GBT Memo 131

# GBT Best-Fitting-Paraboloid [BFP] in C

Don Wells Lee King

June 20, 1995

**Abstract**

The gravitational displacements of the GBT actuators have been fitted with a paraboloid. The parameters of the paraboloid for various elevations have been fitted with polynomials and expressed as C code which computes the parameters of this best-fitting-paraboloid [BFP] as a function of elevation. The BFP will be used by the control software modules for the pointing, focus-tracking and active-surface subsystems of the GBT. We give a description of this C-code version of the BFP and two examples of its application to practical problems. We also give a function in C which fetches node data from the structural model and transforms it to a coordinate system tied to the BFP. The predicted gravitational term of the GBT's traditional pointing model and the predicted prime focus focus-tracking formula of the GBT are given.

## 1 The BFP concept and strategy

In order to implement a consistent control strategy for the active surface of the GBT primary and the Prime/Gregorian optics, we must choose a target paraboloid for the primary as a function of elevation. There are many candidate paraboloids. One obvious choice is the paraboloid which most closely approximates the surface as the structure deforms (almost) homologously as a function of elevation. This paraboloid has been called the "BFP" [Best-Fitting Paraboloid] since the early days of the GBT project. The BFP choice would have the advantage of minimizing the required surface actuator motions as a function of elevation.

In 1989, early in the GBT project, D'Addario [D'A89] argued for a different target paraboloid strategy, one in which the actuators would be used to maintain a fixed paraboloid, presumably with 60 meter focal length, as a function of elevation. Thompson [Tho89] responded with arguments favoring the BFP strategy. After this scholarly debate, a consensus emerged that the GBT active surface actuators and prime Gregorian foci actuators would be designed on the assumption that the GBT will use the BFP strategy.

The debate was resumed three years later by von Hoerner [vH93], who argued that the active surface could be used to move the prime focal point as a function of elevation in order to get an extra degree of freedom for control of the Gregorian optics. von Hoerner advocated using this extra DOF to maintain constant spillover while maximizing the gain as the gravitational deflections move the Gregorian optics away from their nominal geometry. It appears that the needed motions would be just within the range of travel of the surface actuators. A different concern is that the design (rigging angle) geometry of the GBT has been chosen to minimize cross-polarization [MAY76], and a residual cross-polarization will appear as the optics move; in principle the active surface might be used to avoid or minimize this problem rather than being used to control spillover. It is unclear whether both effects could be controlled simultaneously.

At the present time, our intent is to use the BFP strategy, at least for initial operation of the GBT. It is possible, however, that a future review will show that some variation on von Hoerner's concept is worth adopting.

# 2 Four datasets and four fitting programs

Several people have worked on the BFP problem, each producing a different fitting program. In addition, the surface displacement datasets can be produced in several different ways. The programs produced previously read different input data formats and produced their parameter-estimates in different output formats; these differences hindered intercomparison tests. During the course of this project, the previous fitting programs were modified to read surface displacement data in a common input format, and to produce their parameter results in a common output format. In addition, a test dataset generator program and two new fitting programs were developed.

Four different datasets were produced, containing a total of 43 cases to be fitted. There are three datasets each containing cases for 12 different elevations plus a dataset of 7 test cases:

**rmsnsdl** This program was developed by King. It reads two files, `surfgrid.coor` (the "grid" nominal coordinates of the nodes) and `dispzh.m95b` (zenith and horizon node displacements of the nodes); these two files were produced by King from the MSC/NASTRAN run of model 95b, and they are believed to be equivalent to the structural model in C [WK95] used by the program `get_surface` described below. `rmsnsdl` produces a dataset containing only the 1127 nodes on the right-hand half of the GBT primary mirror. Cases are produced for 12 elevations 90°, 80°, 70°, 60°, 50°, 44° (the surface and optics rigging angle), 40°, 35°, 30°, 20°, 10° and 0°. King used his version of `rmsnsdl` to produce the previously distributed [Kin94a] estimates of BFP parameters.

**tipping1/tipping0** These two datasets are produced by program `get_surface` (see Appendix F, p. 27), which calls function `get_node_data()` [WK95, App. A.1, p.9]. The `tipping0` ("half") execution of `get_surface` produces data for the 1127 nodes, analogous to the output of program `rmsnsdl`, while the `tipping1` ("full") execution produces datasets with 2209 nodes. Cases are produced for the same set of elevations listed above for program `rmsnsdl`.

**testgen** This program uses Fortran functions `randgs()` and `rand()`[1] to produce normal deviates of specified standard deviation which are added to the $x$, $y$ and $z$ coordinates of points on a paraboloid with specified parameters. The chosen test case is approximately the BFP paraboloid for $E = 60°$. The vertex is displaced by (-0.71,-0.44) inches from the vertex of the GBT's nominal paraboloid, the axis is tilted by 0.1 milliradians and the focal length is 0.1 inch greater than the nominal 60 meters (2362.2 inches). Seven cases are produced, one with zero noise and three each of two different RMS noise levels, 0.02 inch and 0.04 inch, so that the accuracy of the fitting process can be explored.

