        case 1:   // Read line No. 1

            // Read structure type
            token = strtok_s(inBuff, " ,\t\n", &nextToken);
            structType = atoi(token);

            // Read structure length
            token = strtok_s(0, " ,\t\n", &nextToken);
            length = atof(token);

            // Read number of strain-sensing stations
            token = strtok_s(0, " ,\t\n", &nextToken);
            numStations = atoi(token);

            // Read chord-wise distant at root D0
            token = strtok_s(0, " ,\t\n", &nextToken);
            D0 = atof(token);
```

```
        // Read chord-wise distant at tip Dn
        token = strtok_s(0, " ,\t\n", &nextToken);
        Dn = atof(token);

        // Read wing root depth at front H0
        token = strtok_s(0, " ,\t\n", &nextToken);
        H0 = atof(token);

        // Read wing tip depth at front Hn
        token = strtok_s(0, " ,\t\n", &nextToken);
        Hn = atof(token);

        // Read wing root depth at rear H0_prime
        token = strtok_s(0, " ,\t\n", &nextToken);
        H0_prime = atof(token);

        // Read wing tip depth at rear Hn_prime
        token = strtok_s(0, " ,\t\n", &nextToken);
        Hn_prime = atof(token);

        // Initialize variables
        nStations = numStations / 4;
        noStations = numStations / 2;
        n = nStations - 1;
        break;

    case 2:    // Read line No. 2

        // Read domain type
        token = strtok_s(inBuff, " ,\t\n", &nextToken);
        domainType = atoi(token);

        // Push the index of deltaL and x
        deltaL.push_back(0);
        x.push_back(0);

        if (domainType == VAR_DOMAIN)
        {
            // Read deltaL
            token = strtok_s(0, " ,\t\n", &nextToken);
            while (token)
            {
                deltaL.push_back(atof(token));
                x.push_back(0);

                // read next deltaL
                token = strtok_s(0, " ,\t\n", &nextToken);
            }

            // Calculate Xi for variable domain
            for (i = 1; i < nStations; i++)
```

```
                {
                    x[i] = x[i - 1] + deltaL[i];
                }
            }
            else
            {
                const_deltaL = length / double(n);

                for (i = 1; i < nStations; i++)
                {
                    deltaL.push_back(const_deltaL);
                    x.push_back(0);
                }

                // Calculate Xi for constant domain
                for (i = 1; i < nStations; i++)
                {
                    x[i] = ((double)i) * const_deltaL;
                }
            }
            break;
        } // switch lineNum
    } // while getline

    // Close geoFile
    geoFile.close();

    // Initialize vectors h and c
    for (i = 0; i < numStations; i++)
    {
        h.push_back(0);
        c.push_back(0);
    }

    // Initialize vectors d and phi
    for (i = 0; i < noStations; i++)
    {
        d.push_back(0);
        phi.push_back(0);
    }
    return(OK);
}

int ReadType5()
{
    lineNum = 0;

    // read the 1st line of geometry file
    while (geoFile.getline(inBuff, MAX_LINE))
    {
```

```
lineNum++;

switch (lineNum)
{
// Read data
case 1:   // Read line No. 1

    // Read structure type
    token = strtok_s(inBuff, " ,\t\n", &nextToken);
    structType = atoi(token);

    // Read structure length
    token = strtok_s(0, " ,\t\n", &nextToken);
    length = atof(token);

    // Read number of strain-sensing stations
    token = strtok_s(0, " ,\t\n", &nextToken);
    numStations = atoi(token);

    // Read depth factor of the thin plate
    token = strtok_s(0, " ,\t\n", &nextToken);
    C = atof(token);

    // Read number of lines
    token = strtok_s(0, " ,\t\n", &nextToken);
    numLines = atoi(token);

    // Calculate n
    nStations = numStations / numLines;
    n = nStations - 1;
    break;

case 2:   // Read line No. 2

    // Read domain type
    token = strtok_s(inBuff, " ,\t\n", &nextToken);
    domainType = atoi(token);

    // Push the index of deltaL and x
    deltaL.push_back(0);
    x.push_back(0);

    if (domainType == VAR_DOMAIN)
    {
        // Read deltaL
        token = strtok_s(0, " ,\t\n", &nextToken);
        while (token)
        {
            deltaL.push_back(atof(token));
            x.push_back(0);
```

