GBT MEMO 124

The GBT Tipping-Structure Model in C

Don Wells Lee King

March 21, 1995

Abstract

The finite element model of the GBT tipping structure has been translated into executable code expressed in the C language, so that it can be used by the control software modules for the pointing, focus-tracking, quadrant detector, active-surface and laser-rangefinder subsystems of the GBT. We give a description of this C-code version of the tipping structure model and two examples of its application to practical problems.

1 From NASTRAN-output to C-code

The GBT structure was designed for CRSI[1] by Loral using the "NISA II"[2] finite-element modelling package. The final version of the tipping-structure was model "95B" (28 October 1993). The design was confirmed by CRSI using the "ANSYS"[3] modelling software, a different package from the one used by Loral. NRAO verified the model using yet another package, "MSC/NASTRAN".[4] The NASTRAN line-printer-listing file produced by NRAO's execution of model 95B was parsed by a program coded in the AWK language in order to extract the results for the nodes of the structure and reformat them into source code in the C language (see Appendix C, p.15).

NRAO's version of the model currently contains only the nodes on the right-hand-side of the GBT. Nodes which were not in the GBT's plane of symmetry were duplicated by the AWK program, and negative node-IDs were assigned. The signs of components of the translational and rotational displacements for these duplicated nodes were flipped in accordance with appropriate symmetry rules (for example, see the values for nodes ±768012 on page 16). The full symmetric model has a total of 5577 nodes, of which 40% are the active surface actuator nodes.

The 2209 active surface actuators are node-IDs 700001 through 768012. These ID numbers are composed of 700000 plus an angle part and a hoop part; *e.g.*, node -768012 is −68.0 degrees from the plane of symmetry (positive angles are clockwise as seen when looking down from the +Z axis) and is in hoop number 12 of the structure. This hoop-and-angle notation is equivalent to the notation used to specify actuators in the active surface control system and in the laser rangefinder system; *e.g.*, the retroreflector named "ZG12+680" in [Par94] is the node with ID "768012" in this model.

The undisplaced Z coordinates of this model include the 1900 inch height of the elevation axle above the top of the azimuth track. *I.e.*, to get coordinates in the elevation coordinate system (X_e, Y_e, Z_e) as defined in [Kin94], 1900.0 inches should be subtracted from `node_i->grid[2]`, the undisplaced Z value, which is returned by the function `get_node_data()`.

[1] COMSAT RSI, formerly RSi.

[2] from Engineering Mechanics Research Corporation.

[3] from Swanson Analysis Systems, Inc.

[4] "NASTRAN" is a registered trademark of the National Aeronautics and Space Administration [NASA]; MSC/NASTRAN is an enhanced, proprietary version developed and maintained by The MacNeal-Schwendler Corporation.

2 Software to retrieve the model results

Function `get_node_data()` (see Appendix A.1, p.9) retrieves values from the table of nodes (Appendix C, p.15). It takes the elevation angle as an argument and returns the undisplaced "grid" coordinates, the displacements (to be added to the "grid" values) and the rotations. The displacements and rotations are computed by forming a weighted sum of the values for the zenith and horizon gravitational deflection cases.

Function `get_index()` (see Appendix A.2, p.11) searches the table of nodes to find the table index for a specified node-ID. The indicies are cached so that repeated calls for the same node-IDs are faster.

3 Applications

This structural model is expected to be used for the following purposes:

BFP, Gravity Pointing Term Deflections of the nodes which model the active surface actuators will be fitted to derive parameters of the "best-fitting paraboloid" [BFP] as a function of elevation. The BFP will be the primary component of the "commanded surface" in the open-loop active surface module. Also, the tilt parameter of the BFP will be the *a priori* estimate of the gravitational deflection terms of the "traditional" pointing correction module.

Active Surface The open-loop active surface control for "Phase-2" of the GBT project will consist of driving the actuators with the difference between the structural model and the BFP, projected to local surface normal.

Focus Tracking The focus tracking subsystem will depend on the structural model to compute displacements of the prime focus and Gregorian optics relative to the prime focal point of the BFP (which itself depends on the structural model).

Rangefinder Control & Analysis The laser-rangefinder subsystem will use the structural model to compute the displaced locations of retroreflectors for acquisition and tracking purposes. The displacements will also be used as corrections in fitting of range measurements to derive desired parameters (such as the orientation of the distorted backup structure). Tilts of nodes will be important when rangefinders are attached to the nodes, or when retroreflectors are attached to brackets which are attached to nodes (the ends of the brackets are displaced by the tilts).

Quadrant Detector The *a priori* estimate of the "expected track" of the quadrant detector will be computed using displacements and tilts from the structural model.

In the remainder of this section, we provide two numerical examples of the application of the structural model to particular cases.

3.1 Tipping-structure Nodes Referred to Ground Coordinates

In this example, we will calculate the ground-based coordinates of the home-point of the prime focus box (node 50000) for elevation 5°, azimuth 135° (southeast).[5] `get_node_data()` will calculate the displacements using numbers extracted from the table (see node 50000 in Appendix C) in the expression,[6]

$$\begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{bmatrix} = \begin{bmatrix} 0.0000 & 0.0000 \\ -4.4914 & 9.7803 \\ -3.4570 & 0.3821 \end{bmatrix} \begin{bmatrix} \sin 5° - \sin 44° \\ \cos 5° - \cos 44° \end{bmatrix} \tag{1}$$

$$= \begin{bmatrix} 0.0000 \\ 5.4367 \\ 2.2059 \end{bmatrix}. \tag{2}$$

[5] The example in this section is adapted from a numerical example in an unpublished memo [Par95] by Dave Parker.

[6] See Eq.13 on p.10; node 50000 is one of the nodes which will be adjusted to have a rigging angle of 44°.