There are four different fitting programs, each of which processes all 43 cases:

**rms** This program was originally developed at the Jet Propulsion Laboratory circa 1962 [KS66]. The program was modified by King over the past two decades, as it was being used to check the design of backup structures for successive NRAO antennas. King's original implementation for the GBT was built to fit only the "half" case, with 1127 nodes; Wells modified it to support 2209 nodes. `rms` solves for the paraboloid which minimizes the RMS of the *path-length* residuals, not the RMS of $z$-coordinate residuals or the RMS of orthogonal-distance residuals. Program `rms` has the advantages of being well-proven and of executing rapidly.

**fitrefl1** This program was built for the GBT project by Fred Schwab in 1992 [Sch92]. It uses ODRPACK [BBRS92] for fitting the paraboloids. Wells modified Schwab's program to use the most recent version of ODRPACK. This program minimizes the residuals normal to the paraboloid (ODR is "Orthogonal Distance Regression").

**impodr** This program was built by Wells, and it too solves the ODR problem by using ODRPACK. It uses an "implicit ODR" formulation of the problem. It appears that, at least for fitting paraboloids, this algorithm has no technical advantages over the "explicit ODR" algorithm used by `fitrefl1`, and it has the disadvantage of executing more slowly. The user-function Fortran subroutine supplied to

[1] These two functions were provided by Fred Schwab.

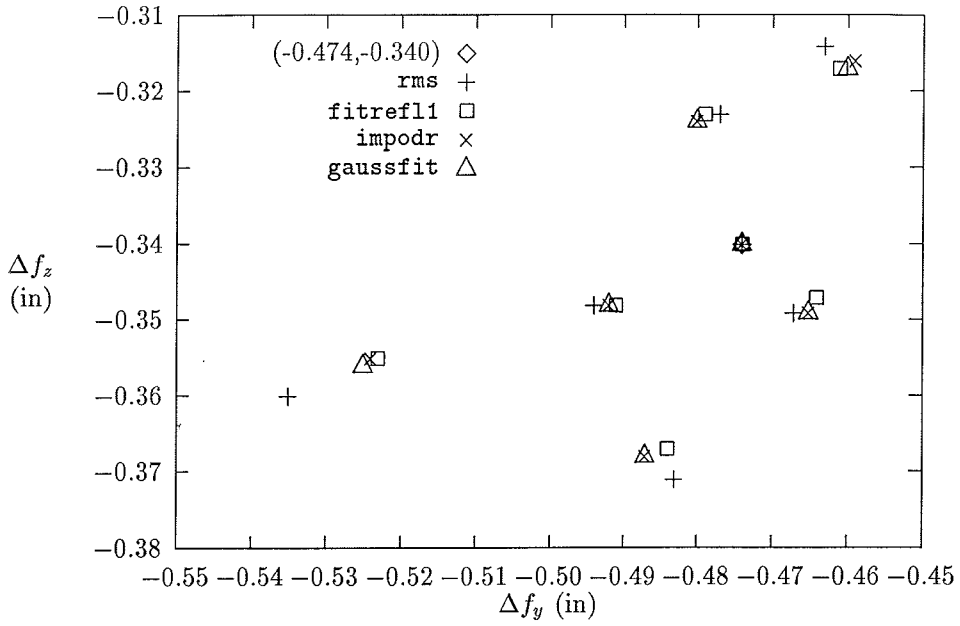


Figure 1: Prime-Focal-Point solutions for the `testgen`*i* datasets

ODRPACK expresses the implicit-ODR BFP problem with code which is essentially identical to the code in the function used in `gaussfit` (described below; the code is shown in Appendix E). In addition to computing the function values, the subroutine computes the necessary partial derivatives using expressions which were produced by Mathematica (Fred Schwab advised on the analytic differentiation). It is unclear whether the analytic derivatives have a performance advantage over the usual numerical differentiation in this case, because of the complexity of the expressions.

**gaussfit** This is an AWK program which drives the "GaussFit" program which was developed circa 1988 by the Astrometry Team of the Hubble Space Telescope project [JFMM88]. GaussFit is a versatile general-purpose function-fitting application which is quite convenient. The user supplies the function, expressed in an elegant C-like language; the function used for this project is shown in Appendix E (see p.25). This function is capable of fitting arbitrary conic sections, not just paraboloids. GaussFit *compiles* this function; it uses analytic differentiation to compute the Jacobians and other partial derivatives needed for the regression problem. GaussFit automatically uses an ODR algorithm whenever, as in this case, the independent variables are asserted to have errors. Even though GaussFit executes the user-supplied function *interpretively*, it is a comparatively fast program.