```
                // Read next deltaL
                token = strtok_s(0, " ,\t\n", &nextToken);
            }

            // Calculate Xi for variable domain
            for (i = 1; i < nStations; i++)
            {
                x[i] = x[i - 1] + deltaL[i];
            }
        }
        else
        {
            const_deltaL = length / double(n);

            for (i = 1; i < nStations; i++)
            {
                deltaL.push_back(const_deltaL);
                x.push_back(0);
            }

            // Calculate Xi for constant domain
            for (i = 1; i < nStations; i++)
            {
                x[i] = (double)i * const_deltaL;
            }
        }
        break;

    } // switch (lineNum)
} // while

// Close geoFile
geoFile.close();
return(OK);
}

int ReadType7()
{
    lineNum = 0;

    // Read the 1st line of geometry file
    while (geoFile.getline(inBuff, MAX_LINE))
    {
        lineNum++;

        switch (lineNum)
        {
        // Read data
        case 1:   // Read line No. 1
```

```
    // Read structure type
    token = strtok_s(inBuff, " ,\t\n", &nextToken);
    structType = atoi(token);

    // Read structure length
    token = strtok_s(0, " ,\t\n", &nextToken);
    length = atof(token);

    // Read number of strain-sensing stations
    token = strtok_s(0, " ,\t\n", &nextToken);
    numStations = atoi(token);

    // Read wing root depth at front H0
    token = strtok_s(0, " ,\t\n", &nextToken);
    H0 = atof(token);

    // Read wing tip depth at front Hn
    token = strtok_s(0, " ,\t\n", &nextToken);
    Hn = atof(token);

    // Initialize variables
    nStations = numStations / 2;
    n = nStations - 1;

    break;

case 2:    // Read line No. 2

    // Read domain type
    token = strtok_s(inBuff, " ,\t\n", &nextToken);
    domainType = atoi(token);

    // Push the index of deltaL and x
    deltaL.push_back(0);
    x.push_back(0);

    if (domainType == VAR_DOMAIN)
    {
        // Read deltaL
        token = strtok_s(0, " ,\t\n", &nextToken);
        while (token)
        {
            deltaL.push_back(atof(token));
            x.push_back(0);

            // read next deltaL
            token = strtok_s(0, " ,\t\n", &nextToken);
        }

        // Calculate Xi for variable domain
        for (i = 1; i < nStations; i++)
```

```
                {
                    x[i] = x[i - 1] + deltaL[i];
                }
            }
            else
            {
                const_deltaL = length / double(n);

                for (i = 1; i < nStations; i++)
                {
                    deltaL.push_back(const_deltaL);
                    x.push_back(0);
                }

                // Calculate Xi for constant domain
                for (i = 1; i < nStations; i++)
                {
                    x[i] = ((double)i) * const_deltaL;
                }
            }

            break;

        } // switch lineNum

    } // while getline

    // Close geoFile
    geoFile.close();

    // Initialize vectors h and c
    for (i = 0; i < numStations; i++)
    {
        h.push_back(0);
        c.push_back(0);
    }
    return(OK);
}

int CreateOutputFiles()
{
    unsigned int loc, loc1;

    // Create deflection output file
    outputFile = inputFile;
    loc = outputFile.find(".");
    outputFile.insert(loc, "_Deflections");

    // Open deflection output file
    outFile.open(outputFile.c_str(), ios::out);
```

```
if (outFile.fail())
{
    cerr << "Could not open " << outputFile << endl;
    return(ERROR);
}

// Create slope (angle theta) output file
if ((structureType != 2) && (structureType != 5))
{
    outthetaFile = inputFile;
    outthetaFile.insert(loc, "_Slopes");

    // Open slope output file
    thetaFile.open(outthetaFile.c_str(), ios::out);
    if (thetaFile.fail())
    {
        cerr << "Could not open " << outthetaFile << endl;
        return(ERROR);
    }
}

// Create max and min deflection output file
outmaxminFile = outputFile;
loc1 = outmaxminFile.find(".");
outmaxminFile.insert(loc1, "_MaxMin");