`get_node_data()` will also extract the undisplaced ("grid") coordinates (X_g, Y_g, Z_g) of node 50000 from the table; deflected coordinates of node 50000 in the elevation coordinate system [Kin94, KM93] can then be computed with the expression

$$\begin{bmatrix} X_e(E) \\ Y_e(E) \\ Z_e(E) \end{bmatrix} = \begin{bmatrix} X_g \\ Y_g \\ Z_g - 1900.0 \end{bmatrix} + \begin{bmatrix} \Delta_x(E) \\ \Delta_y(E) \\ \Delta_z(E) \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} 0.00 \\ -2159.02 \\ 4459.06 - 1900.0 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 5.4367 \\ 2.2059 \end{bmatrix} \tag{4}$$

$$= \begin{bmatrix} 0.0 \\ -2153.58 \\ +2561.27 \end{bmatrix}. \tag{5}$$

These coordinates can be transformed from the elevation coordinate system to the alidade coordinate system [Kin94, KM93] by multiplying by a rotation matrix[7] and adding 1900.0 to the Z axis:

$$\begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sin E & \cos E \\ 0 & -\cos E & \sin E \end{bmatrix} \begin{bmatrix} X_e \\ Y_e \\ Z_e \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1900 \end{bmatrix} \tag{6}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.0872 & 0.9962 \\ 0 & -0.9962 & 0.0872 \end{bmatrix} \begin{bmatrix} 0.0 \\ -2153.58 \\ +2561.27 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1900.0 \end{bmatrix} \tag{7}$$

$$= \begin{bmatrix} 0 \\ 2363.74 \\ 4268.74 \end{bmatrix}. \tag{8}$$

These alidade coordinates can then be transformed to "base" coordinates[8] with the expression

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \cos A & -\sin A & 0 \\ \sin A & \cos A & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix} \tag{10}$$

$$= \begin{bmatrix} -0.707 & +0.707 & 0 \\ -0.707 & -0.707 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2363.74 \\ 4268.74 \end{bmatrix} \tag{11}$$

$$= \begin{bmatrix} +1671.16 \\ -1671.16 \\ +4268.74 \end{bmatrix}, \tag{12}$$

where A is the astronomical azimuth, 135° in our example.

There are three corrections which should be included in a production version of the above calculations:

Elevation Axle Collimation Error The as-built elevation axle will not be orthogonal to the azimuth axis. The correction for this problem will appear as an additional rotation matrix in the transformation from elevation coordinates to alidade coordinates; the collimation angle for the matrix will be obtained from the "traditional" pointing error model.

[7] The rotation matrix of Eq.6 has been modified from the version in [Kin94, KM93] in order to accomodate the initial condition $E = 90°$ for the structure grid coordinates.

[8] [Kin94, KM93] defines these coordinates: A_e (Azimuth encoder value) is measured *counter-clockwise* from Y_a (*due South*); i.e., when $A_e = 0°$, X_a points West, Y_a points South. The conventional azimuth, as used in the numerical example above, is related to A_e by

$$A = 180° - A_e. \tag{9}$$

In the "base" coordinate system $+X$ is East, $+Y$ is North and $+Z$ is up, with the origin at the pintle bearing.

Table 1: Upper-feedarm differential deflections

	Greg. Feed (40700)			Subreflector (50005)			Differences					
E	δ_y	δ_z	t_x	δ_y	δ_z	t_x	ΔL	$\Delta\theta$	$\Delta t_{s\mapsto f}$	$\Delta t_{f\mapsto s}$	ΔF_y	ΔF_z
(d)	(in)	(in)	(rad)	(in)	(in)	(rad)	(in)	(rad)	(rad)	(rad)	(in)	(in)
0	4.8	2.5	-0.0021	6.2	2.9	-0.0017	0.111	-0.0024	0.0006	0.0002	-0.351	-0.190
5	4.5	2.2	-0.0020	5.7	2.6	-0.0017	0.090	-0.0022	0.0006	0.0002	-0.305	-0.158
10	4.1	1.9	-0.0018	5.2	2.2	-0.0015	0.070	-0.0020	0.0005	0.0002	-0.260	-0.128
15	3.6	1.6	-0.0016	4.6	1.9	-0.0014	0.053	-0.0018	0.0004	0.0002	-0.216	-0.101
20	3.1	1.3	-0.0014	4.0	1.6	-0.0012	0.037	-0.0015	0.0003	0.0001	-0.174	-0.076
25	2.5	1.0	-0.0011	3.2	1.2	-0.0010	0.025	-0.0013	0.0002	0.0001	-0.133	-0.054
30	1.9	0.7	-0.0009	2.4	0.9	-0.0008	0.014	-0.0010	0.0002	0.0001	-0.095	-0.035
35	1.2	0.5	-0.0006	1.6	0.6	-0.0005	0.007	-0.0006	0.0001	0.0001	-0.058	-0.020
40	0.6	0.2	-0.0003	0.7	0.2	-0.0002	0.002	-0.0003	0.0000	0.0000	-0.025	-0.007
45	-0.1	-0.0	0.0001	-0.2	-0.1	0.0001	-0.000	0.0001	-0.0000	-0.0000	0.006	0.002
50	-0.9	-0.3	0.0004	-1.1	-0.3	0.0004	0.000	0.0005	-0.0001	-0.0000	0.034	0.007
55	-1.6	-0.5	0.0008	-2.1	-0.6	0.0007	0.004	0.0008	-0.0001	-0.0001	0.058	0.009
60	-2.4	-0.7	0.0011	-3.1	-0.9	0.0011	0.010	0.0012	-0.0001	-0.0001	0.079	0.007
65	-3.1	-0.9	0.0015	-4.1	-1.1	0.0015	0.019	0.0016	-0.0002	-0.0001	0.096	0.002
70	-3.9	-1.1	0.0019	-5.1	-1.3	0.0019	0.031	0.0020	-0.0002	-0.0002	0.109	-0.007
75	-4.6	-1.2	0.0022	-6.1	-1.5	0.0023	0.045	0.0025	-0.0002	-0.0002	0.119	-0.019
80	-5.4	-1.3	0.0026	-7.1	-1.6	0.0027	0.061	0.0029	-0.0002	-0.0003	0.125	-0.035
85	-6.1	-1.4	0.0030	-8.0	-1.7	0.0031	0.080	0.0033	-0.0002	-0.0003	0.126	-0.054
90	-6.8	-1.5	0.0033	-8.9	-1.8	0.0035	0.101	0.0036	-0.0002	-0.0003	0.124	-0.076

Azimuth/Elevation Non-intersection Error The as-built elevation axis will not intersect the azimuth axis. This correction will appear as a non-zero Y-axis element in the $[0, 0, 1900.0]$ vector which is added to transform from elevation coordinates to alidade coordinates.