The four programs operating on the four datasets produced 172 ($43 \times 4$) BFP solutions, which were collected into a single table to facilitate various intercomparisons.

## 2.1 Test generator results

The four sets of prime-focal-point solutions for the `testgen`*i* datasets are plotted in Figure 1; the coordinates are relative to the prime-focal-point of the undistorted 60 meter paraboloid at $E = 44°$ (the rigging angle). All four programs produce the correct answer ( -0.474 -0.340 ) for the noiseless test problem. Some minor differences are seen for the noisy cases. In particular, `rms` differs from the three ODR programs; this is

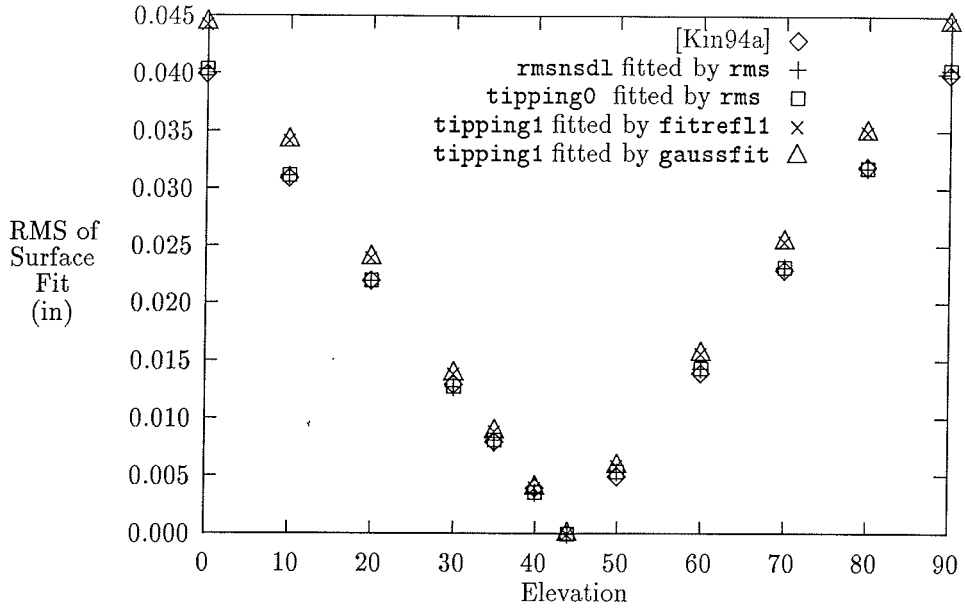


Figure 2: RMS-of-Fit as a function of Elevation

probably due to different weighting of the noisy data points (discussed in Section 2.2). The largest difference is about 0.02 inch, six times smaller than the ±3 mm (0.12 inch) GBT construction tolerance and, relative to the focal length, a difference of about $8 \times 10^{-6}$ (about one part in 120000).

Gaussian noise of ±0.04 inch in the datapoints appears to produce a similar dispersion in the prime-focal-point solutions. The dispersion of the solution points appears to be elliptical rather than circular. This elongation is presumably due to the correlations between the parameters of the BFP solutions. The correlation matrix for $E = 60°$ computed by program GaussFit for the `tipping1` (full model) dataset is

| | $r_x$ | $v_z$ | $v_y$ | $F$ |
|---|---|---|---|---|
| $r_x$ | 1.00 | 0.43 | 0.96 | 0.81 |
| $v_z$ | | 1.00 | 0.63 | -0.01 |
| $v_y$ | | | 1.00 | 0.61 |
| $F$ | | | | 1.00 |

We see that parameters $v_y$ and $r_x$ are highly correlated[2] in the solutions, and other combinations also have large correlations. The correlated errors in these BFP parameters are combined in the formulae for $f_y$ and $f_z$ (see the last two statements of `gbt_paraboloid.c` in Appendix A).

## 2.2 RMS of the fits

All four programs report the RMS of the fit to be zero for the noiseless test problem. But for all cases containing noise, program `rms` consistently reports smaller RMS values than do the other three programs,

[2] "**correlation coefficient** — A measurement, which is unchanged by both addition and multiplication of the random variable by positive constants, of the tendency of two random variables $X$ and $Y$ to vary together; it is given by the ratio of the covariance of $X$ and $Y$ to the square root of the product of the variance of $X$ and the variance of $Y$." [JFMM88, p.69]