// Open max and min deflection output file
maxminFile.open(outmaxminFile.c_str(), ios::out);
if (maxminFile.fail())
{
    cerr << "Could not open " << outmaxminFile << endl;
    return(ERROR);
}
maxminFile << "SG Name, Time at Max Deflection, Max Deflection, Time at Min Deflection,
                Min Deflection\n";

if ((structureType == 3) || (structureType == 4))
{
    // Create twist angle (phi) output file
    outphiFile = inputFile;
    outphiFile.insert(loc, "_TwistAngles");

    // Open twist angle output file
    phiFile.open(outphiFile.c_str(), ios::out);
    if (phiFile.fail())
    {
        cerr << "Could not open " << outphiFile << endl;
        return(ERROR);
    }
    phiCreated = 1;
}
```

```
if ((structureType == 4) || (structureType == 7))
{
    // Create depth factor output file
    outcFile = inputFile;
    outcFile.insert(loc, "_DepthFactors");

    // Open depth factor output file
    cFile.open(outcFile.c_str(), ios::out);
    if (cFile.fail())
    {
        cerr << "Could not open depth factor file " << outcFile << endl;
        checked = ERROR;
        return(ERROR);
    }

    // Initialize cCreated
    cCreated = 1;
}

// Read 1st line of input file
inFile.getline(inBuff, MAX_LINE);

// Write to output files
outFile << inBuff << endl;

if ((structureType != 2) && (structureType != 5))
{
    thetaFile << inBuff << endl;
}

// Read the 1st name of the 1st line
token = strtok_s(inBuff, " ,\t\n", &nextToken);

if ((structureType == 3) || (structureType == 4))
{
    phiFile << token << ",";
}

// Read strain-sensing station names
token = strtok_s(0, " ,\t\n", &nextToken);
while (token)
{
    if ((structureType == 4) || (structureType == 7))
    {
        cFile << token << ", ";
    }
    stationNames.push_back(token);

    token = strtok_s(0, " ,\t\n", &nextToken);
```

```
    }

    // Done reading the first title line
    if (structureType == 3)
    {
        for (i = 0; i < nStations; i++)
        {
            phiFile << "Station_" << i << ",";
        }
        phiFile << endl;
    }
    else if (structureType == 4)
    {
        // Write names for lower strain-sensing stations
        for (i = 0; i < nStations; i++)
        {
            phiFile << "LwrStation_" << i << ",";
        }

        // Write names for upper strain-sensing stations
        for (i = nStations; i < noStations; i++)
        {
            phiFile << "UprStation_" << (i - nStations) << ",";
        }
        phiFile << endl;
        cFile << endl;
    }
    else if (structureType == 7)
    {
        cFile << endl;
    }
    return(OK);
}

void CalculateC()
{
    //  Initialize h[0] and h[n]
    h[0] = H0;
    h[n] = Hn;
    H_ratio = Hn / H0;

    //  Calculate front c[i] using Eqs. (1a & 2a)
    for (i = 0; i < n; i++)
    {
        // Lower front
        h[i] = H0 - (H0 - Hn)*(x[i] / length);
        c[i] = abs(epsilon[i]) * h[i] / (abs(epsilon[i]) + abs(epsilon[i + nStations]));

        // Upper front
        c[i + nStations] = h[i] - c[i];
    }
```

```
    c[n] = H_ratio*c[0];
    c[n + nStations] = H_ratio*c[nStations];

    // if structure type 4, need to do the rear
    if (structureType == 4)
    {
        // Calculate rear c[i] using Eqs. (1b & 2b)
        h[noStations] = H0_prime;
        h[noStations + n] = Hn_prime;
        Hprime_ratio = Hn_prime / H0_prime;

        for (i = noStations; i < (noStations + n); i++)
        {
            // Lower rear
            h[i] = H0_prime - (H0_prime - Hn_prime)*(x[i - noStations] / length);
            c[i] = abs(epsilon[i]) * h[i] / (abs(epsilon[i]) + abs(epsilon[i + nStations]));

            // Upper rear
            c[i + nStations] = h[i] - c[i];
        }

        c[noStations + n] = Hprime_ratio*c[noStations];
        c[noStations + n + nStations] = Hprime_ratio * c[noStations + nStations];
    }

    for (i = 0; i < numStations; i++)
    {
        cFile << c[i] << ", ";
    }

    cFile << endl;
    cFile.close();
    calC = TRUE;
}

void CalcTwistAngles()
{
    // Initialize d[0] and d[n]
    d[0] = D0;
    d[n] = Dn;

    switch (structureType)
    {
    case 3: // 2 lines
    {
        // Set twist angle at the root to 0.0
        phi[0] = 0.0;
        phiFile << fixed << setprecision(6) << phi[0];