Encoder Zero-points The elevation E in the above expressions is not simply the raw elevation encoder reading: a zero-point correction must be applied. Likewise, a zero-point correction must be applied to the raw azimuth encoder reading A_e. These zero-point terms will be obtained from the "traditional" pointing correction model.

3.2 Differential Displacements of the Upper Feedarm

This section presents calculations of certain distortions of the upper part of the feedarm as a function of elevation. The results are interesting in their own right, but the real purpose here is tutorial: the program which computes these results demonstrates how to use the structural model functions, and the problem posed demonstrates the usefulness of the structural model for control of the optics (the focus-tracking problem). We wish to compute differential motions of the subreflector and the Gregorian feedhorn. The subreflector mirror and backup frame are represented in the model by node IDs 50004, 50005, 50010, 50040 and 50140; in particular, node 50005 represents a point in the center of the subreflector mirror. The Gregorian feed is node 40700. Table 1 presents the displacements δ_x and δ_y and tilt t_x of nodes 40700 and 50005 as a function of elevation.[9] The displacements of these nodes are plotted in Figure 1.

The results shown in Table 1 were produced by the program shown in Table 2.[10] The first 12 lines of

[9] Displacement δ_x and tilts t_y and t_z are not shown in Table 1 because they are always zero for these nodes, which are in the meridional plane, the plane of symmetry of the GBT.

[10] At several points in this program, there are expressions similar to this one: "`(double)greg_feed.grid[i]`". The purpose of the `(double)` in such expressions is to ensure that the `float` values from `struct node_data` will be cast into double precision in order to avoid truncation errors in calculations.

Table 2: Program which computes Table 1

```
/* This program, tipping_model_tab_ufdd.c, computes the 'ufdd'
   [upper_feedarm_differential_deflection] table of the
   tipping_model.tex GBT memo.  D.Wells, NRAO-CV, 1/25-3/21/95 */

#include <stdlib.h>
#include <math.h>
#include "structural_model.h"

main() {
    struct node_data greg_feed, subr_vrtx;
    double elevation, sf[3], sf44[3], lsf44, tsf44, sfsum, dlsf, dtheta,
    delta_tsf, delta_tfs, stilt, ctilt, rotate[3][3], sf44p[3], delta_f[3];
    int greg_ind, subr_ind, i, j;
    char form[200];

    greg_ind = get_index (40700);    /* Gregorian-feed        node */
    subr_ind = get_index (50005);    /* Subreflector "Vertex" node */
    elevation = 44.0;
    if (get_node_data (greg_ind, elevation, &greg_feed)) exit(EXIT_FAILURE);
    if (get_node_data (subr_ind, elevation, &subr_vrtx)) exit(EXIT_FAILURE);
    for (i = 0, sfsum = 0.0; i < 3; i++) {
        sf44[i] = ((double)greg_feed.grid[i] + greg_feed.at_elev.delta[i])
                - ((double)subr_vrtx.grid[i] + subr_vrtx.at_elev.delta[i]);
        sfsum += (sf44[i] * sf44[i]);
    }
    lsf44 = sqrt (sfsum);             /* subrefl-->feed distance @ 44d */
    tsf44 = atan2 (sf44[2], sf44[1]); /* subrefl-->feed angle    @ 44d */

    for (elevation = 0.0; elevation <= 90.1; elevation += 5.0) {
        if (get_node_data (greg_ind, elevation, &greg_feed)) exit(EXIT_FAILURE);
        if (get_node_data (subr_ind, elevation, &subr_vrtx)) exit(EXIT_FAILURE);

        for (i = 0, sfsum = 0.0; i < 3; i++) {
            sf[i] = ((double)greg_feed.grid[i] + greg_feed.at_elev.delta[i])
                  - ((double)subr_vrtx.grid[i] + subr_vrtx.at_elev.delta[i]);
            sfsum += (sf[i] * sf[i]);
        }
        dlsf   = (sqrt (sfsum) - lsf44); /* change in subrefl-->feed distance */
        dtheta = (atan2 (sf[2], sf[1]) - tsf44); /* change in angle of line */
        delta_tsf = subr_vrtx.at_elev.tilt[0] - dtheta; /* extra subr */
        delta_tfs = greg_feed.at_elev.tilt[0] - dtheta; /* extra feed */

        ctilt = cos(subr_vrtx.at_elev.tilt[0]); /* rotation about X-axis */
        stilt = sin(subr_vrtx.at_elev.tilt[0]);
        rotate[0][0] = 1.0; rotate[0][1] = 0.0;    rotate[0][2] = 0.0;
        rotate[1][0] = 0.0; rotate[1][1] = ctilt;  rotate[1][2] = -stilt;
        rotate[2][0] = 0.0; rotate[2][1] = stilt;  rotate[2][2] = ctilt;

        for (i = 0; i < 3; i++) {
            for (j = 0, sf44p[i] = 0.0; j < 3; j++)
                sf44p[i] += sf44[j] * rotate[i][j];
            delta_f[i] =
                ((double)greg_feed.grid[i] + greg_feed.at_elev.delta[i])
              - ((double)subr_vrtx.grid[i] + subr_vrtx.at_elev.delta[i]
                                                          + sf44p[i]);
        }

        strcpy (form, "%2.0lf&%4.1f&%4.1f&%7.4f&%4.1f&%4.1f&%7.4f");
        strcat (form, "&%5.3lf&%7.4lf&%7.4lf&%7.4lf&%5.3lf&%5.3lf\\\\\\n");
        printf (form, elevation, greg_feed.at_elev.delta[1],
                greg_feed.at_elev.delta[2], greg_feed.at_elev.tilt[0],
                subr_vrtx.at_elev.delta[1], subr_vrtx.at_elev.delta[2],
                subr_vrtx.at_elev.tilt[0], dlsf, dtheta, delta_tsf, delta_tfs,
                delta_f[1], delta_f[2]);
    }
    exit(EXIT_SUCCESS);
}
```