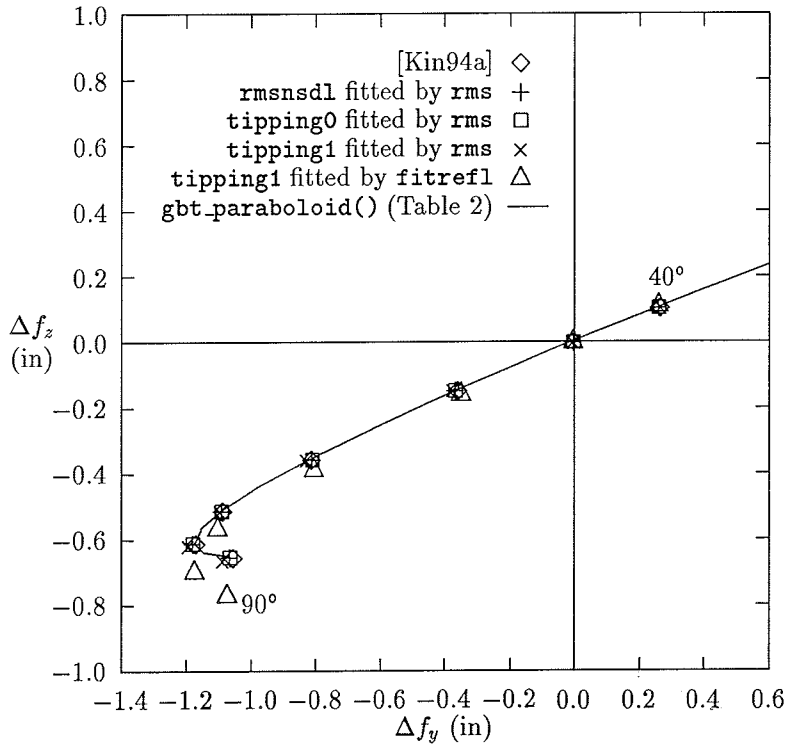


Figure 3: Six Prime-Focal-Point Trajectories

which generally report exactly the same RMS (see Figure 2). The RMS reported by `rms` when fitting the `tipping1` dataset is 10-12% smaller than the RMS of the same dataset being fitted by `gaussfit` or `fitrefl1`. Similarly, for the first of the three test cases with 0.04 inch noise in the XYZ coordinates, programs `fitrefl`, `impodr` and `gaussfit` all report $RMS = 0.0400$ while `rms` reports $RMS = 0.0354$, 11% smaller. This discrepancy is due to the fact that `rms` is minimizing the *wavefront* error, not the fit of the paraboloid to the data. It is shown in [KS66] that half the pathlength change due to surface deformation is given by

$$\frac{1}{2}\Delta PL = \delta \cos\phi/2, \tag{1}$$

where $\phi$ is the angle between the paraboloid axis and the point being fitted subtended at the focal point, and $\delta$ is the "normal component of the distortion vector" (i.e., the orthogonal distance residual). Program `rms` computes the RMS using the formula

$$RMS = \sqrt{\frac{\sum(\Delta PL_i)^2 A_i}{4\sum A_i}}, \tag{2}$$

where the weights $A_i$ are unity in these calculations. We see that this RMS summation is an ODR RMS which is weighted by $\cos^2\phi/2$. Calculation shows that the RMS of this function over the circular area offset by 54 meters from the axis is 0.89, consistent with the observed 11% difference in the solutions. In summary, the three ODR programs weight residuals uniformly while `rms` emphasizes the residuals closest to the axis of the paraboloid, where surface residuals project almost completely into wavefront residuals.

### 2.3 Comparison with previous BFP solutions

King [Kin94a] produced preliminary BFP parameters which were distributed within the GBT project, and which were used by Srikanth [Sri94] for optical design calculations. In Figure 3 we show how the preliminary BFP results for the trajectory of the prime focal point compare with the new solutions. We see that the new solutions by program `rms` agree well with the preliminary ones, but differ by slightly more than 0.1 inch from the solutions produced by the ODR programs for $E = 90°$. This difference is probably produced by the different weighting of the fits to the distorted surface, as discussed above in Section 2.2.

## 3 Which BFP solution to use?

We must choose one of the sets of solutions to use for the GBT. The actuators are normal to the surface; therefore the ODR solutions would minimize the actuator travel as a function of elevation. However, the wavefront (pathlength) solution produced by `rms` will be the best predictor of the prime focal point during the initial period of operation of the GBT when the actuators are not yet functional. (The additional actuator travel required for the `rms` solution is small, and has already been allowed for in the actuator design.) The solutions produced from the full-surface (`tipping1`) dataset are slightly different from those produced from the half-surface dataset, because the nodes on the plane of symmetry appear only once in the regression while other nodes appear twice, producing different weighting. Therefore, strictly speaking, once the actuators are operational, the ideal BFP solution would be produced by one of the three ODR programs operating on dataset `tipping1`. However, the differences between all of the solutions are small, and the GBT will operate satisfactorily with any of them. We have decided that, because preliminary versions of the BFP solutions using program `rms` and dataset `rmsnsdl` (similar to dataset `tipping0`) have been used in analyses of the GBT optics [Nor95, Sri94], there is some value in (almost) maintaining consistency with the prior BFP results. For this reason, we have decided to select dataset `tipping0` fitted by program `rms` to provide the BFP for the GBT.