        // Eq. (11) in this paper or Eq. (38) in NASA/TP-2009-214643
```

```
        // Calculate twist angle phi

        for (i = 1; i < nStations; i++)
        {
            d[i] = D0 - (D0 - Dn)*(x[i] / length);
            sin_phi[i] = (y[i] - y[i + nStations]) / d[i];
            phi[i] = asin(sin_phi[i])*180.0 / PI;
            phiFile << "," << fixed << setprecision(6) << phi[i];
        }
        break;
    }

    case 4: // 4 lines
    {
        // Set twist angle at lower root to 0.0
        phi[0] = 0.0;
        phiFile << fixed << setprecision(6) << phi[0];

        // Eq. (11) in this paper or Eq. (38) in NASA/TP-2009-214643
        // Calculate lower twist angle phi
        for (i = 1; i < nStations; i++)
        {
            d[i] = D0 - (D0 - Dn)*(x[i] / length);
            sin_phi[i] = (y[i] - y[i + noStations]) / d[i];
            phi[i] = asin(sin_phi[i])*180.0 / PI;
            phiFile << "," << fixed << setprecision(6) << phi[i];
        }

        // Set twist angle at upper root to 0.0
        phi[nStations] = 0.0;
        phiFile << "," << fixed << setprecision(6) << phi[nStations];

        // Calculate upper twist angle phi
        for (i = (1 + nStations); i < noStations; i++)
        {
            sin_phi[i] = (y[i] - y[i + noStations]) / d[i - nStations];
            phi[i] = asin(sin_phi[i])*180.0 / PI;
            phiFile << "," << fixed << setprecision(6) << phi[i];
        }
        break;
    } // case 4

    } // switch structureType
}

void DetermineMaxMin()
{
    for (i = 1; i < numStations; i++)
    {
        // Check for max and min deflections
        if ((structureType != 2) && (structureType != 5))
```

```
            {
                if (y[i] >= yMax[i])
                {
                    tMax[i] = t;
                    yMax[i] = y[i];
                }
                else if (y[i] < yMin[i])
                {
                    tMin[i] = t;
                    yMin[i] = y[i];
                }
            }
            else
            {
                if (yB[i] >= yMax[i])
                {
                    tMax[i] = t;
                    yMax[i] = yB[i];
                }
                else if (yB[i] < yMin[i])
                {
                    tMin[i] = t;
                    yMin[i] = yB[i];
                }
            }
        }
    }

    void  WriteMaxMinFile()
    {
        // Write yMax and yMin to file
        for (i = 0; i < numStations; i++)
        {
            maxminFile << fixed;
            maxminFile << stationNames[i] << "," << tMax[i] << "," << setprecision(6) << yMax[i]
                << "," << tMin[i] << "," << setprecision(6) << yMin[i] << endl;
        }
    }

    void PrintOutputFilenames()
    {
        // Print out successful messages
        cout << "\n$ Displacement Calculation program completed successfully!\n" << endl;
        cout << "$ Output files are listed below:\n" << endl;
        cout << "$ Deflection file: " << outputFile << endl;

        if ((structureType != 2) && (structureType != 5))
        {
            cout << "$ Slope file: " << outthetaFile << endl;
        }
```

```
    if (cCreated == 1)
    {
        cout << "$ Depth Factor file: " << outcFile << endl;
    }

    if (phiCreated == 1)
    {
        cout << "$ Twist Angle file: " << outphiFile << endl;
    }

    cout << "$ Max and Min Deflection file: " << outmaxminFile << endl << endl;
}


void CloseFiles_ClearVectors()
{
    // Close all files
    outFile.close();
    maxminFile.close();

    if ((structureType != 2) && (structureType != 5))
    {
        thetaFile.close();
    }

    if ((structureType == 3) || (structureType == 4))
    {
        phiFile.close();
    }