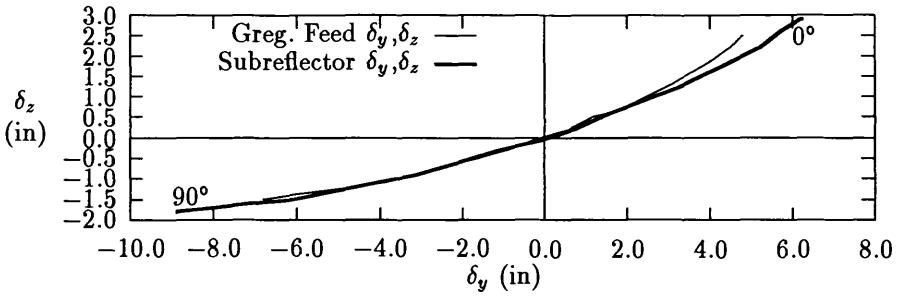

Figure 1: Two trajectories as a function of Elevation

executable code in this program demonstrate how to use functions **get_index()** and **get_node_data()** to obtain information about the geometry of the GBT in order to perform simple geometric calculations on the design (rigging-angle) geometry, such as the length (**lsf44**) and position angle (**tsf44**) of the line connecting two nodes. The **for**-loop on variable **elevation** illustrates how to utilize node displacements and tilts as a function of elevation. Three sets of results are shown in the table:

Change of Node-to-Node Line-of-Sight The line connecting nodes 40700 and 50005 changes length and orientation as a function of elevation. ΔL is the difference between the length at the specified elevation and the length (**lsf44**) at the rigging angle; it is plotted in Figure 2. $\Delta\theta$ is the difference between the position angle of the line-of-sight at the specified elevation and the position angle (**tsf44**) at the rigging angle; the principal effect represented in $\Delta\theta$ is the gross bending of the entire feedarm. $\Delta\theta$ is plotted in Figure 3.

"Excess Rotation" of the Nodes $\Delta t_{s\mapsto f}$ is the *excess rotation* of the subreflector relative to the rotation of the line of sight from the subreflector to the Gregorian feed; see Figure 3. It is simply $(t_x - \Delta\theta)$ (*i.e.*, 0.0006 at $E = 0°$ is $(-0.0017) - (-0.0024)$, see the expression for **delta_tsf** in Table 2). $\Delta f_{f\mapsto s}$ is the same difference for the Gregorian feedhorn.

Displacement of Second Focal Point Relative to Feedhorn If a rigid rod is attached to the subreflector node, representing the second focal point of the ellipsoidal mirror, the tip of this rod moves as a function of elevation. The subreflector node is displaced as a function of elevation, which displaces the base of the rod, and it also tilts, which moves the tip of the rod relative to its base. The algorithm shown here uses the node tilt to construct a rotation matrix which is multiplied by the vector **sf44[]**, representing the undisplaced orientation of the rod. The tilted rod is vector **sf44p[]**, which is added to the displaced position of node 50005 in order to get the position of the tip of the rod. Vector **delta_f[]** is the position difference between that tip of the rod and the displaced position of node 40700, the Gregorian feed; it appears in the table as ΔF_y and ΔF_z,[11] and is plotted in Figure 4. Correction of this position difference will be part of the focus tracking algorithm of the GBT.[12]

We offer the following comments and conjectures about the results in Table 1:

- Eleven of the twelve tabulated quantities change sign between the lines for 40° and 45°. This is because of the 44° rigging angle.

[11] ΔF_x is always zero in the plane of symmetry.

[12] The difference between the first focal point of the ellipsoid and the prime focal point of the best-fitting paraboloid [BFP] as a function of elevation will the dominant term of the focus tracking algorithm (magnitude about 20× larger than the ΔF_y,ΔF_z values in Table 1 and Figure 4): the displacement of the first focus is not only larger but also has a disproportionate effect on the imaging and gain compared to the differential distortion of the feedarm because the first focus displacement is magnified at the second focus by the ellipsoid.