Table 1: Program which computes Table 2

```
/* This program, paraboloid_tab_bfp.c, uses the gbt_paraboloid()
   function to compute the BFP table of the BFP memo.
   D.Wells, NRAO-CV, Dec94-Mar95 */

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

main()
{
    FILE *t1;
    double el, vxyz[3], rx, fl, bfp_xyz[3], bfp_dxyz[3];

    t1 = fopen("paraboloid_tab_bfp.tex", "w"); /* BFP parameters table */
    for (el = 0.0; el < 90.1; el += 5.0) {
        bfp_xyz[0] = bfp_dxyz[0] = 0.0;
        gbt_paraboloid (el, vxyz, &rx, &fl, bfp_dxyz, bfp_xyz);
        fprintf(t1,
                "%2.0lf&%6.3lf&%6.3lf&%9.6lf&%9.3lf&%6.3lf&%6.3lf&%8.2lf&%8.2lf\\\\\n",
                el, vxyz[1], vxyz[2], rx, fl,
                bfp_dxyz[1], bfp_dxyz[2],
                bfp_xyz[1], bfp_xyz[2]);
    }
    fclose(t1);
    exit(EXIT_SUCCESS);
}
```

# 4 BFP produced by program `rms` for dataset `tipping0`

A program coded in AWK was used to read a table of the set of BFP solutions produced by program `rms` for dataset `tipping0` , and to invoke GaussFit to fit Chebyshev polynomials as a function of elevation to the columns. The program extracted the Chebyshev coefficients from the GaussFit output, and printed them in the form of a function coded in C. This *computed* function `gbt_paraboloid.c` is shown in Appendix A (see p.21); the `include` file containing its function prototype is shown in Appendix D. This function, which concisely summarizes the BFP analysis, is code which is ready to be incorporated into the GBT M&C applications.

The BFP function `gbt_paraboloid.c` is called by program `paraboloid_tab_bfp.c` (see Table 1) in order to compute Table 2, which gives the results returned by `gbt_paraboloid()` as a function of elevation, at 5° steps.

Figure 4 shows the vertex trajectory produced by `gbt_paraboloid()`, plus the solution values which went into the Chebyshev polynomial fit. The latter demonstrates that the Chebyshev fitting is correct. The preliminary BFP [Kin94a] vertex trajectory is shown for reference.

Figure 5 shows the focal length of the BFP as a function of elevation. The axis tilt as a function of elevation is shown in Figure 6. This tilt angle is the *a priori* estimate of the gravitational term of the "traditional" pointing model ($\Delta E$ as function of $E$). The maximum value is about 0.8 milliradians, or 165 arcseconds (2.8 arcminutes).

The trajectory of the prime-focal-point of the BFP is shown in Figure 7. This trajectory is computed by function `gbt_paraboloid()` from a combination of the other BFP parameters (see the expressions which compute `dxyz[]` in the last few lines of Appendix A). Note that $f_y$ and $f_z$ in Table 2 (which are `xyz[]` in Appendix A) are the coordinates of the prime focal point at the rigging angle, expressed in alidade coordinates (the system of the structural model).