    // Clear out all vectors
    epsilon.clear();
    x.clear();
    y.clear();
    yB.clear();
    yMax.clear();
    yMin.clear();
    tMax.clear();
    tMin.clear();
    theta.clear();
    tan_theta.clear();
    phi.clear();
    sin_phi.clear();

    c.clear();
    d.clear();
    h.clear();
    deltaL.clear();}

void main()
```

```
{
    GetUserInputs();
    if (checked != ERROR)
    {
        ReadGeometryFile();
        if (checked != ERROR)
        {
            CreateOutputFiles();
            if (checked != ERROR)
            {
                // Let user know the program is running
                cout << "\n$ Displacement Calculation program is running  ..." << endl;

                CalcDisplacement();
                if (checked != ERROR)
                {
                    WriteMaxMinFile();
                    PrintOutputFilenames();
                }
            }
        }
    }
    CloseFiles_ClearVectors();
}
```

References

1. Ko, William L., W. L. Richards, and Van T. Tran., "Displacement Theories for In-Flight Deformed Shape Predictions of Aerospace Structures," NASA/TP-2007-214612, October 2007.

2. Ko, William L., and William Lance Richards, *Method for Real-Time Structure Shape-Sensing*, U.S. Patent No. 7,520,176, issued April 21, 2009.

3. Ko, William L., and Van Tran Fleischer, "Further Development of Ko Displacement Theory for Deformed Shape Predictions of Nonuniform Aerospace Structures," NASA/TP-2009-214643, September 2009.

4. Ko, William L., and Van Tran Fleischer, "Methods for In-Flight Wing Shape Predictions of Highly Flexible Unmanned Aerial Vehicles: Formulation of Ko Displacement Theory," NASA/TP-2010-214656, August 2010.

5. Ko, William L., and Van Tran Fleischer, "First- and Second-Order Displacement Transfer Functions for Structural Shape Calculations Using Analytically Predicted Surface Strains," NASA/TP-2012-215976, March 2012.

6. Ko, William L., and Van Tran Fleischer, "Improved Displacement Transfer Functions for Structure Deformed Shape Predictions Using Discreetly Distributed Surface Strains," NASA/TP-2012-216060, November 2012.

7. Ko, William L., and Van Tran Fleischer, "Extension of Ko Straight-Beam Displacement Theory to Deformed Shape Predictions of Slender Curved Structures," NASA/TP-2011-214657, April 2011.

8. Ko, William L., and Van Tran Fleischer, "Large-Deformation Displacement Transfer Functions for Shape Predictions of Highly Flexible Slender Aerospace Structures," NASA/TP-2013-216550, December, 2013.

9. Ko, William L., and Van Tran Fleischer, "Variable-Domain Displacement Transfer Functions for Converting Surface Strains into Deflections for Structural Deformed Shape Predictions," NASA/TP-2015-218464, March 2015.

10. Ko, William L., and Van Tran Fleischer, "Modified Displacement Transfer Functions for Deformed Shape Predictions of Slender Curved Structures with Varying Curvatures," NASA/TM-2014-216660, May 2014.

11. Ko, William L., Van Tran Fleischer, and Shun-Fat Lung, "Curved Displacement Transfer Functions for Geometric Nonlinear Large Deformation Structure Shape Predictions," NASA/TP-2017-219406, March 2017.

12. Ko, William L., Van Tran Fleischer, and Shun-Fat Lung, "Curvilinear Displacement Transfer Functions for Deformed Shape Predictions of Curved Structures Using Distributed Surface Strains," NASA/TP-2018-219692, September 2018.

13. Ko, William L., W. L. Richards, and Van Tran Fleischer, "Applications of Ko Displacement Theory to the Deformed Shape Predictions of Doubly Tapered Ikhana Wing," NASA/TP-2009-214652, November 2009.

14. Jutte, Christine, William L. Ko, Craig A. Stephens, John A. Bakalyar, W. Lance Richards, and Allen R. Parker, "Deformed Shape Calculations of a Full-Scale Wing Using Fiber Optic Strain Data from a Ground Loads Test," NASA/TP-2011-215975, November 2011.

15. Lung, Shun-Fat, and William L. Ko, "Applications of Displacement Transfer Functions to Deformed Shape Predictions of the GIII Swept-Wing Structure," Presented at the 30th Congress of the International Council of the Aeronautical Sciences, Daejeon, Korea, September 25–30, 2016.