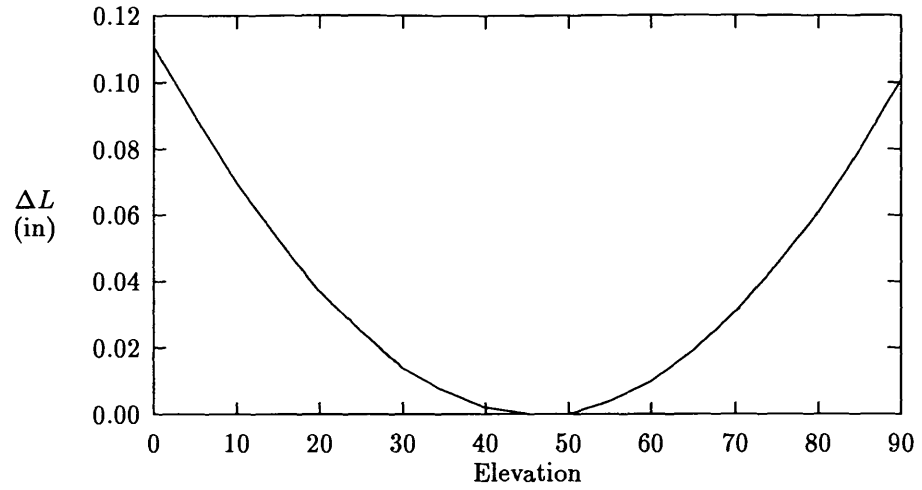

Figure 2: Change of distance between nodes 40700 and 50005

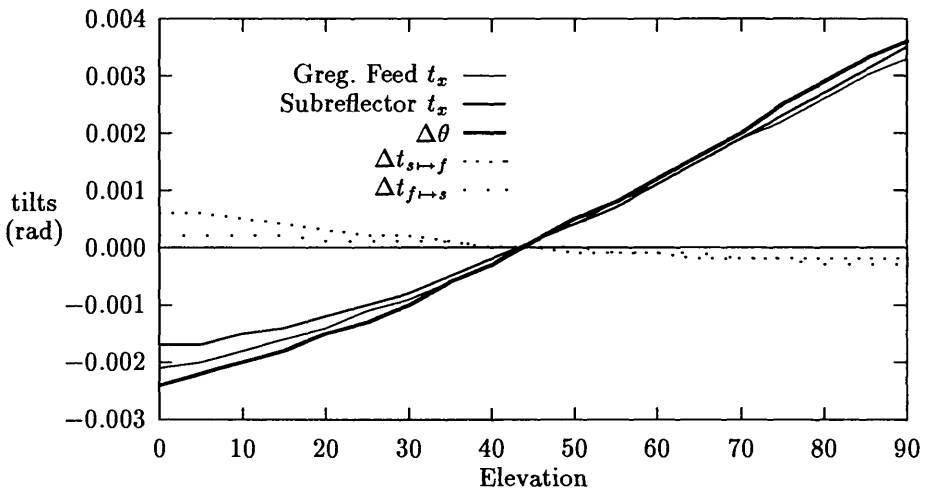

Figure 3: Five tilts as a function of Elevation

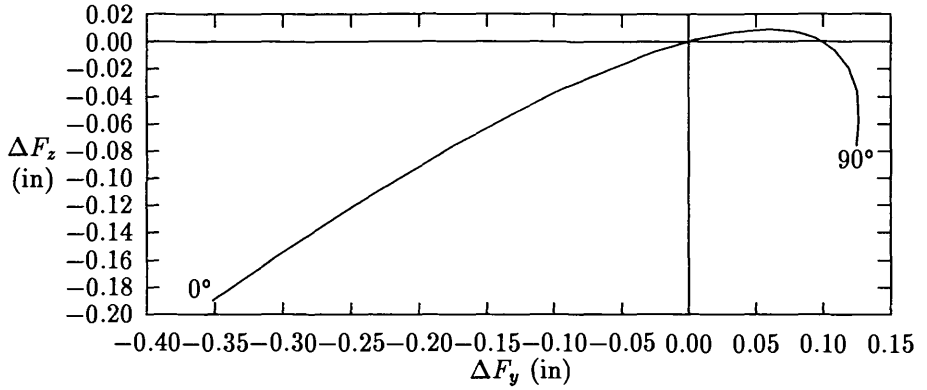

Figure 4: ΔF_y,ΔF_z trajectory as a function of Elevation

- The displacements and tilts of the Gregorian feed and subreflector are similar (see Figures 1 and 3). This shows the gross deflection of the upper part of the feedarm as a function of elevation, and the associated rotation (bending). The trajectory of the subreflector is longer (see Figure 1) because it is further out on the feedarm. The δ_y components are large at small elevations; the values are positive because when the feedarm is nearly horizontal, the +Y direction of the tipping coordinate system is nearly parallel to the gravity vector.

- $\Delta t_{s \mapsto f}$ is consistent with ΔF_y (as we would expect): the distance between nodes 40700 and 50005 is about 600 inches, so the 0.6 milliradian excess rotation of the subreflector at $E = 0°$ implies a lateral motion of the second focal point of the ellipsoid relative to the Gregorian feedhorn of about 0.36 inches (compared to $\Delta F_y = -0.35$). The sign of ΔF_y is negative because the subreflector sags downward, away from the feedarm, in the +Y direction, and ΔF_y is computed as the Gregorian feed minus the focal point (see the formula for `delta_f[]` in Table 2).

- $\Delta t_{f \mapsto s}$ is the excess rotation of the Gregorian feed horn relative to the line of sight from the feedhorn to the center of the subreflector. The rotation is positive for low elevations because the feedroom sags downward (-Z direction) at high elevations, which is a negative rotation about the X axis (right-hand rule). The distance between nodes 40700 and 50005 is about 600 inches, and so the 0.2 milliradian misalignment for $E = 0°$ implies a lateral shift of the power pattern of the horn on the subreflector by about 0.12 inch (about 3 mm). This will produce a very slight change of the spillover.

4 Three limitations of the tipping-structure model

Users of these functions should be aware that the as-built undisplaced XYZ coordinates of nodes will be different from the values returned by function `get_node_data()` in the array `*node_i.grid[]`, for several reasons:

Construction Tolerances CRSI will assemble the structure with accuracy no better than ±3 mm.

Temperature Variations Temperature changes during assembly and during operation will cause substantial changes from the as-designed coordinates: the expansion coefficient of steel is about 10^{-5} per °C, which, for a 100 meter structure, implies displacements of about 1 mm per °C.

Elevator Assymetry NRAO's current NASTRAN model is symmetric – the left half is a mirror image of the right half. This is slightly incorrect, because of the elevator on the right half. The mass of the elevator has been included in the model, and so the mean deflection of the structure has been modelled. However, the assymetry of the mass distribution will cause a small rotational deflection of the feed arm. We should replace the present model with the assymetric version eventually, so that the focus-tracking subsystem will properly compensate for this deflection by rotating the optical axis of the Gregorian ellipsoid about its Z-axis.

5 Availability

The complete package of code and data for the structural model is available at the URL

`ftp://fits.cv.nrao.edu/pub/gbt_tipping.tar.gz`

This compressed "tar" file is 355 kilobytes in length. Use "**gunzip**" to decompress it, then "**tar**" to unpack it and then do command "**make**" in the directory. In addition to the three pieces of code listed in the appendices of this document, the **tar**-file contains a Makefile and two testcases for compiling and testing the code. It also contains a copy of this GBT memo in Postscript, plus a 330 kilobyte Postscipt file which prints a convenient listing of the node coordinates of the tipping structure.

References

[Kin94] Lee King. GBT coordinate systems. Limited-distribution memo, January 1994.

[KM93] Lee King and Greg Morris. Foci arrangement and coordinate systems for the GBT. GBT Drawing C35102M081, NRAO, December 1993. The first sheet of this set of five drawings schematically defines six different coordinate systems to be used in the GBT project. Sheets 2-5 define the algebraic relationships between these coordinate systems.

[Par94] David H. Parker. GBT Actuator/retroreflector/panel Spreadsheet. GBT Memo 114, National Radio Astronomy Observatory, September 1994. This memo describes a Quattro Pro spreadsheet which has been built from the actuator and panel information on drawings 12520/2 (11/13/92) and 121010/7/A (11/12/93).

[Par95] David Parker. FEA model. Limited-distribution memo. This memo discusses details of the coordinate conventions used in the NASTRAN model of the tipping structure, and includes a numerical example of representing a node of the structure in the ground-referenced coordinate system., January 1995.