Table 2: BFP parameters returned as function of elevation

| | Vertex | | Tilt | Focal Length | Prime Focal Point | | | |
|---|---|---|---|---|---|---|---|---|
| | Figure 4 | | Figure 6 | Figure 5 | Figure 7 | | | |
| $E$ (d) | $v_y$ (in) | $v_z$ (in) | $r_x$ (rad) | $F$ (in) | $\Delta f_y$ (in) | $\Delta f_z$ (in) | $f_y$ (in) | $f_z$ (in) |
| 0 | 5.912 | 1.213 | 0.000800 | 2362.383 | 4.021 | 1.395 | -2159.02 | 4459.05 |
| 5 | 5.010 | 1.090 | 0.000648 | 2362.333 | 3.480 | 1.223 | -2159.02 | 4459.05 |
| 10 | 4.153 | 0.961 | 0.000510 | 2362.290 | 2.949 | 1.051 | -2159.02 | 4459.05 |
| 15 | 3.349 | 0.827 | 0.000387 | 2362.254 | 2.436 | 0.881 | -2159.02 | 4459.05 |
| 20 | 2.603 | 0.688 | 0.000279 | 2362.226 | 1.943 | 0.714 | -2159.02 | 4459.05 |
| 25 | 1.921 | 0.547 | 0.000188 | 2362.205 | 1.477 | 0.552 | -2159.02 | 4459.05 |
| 30 | 1.309 | 0.403 | 0.000114 | 2362.192 | 1.039 | 0.395 | -2159.02 | 4459.05 |
| 35 | 0.770 | 0.259 | 0.000058 | 2362.188 | 0.634 | 0.246 | -2159.02 | 4459.05 |
| 40 | 0.310 | 0.114 | 0.000019 | 2362.191 | 0.265 | 0.106 | -2159.02 | 4459.05 |
| 45 | -0.069 | -0.029 | -0.000001 | 2362.203 | -0.067 | -0.026 | -2159.02 | 4459.05 |
| 50 | -0.364 | -0.169 | -0.000002 | 2362.222 | -0.358 | -0.147 | -2159.02 | 4459.05 |
| 55 | -0.572 | -0.306 | 0.000015 | 2362.250 | -0.608 | -0.257 | -2159.02 | 4459.05 |
| 60 | -0.691 | -0.439 | 0.000052 | 2362.285 | -0.814 | -0.354 | -2159.02 | 4459.05 |
| 65 | -0.722 | -0.566 | 0.000107 | 2362.327 | -0.974 | -0.439 | -2159.02 | 4459.05 |
| 70 | -0.663 | -0.687 | 0.000180 | 2362.376 | -1.089 | -0.511 | -2159.02 | 4459.05 |
| 75 | -0.515 | -0.800 | 0.000271 | 2362.432 | -1.156 | -0.568 | -2159.02 | 4459.05 |
| 80 | -0.279 | -0.905 | 0.000379 | 2362.494 | -1.175 | -0.611 | -2159.02 | 4459.05 |
| 85 | 0.043 | -1.001 | 0.000503 | 2362.562 | -1.145 | -0.639 | -2159.02 | 4459.05 |
| 90 | 0.448 | -1.087 | 0.000640 | 2362.635 | -1.065 | -0.652 | -2159.02 | 4459.05 |

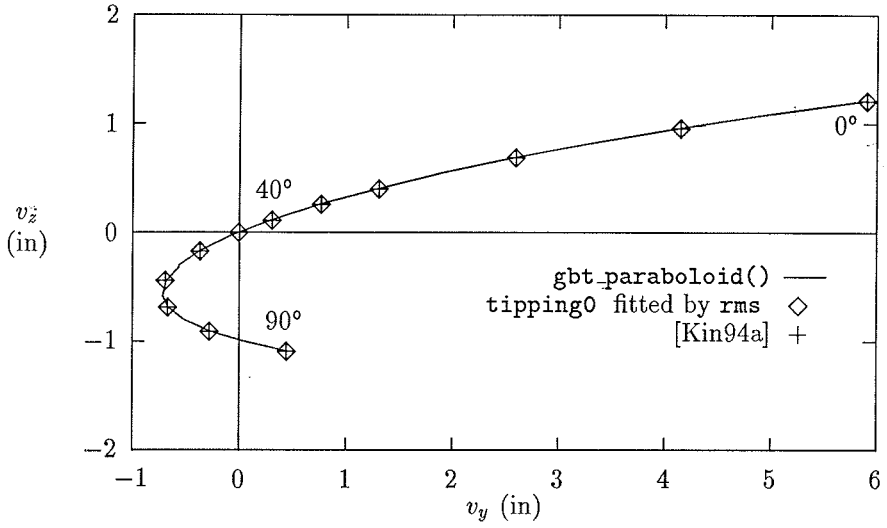


Figure 4: Paraboloid-Vertex trajectory as a function of Elevation

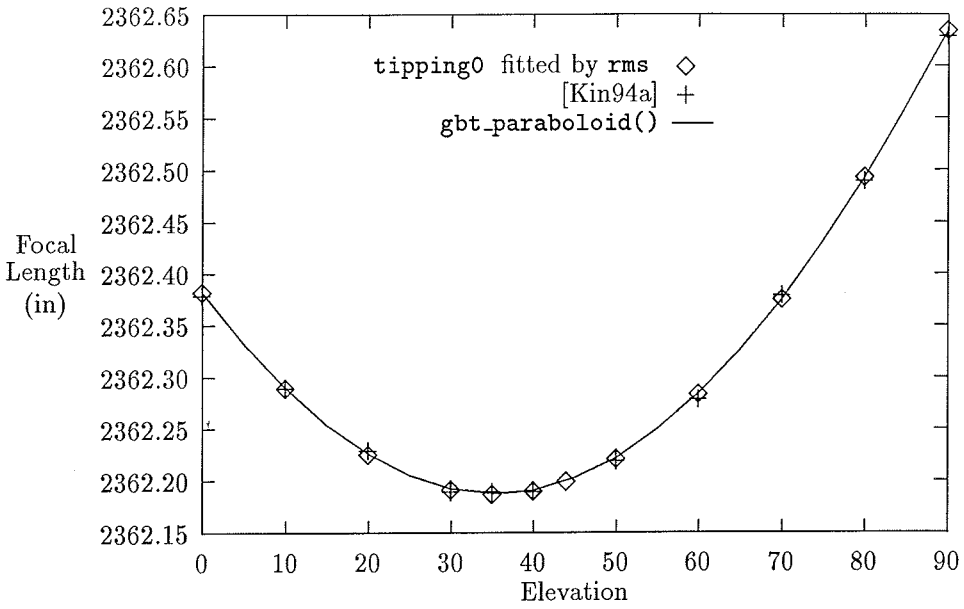


Figure 5: Focal length as a function of Elevation

# 5 Applications

The GBT Best-Fitting-Paraboloid [BFP] function `gbt_paraboloid()` is expected to be used in the M&C system for the following purposes:

**Gravity Pointing Term** 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 BFP will be the primary component of the "commanded surface" in the open-loop active surface module. This component of the open-loop active surface control for Phase-2 of the project will consist of driving the actuators with the difference between the displacements computed by the structural model and the BFP paraboloid, projected to local surface normal.

**Focus Tracking** The focus tracking subsystems will compute displacements of the prime focus and Gregorian optics relative to the prime focal point of the BFP, and will use these to maintain the optics in proper alignment for optimum gain.

In the remainder of this section, we provide three numerical examples of the application of the BFP to particular cases.

## 5.1 Relative Motions of the BFP and Prime Focus Box

The focus tracking subsystem for the GBT prime focus will move the prime focus box to correct for the gravitational deflections of the prime focus box relative to the BFP prime focal point. The program shown in Table 3 calculates this relative motion. In the loop on elevation, it reads a line of Table 2 to get the trajectory of the prime focal point of the BFP. It then calls `get_node_data()` (see [WK95]) to get the gravitational

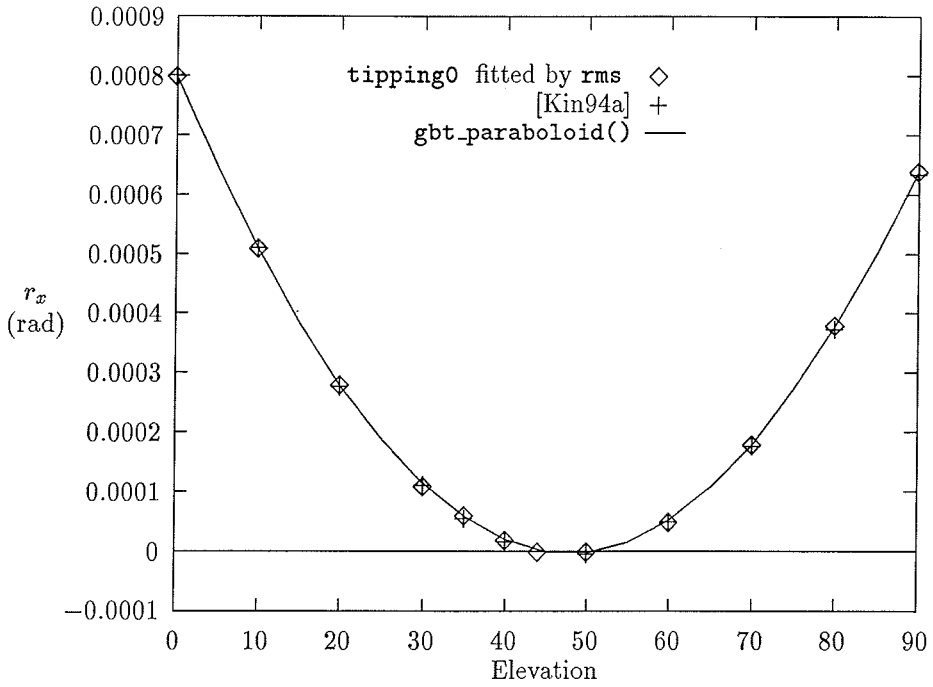


Figure 6: Axis Tilt as a function of Elevation

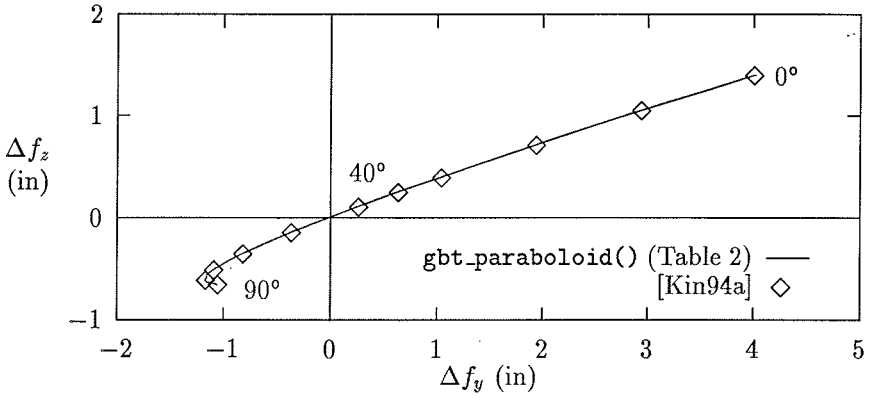


Figure 7: Prime-Focal-Point trajectory as a function of Elevation

Table 3: Program which computes Table 4

```
/* Program paraboloid_tab_pfb.c uses the BFP table (from
   gbt_paraboloid()) plus the structural model functions to compute
   the prime-focus-box table of the BFP memo.
   D.Wells, NRAO-CV, Mar95 */

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

main()
{
    const int pfid = 50000;      /* prime-focus-box node-ID */
    const double dtr = 0.017453293, mti = 39.37, box_angle = 45.5 * dtr;
    FILE *t1, *t3;
    int pfi, i;
    double
        el, el1, vy, vz, rx, fl, bfp_xyz[3], bfp_dxyz[3], pfb[3],
        box_rotation, box[3];
    struct node_data pfi_data;

    t1 = fopen("paraboloid_tab_bfp.tex",  "r"); /* BFP parameters  table */
    t3 = fopen("paraboloid_tab_pfb.tex",  "w"); /* prime focus box table */