A The C-functions

These functions are in two files, **get_node_data.c** and **get_index.c**.

A.1 Function to retrieve data for nodes

The text reproduced below is the file **get_node_data.c**, which contains the ANSI-C code for fetching values from the structural model. Note that this module depends on include-file **structural_model.h** (see Appendix B) and that it also references the table of nodes (Appendix C) in a manner which causes that structure definition to be linked as a static entity known to this function.

The gravitational deformations as a function of elevation are computed in the `for`-loop near the end of function `get_node_data()`. The displacements `node_i->at_elev.delta[]` are computed as a weighted sum of values from the model; this sum can be represented as the vector-matrix product[13]

$$\begin{bmatrix} \Delta_x(E) \\ \Delta_y(E) \\ \Delta_z(E) \end{bmatrix} = \begin{bmatrix} \sigma_{x,-z} & \sigma_{x,y} \\ \sigma_{y,-z} & \sigma_{y,y} \\ \sigma_{z,-z} & \sigma_{z,y} \end{bmatrix} \begin{bmatrix} \sin E - \sin E_r \\ \cos E - \cos E_r \end{bmatrix} \tag{13}$$

where $\Delta_x(E)$, $\Delta_y(E)$ and $\Delta_z(E)$ are the deformations from the rigid body nominal coordinates at the rigging angle, $\sigma_{i,j}$ are the deformations computed by NASTRAN for the i direction for a gravity force in the j direction, E is the elevation and E_r is the rigging angle.

The logic in this function supports two different rigging angles, called **birdbath** (66°) and **surf_rig** (44°). The GBT structure will be assembled in the "birdbath" orientation, which therefore is the default rigging angle, but the surface and optics nodes will be adjusted to the correct coordinates (within a tolerance, of course) for rigging angle 44°. The IDs of the nodes which will be adjusted are specified in the `if`-statement below. There is a possibility that the feedarm will be assembled at some other angle, not 66°; if that happens we will introduce the third rigging angle into the logic of this function.

```
/* Function get_node_data() forms a weighted sum of the zenith and
   horizon cases of the GBT structural model, in order to compute the
   translation and rotation of a specified node for a specified
   elevation. The computation accounts for the "rigging angle",
   including the fact that different parts of the GBT will be built
   with different rigging angles.  The node is specified as the index
   in the model table. Such indicies can be obtained for a specified
   node_id by calling function get_index().
   Don Wells <dwells@nrao.edu>, NRAO-CV, Mar94-Mar95.  */

#include "structural_model.h"
#include <math.h>

int get_node_data (int i,                    /* tipping_model[] index to get */
                   double elev,              /* elevation             (deg) */
                   struct node_data *node_i) /* results returned in a struct */
{
    extern struct model_node GBT_TIPPING;
    struct model_node *tipping_model = &GBT_TIPPING;
    const double
        dtr      = 0.017453293,
        birdbath = (66.0 * dtr),   /* backup structure rigging_angle */
        surf_rig = (44.0 * dtr),   /* surface actuator rigging_angle */
        sin_birdbath = sin(birdbath),
        cos_birdbath = cos(birdbath),
        sin_surf_rig = sin(surf_rig),
        cos_surf_rig = cos(surf_rig);
    double sin_elev, cos_elev, sin_rig, cos_rig;
    int l, status, abs_node_id;

    sin_elev = sin(elev * dtr);
    cos_elev = cos(elev * dtr);

    if ((i >= 0) && (i <= tipping_model[0].node_id)) { /* legal index? */
        node_i->node_id = tipping_model[i].node_id;    /* yes */

        /* get the sine and cosine of the appropriate rigging_angle: */
```

[13] This notation is adapted from Dave Parker's memo [Par95].

```
        abs_node_id = (node_i->node_id < 0)
            ? -(node_i->node_id) : node_i->node_id;
        if ( ((abs_node_id >= 700001) && (abs_node_id <= 768012)) /* surface */
            || (abs_node_id == 40700)            /* plus Gregorian Feed node */
            || (abs_node_id == 50000)             /* and Prime Focus feedbox */
            || (abs_node_id == 50005) ) {  /* and the Subreflector "vertex". */
            sin_rig = sin_surf_rig;  /* rigging for surface (actuator) nodes */
            cos_rig = cos_surf_rig;  /*     plus optics nodes */
        } else {
            sin_rig = sin_birdbath;  /* rigging for other nodes */
            cos_rig = cos_birdbath;
        }

        /* form the weighted sum of the zenith and horizon models: */
        for (l = 0; l < 3; l++) {
            node_i->grid[l] = tipping_model[i].grid[l];
            node_i->at_elev.delta[l] =
                (sin_elev - sin_rig) * tipping_model[i].zenith.delta[l]
              + (cos_elev - cos_rig) * tipping_model[i].horizon.delta[l];
            node_i->at_elev.tilt[l] =
                (sin_elev - sin_rig) * tipping_model[i].zenith.tilt[l]
              + (cos_elev - cos_rig) * tipping_model[i].horizon.tilt[l];
            status = 0;
        }
    } else {
        status = 13; /* return bad status on illegal requested index */
    }
    return(status);
}
```

A.2 Function to search for indicies of node-IDs

```
/* Function get_index() searches the GBT structural model table to
   find the index (subscript) for the data in the array of struct
   tipping_model[] for a specified node_id. It caches the index in a
   hash table to speed up subsequent calls for the same node_id.  In
   many practical cases, with only a few relevant nodes, get_index()
   can be executed once for each node of interest, and the indicies
   saved and used repeatedly to call get_node_data() for various
   elevations. D.Wells <dwells@nrao.edu>, NRAO-CV, Mar94-Jan95.  */

#include "structural_model.h"
#include <math.h>

/* Define the hash table as a static array. Dimension HASH_M should be
   a prime number which is at least 10 percent greater than the number
   of nodes in the model, tipping_model[0].node_id (which is currently
   5577). */
#define HASH_M 7993
static struct hash_node {
    int node_id;
    int model_index;
};
static struct hash_node hash_table[HASH_M+1];
static int hash_init = 0, hash_count, chain_count, max_chain;
```

```
int get_index (int node_id)
{
    int i, k, k_step, index, hash_chain;
    unsigned int u_node_id;
    extern struct model_node GBT_TIPPING;
    struct model_node *tipping_model = &GBT_TIPPING;

    if (hash_init == 0) {
        for (i = 0; i <= HASH_M; i++)
            hash_table[i].node_id = hash_table[i].model_index = 0;
        hash_count = chain_count = max_chain = 0;
        hash_init = 1;
    }
    u_node_id = (node_id < 0) ? (1000000 - node_id) : node_id;
    k = (int) hash1 (u_node_id, HASH_M);
    k_step = index = 0;
    for (hash_chain = 0; ; hash_chain++) {
        /* is this node_id in the hash table? */
        if (hash_table[k].node_id == node_id) {
            /* found it! return value: */
            index = hash_table[k].model_index;
            break;
        }
        /* has search reached an empty node? */
        if (hash_table[k].node_id == 0) {
            /* table node is empty, does this node_id even exist? */
            for (i = 1; i <= tipping_model[0].node_id; i++) {
                if (tipping_model[i].node_id == node_id) {
                    /* node_id does exist; install it in this table node: */
                    index = i;
                    hash_table[k].node_id = node_id;
                    hash_table[k].model_index = index;
                    chain_count += hash_chain;
                    if (hash_chain > max_chain) max_chain = hash_chain;
                    /* check whether hash_table is too full: */
                    hash_count++;
                    if (hash_count > (int) (0.9 * HASH_M)) {
                        printf("hash_count=%d > ninety percent of %d. ABORT.\n",
                               hash_count, HASH_M);
                        exit (HASH_M);
                    }
                    break;
                }
            }
            /* if node_id doesn't exist, will return zero */
            break;
        } else {
            /* step around hash table: */
            if (k_step == 0) k_step = (int) hash2 (u_node_id, HASH_M);
            k = (k + k_step) % HASH_M;
        }
    }
    return(index);
}

/* The first hash function here is used for initial probes into the
   hash table. The original version of this function was based on
   function hash() on p.233 of Sedgewick, "Algorithms in C"
```

```
   (Addison-Wesley 1990, ISBN=0-201-51425-7, LOC=QA76.73.C15S43). The
   current version is a DonWells idea (multiply each byte by a
   different prime number).  */

unsigned int hash1 (unsigned int k, /* input key value to be hashed */
                    unsigned int m) /* hash table size              */
{
    /* int i; */
    unsigned int h;
    union {
        unsigned int kp;
        char kc[4];
    } u;
    /* const int multiplier = 64; */

    u.kp = k; /* kc[] now contains 4 bytes of input k */

    /* h = 0;   --- the algorithm from Sedgewick  */
    /* for (i = 0; i < 4; i++)                    */
    /* h = (multiplier * k + u.kc[i]) % m;        */

    h = (u.kc[0] * 71
         + u.kc[1] * 113
         + u.kc[2] * 173
         + u.kc[3] * 229) % m;

    return (h);
}

/* The second hash function is used to calculate the step size for
   linear probing when we have collisions in the hash table, i.e. it
   is used for "double hashing". Originally this was the first h2()
   function on p.240 of Sedgewick, but now it is like hash1(), but
   with different primes in the formula.  */

unsigned int hash2 (unsigned int k, /* input key to be hashed */
                    unsigned int m) /* hash table size        */
{
    unsigned int h;
    union {
        unsigned int kp;
        char kc[4];
    } u;

    u.kp = k; /* kc[] now contains 4 bytes of input k */

    /* h = (m - 2) - k % (m - 2); --- the original Sedgewick code */

    h = (u.kc[0] * 73
         + u.kc[1] * 127
         + u.kc[2] * 179
         + u.kc[3] * 233) % m;

    return (h);
}

/* Function get_hash_chain() is used to retrieve internal statistics
   maintained in static memory by the hash-table-building algorithm of
```

```
   get_index(). */

void get_hash_chain (int *hash_size,    /* size of hash table           */
                     int *hash_total,   /* number of entries            */
                     int *chain_total,  /* number of linear search steps */
                     int *chain_max,    /* longest linear search        */
                     double *chain_avg) /* average linear search        */
{
    *hash_size = HASH_M;
    *hash_total = hash_count;
    *chain_total = chain_count;
    *chain_max = max_chain;
    *chain_avg = ((double) chain_count / (double) hash_count);
}
```

B The "include" file

The text reproduced below is the file **structural_model.h**, which contains definitions for the data structures, plus the ANSI-C prototypes for the functions.

```
/* The include for the GBT structural model functions and data table.
   Note use of 'float' rather than 'double' in these declarations in
   order to save space.
   Don Wells, NRAO-CV, 3/18/94,5/19,12/16,1/26/95.  */

struct displacement {
    float delta[3];             /* displacements in XYZ        (inches) */
    float tilt[3];              /* rotations about  XYZ       (radians) */
};

struct node_data {                /* struct returned by get_node_data()   */
    int node_id;                  /* node_id returned by get_node_data()  */
    double elevation;             /* elevation for node_data    (degrees) */
    float grid[3];                /* undisplaced XYZ of node_id  (inches) */
    struct displacement at_elev;  /* displacement & tilt at elevation     */
};

struct model_node {             /* the model is an array of this struct */
    int node_id;                /* node id code                         */
    float grid[3];              /* XYZ of undisplaced node     (inches) */
    struct displacement  zenith;
    struct displacement horizon;
};

#define GBT_TIPPING static_gbt_tipping

/* -=-=-=-=-=-=-=-=-=-=- function prototypes: -=-=-=-=-=-=-=-=-=-=-=-  */

int get_node_data (int i,                   /* tipping_model[] index to get */
                   double elev,             /* elevation              (deg) */
                   struct node_data *node_i) /* results returned in a struct */
;
int get_index (int node_id)
;
unsigned int hash1 (unsigned int k, /* input key value to be hashed */
                    unsigned int m) /* hash table size              */
```

```
;
unsigned int hash2 (unsigned int k, /* input key to be hashed */
                    unsigned int m) /* hash table size       */
;
void get_hash_chain (int *hash_size,    /* size of hash table           */
                     int *hash_total,   /* number of entries            */
                     int *chain_total,  /* number of linear search steps */
                     int *chain_max,    /* longest linear search        */
                     double *chain_avg) /* average linear search        */
;
```

C The table of nodes

In this section, we reproduce the data for selected nodes from the file `gbt_tipping.c`, whose 1.1 megabytes of text contains the structural model information for the 5744 nodes of the tipping structure. Readers will notice that the information for the left-half nodes (with negative node-IDs) has been computed from the right-half nodes, and they may wonder why memory space is being used for redundant numbers. The reason is that we will eventually substitute an assymetric model for this symmetric one (see Sect. 4), and we prefer not to have to revise the software at that time. Several of the nodes in the table have special significance:

Node-ID	Significance
40700	Gregorian Feed
50000	Prime Focus Box
50005	Subreflector Vertex[14]
700001	first actuator
768012	last actuator

Two of these five nodes have a special property: the Euclidean distance between nodes 40700 and 50000 is 433.08 inches, which is *exactly* 11.000 meters (the design focal length of the ellipsoidal subreflector).

The compiled code for this table is about 368 kilobytes; this is also the amount of memory which is needed for the table at execution time. (Each node is represented as 16 4-byte objects, therefore 5744 nodes will need 367616 bytes.) The reader will see that each node is represented in this table by three lines of text. The sixteen numbers appearing in the three lines are grouped by nested brackets. These groupings correspond to the elements of C-structs `model_node` (which uses struct `displacement`), as defined in file `structural_model.h` (Appendix B). Distances are in inches, angles in radians. The rotation convention for the angles is "right-hand-rule": if the thumb of the right hand points in the plus-X direction, the fingers curl in the plus-rotation sense of the first rotation number (rotation about the X-axis).

```
#include "structural_model.h"

char GBT_TIPPING_TEXT[] =

"MOD95.DAT GRAVITY LOADS (ZEN/HOR) OCTOBER 28, 1993 MSC/NASTRAN 4/15/93";

/* ------------------------------------------------------------------ */
/* 0010                                DATA2/LKING/MODEL95 DIRECTORY  */
/* 0012                             S O R T E D B U L K D A T A E C H O  */
/* 3444        ZENITH GRAVITY (-1 Z GRAVITY: 386.00 IN/SEC^2) SUBCASE 1  */
/* 6931       HORIZON GRAVITY (+1 Y GRAVITY: 386.00 IN/SEC^2) SUBCASE 3  */
/* ------------------------------------------------------------------ */
```

[14] This node, which is referred to as the "vertex" of the subreflector, is associated with the center of the off-axis subreflector, not with the true vertex of the full ellipsoid.

```
struct model_node GBT_TIPPING[] = {

{   5744, /* number of nodes */
 {0.0,0.0,0.0},{{0.0,0.0,0.0},{0.0,0.0,0.0}},{{0.0,0.0,0.0},{0.0,0.0,0.0}}},

{   1000, {    0.00,     0.00,  1900.00},
         {{  0.0000,   0.0387,  -0.7573},{  0.00021,   0.00000,   0.00000}},
         {{  0.0000,   0.6583,  -0.0561},{ -0.00060,   0.00000,   0.00000}}},
{   1200, {  463.89,     0.00,  1900.00},
         {{ -0.1498,  -0.0793,  -0.5635},{  0.00035,  -0.00033,  -0.00013}},
         {{ -0.0417,   0.6162,   0.0261},{ -0.00081,  -0.00020,   0.00003}}},
{  -1200, { -463.89,     0.00,  1900.00},
         {{  0.1498,  -0.0793,  -0.5635},{  0.00035,   0.00033,   0.00013}},
         {{  0.0417,   0.6162,   0.0261},{ -0.00081,   0.00020,  -0.00003}}},
    .
    .
    .
{  40700, {    0.00, -2201.06,  4028.03},
         {{  0.0000,  -3.7888,  -3.3679},{  0.00148,   0.00000,   0.00000}},
         {{  0.0000,   7.8618,   0.6270},{ -0.00399,   0.00000,   0.00000}}},
    .
    .
    .
{  50000, {    0.00, -2159.02,  4459.06},
         {{  0.0000,  -4.4914,  -3.4570},{  0.00037,   0.00000,   0.00000}},
         {{  0.0000,   9.7803,   0.3821},{ -0.00332,   0.00000,   0.00000}}},
    .
    .
    .
{  50005, {    0.00, -2327.96,  4608.71},
         {{  0.0000,  -4.6923,  -3.8280},{  0.00065,   0.00000,   0.00000}},
         {{  0.0000,  10.4457,   0.9416},{ -0.00454,   0.00000,   0.00000}}},
    .
    .
    .
{ 700001, {    0.00, -2002.17,  2099.45},
         {{  0.0000,  -0.0058,  -1.3308},{  0.00014,   0.00000,   0.00000}},
         {{  0.0000,   0.8346,   0.9210},{ -0.00085,   0.00000,   0.00000}}},
    .
    .
    .
         {{  0.0396,   0.8825,   0.8192},{ -0.00109,   0.00009,  -0.00008}}},
{ 768011, { 1073.13, -1743.40,  2231.31},
         {{  0.0615,  -0.0327,  -1.2592},{  0.00088,   0.00018,  -0.00028}},
         {{ -0.0207,   0.8458,   0.8163},{ -0.00115,  -0.00035,   0.00061}}},
{-768011, {-1073.13, -1743.40,  2231.31},
         {{ -0.0615,  -0.0327,  -1.2592},{  0.00088,  -0.00018,   0.00028}},
         {{  0.0207,   0.8458,   0.8163},{ -0.00115,   0.00035,  -0.00061}}},
{ 768012, { 1162.50, -1707.29,  2256.94},
         {{  0.0549,  -0.0116,  -1.2660},{  0.00094,   0.00006,   0.00006}},
         {{ -0.0313,   0.8508,   0.8095},{ -0.00087,  -0.00010,  -0.00006}}},
{-768012, {-1162.50, -1707.29,  2256.94},
         {{ -0.0549,  -0.0116,  -1.2660},{  0.00094,  -0.00006,  -0.00006}},
         {{  0.0313,   0.8508,   0.8095},{ -0.00087,   0.00010,   0.00006}}}};
```