    if ((pfi = get_index(pfid)) == 0) exit(EXIT_FAILURE);

    for (el = 0.0; el < 90.1; el += 5.0) {
        bfp_xyz[0] = bfp_dxyz[0] = 0.0;
        fscanf(t1, "%2lf&%lf&%lf&%lf&%lf&%lf&%lf&%lf&%lf\\\\\n",
               &el1, &vy, &vz, &rx, &fl,
               &bfp_dxyz[1], &bfp_dxyz[2], &bfp_xyz[1], &bfp_xyz[2]);
        if (el1 != el) {
            printf("el1=%lg, el=%lg, NOT EQUAL!?!\n", el1, el);
            printf("%2.0lf&%6.3lf&%6.3lf&%8.5lf&%7.4lf&%9.3lf&%6.3lf&%6.3lf\\\\\n",
                    el, vy, vz, rx, fl, bfp_dxyz[1], bfp_dxyz[2]);
            exit(EXIT_FAILURE);
        }
        fprintf(t3, "%2.0lf", el);
        if (get_node_data(pfi, el, &pfi_data)) exit(EXIT_FAILURE);
        fprintf(t3, "&%5.2f&%5.2f&%8.5f", pfi_data.at_elev.delta[1],
                pfi_data.at_elev.delta[2], pfi_data.at_elev.tilt[0]);
        for (i=0; i<3; i++) {
            pfb[i] = (bfp_xyz[i] + bfp_dxyz[i])
                   - ((double)pfi_data.grid[i] + pfi_data.at_elev.delta[i]);
        }
        fprintf(t3, "&%5.2lf&%5.2lf", pfb[1], pfb[2]);
        box_rotation = box_angle + pfi_data.at_elev.tilt[0];
        box[0] =  cos(box_rotation) * pfb[1] + sin(box_rotation) * pfb[2];
        box[1] = -sin(box_rotation) * pfb[1] + cos(box_rotation) * pfb[2];
        box[2] = pfb[0];
        fprintf(t3, "&%5.3lf&%5.3lf", box[0], box[1]);
        fprintf(t3, "\\\\\n");
    }
    fclose(t1); fclose(t3);
    exit(EXIT_SUCCESS);
}
```

Table 4: Focus Tracking for the prime focus box

| | Prime Boom (node 50000) | | | PF-Box Focus Tracking | | | |
|---|---|---|---|---|---|---|---|
| | Figure 8 | | | Figure 8 | | Figure 9 | |
| $E$ (d) | $\Delta p_y$ (in) | $\Delta p_z$ (in) | $t_x$ (rad) | $\Delta b_y$ (in) | $\Delta b_z$ (in) | $B_x$ (in) | $B_y$ (in) |
| 0 | 5.86 | 2.51 | -0.00119 | -1.84 | -1.12 | -2.095 | 0.525 |
| 5 | 5.44 | 2.21 | -0.00114 | -1.96 | -0.99 | -2.080 | 0.697 |
| 10 | 4.94 | 1.90 | -0.00107 | -1.99 | -0.86 | -2.008 | 0.811 |
| 15 | 4.37 | 1.60 | -0.00098 | -1.93 | -0.73 | -1.876 | 0.865 |
| 20 | 3.74 | 1.30 | -0.00086 | -1.80 | -0.60 | -1.687 | 0.859 |
| 25 | 3.05 | 1.01 | -0.00072 | -1.57 | -0.47 | -1.439 | 0.792 |
| 30 | 2.31 | 0.73 | -0.00056 | -1.27 | -0.34 | -1.136 | 0.664 |
| 35 | 1.52 | 0.46 | -0.00038 | -0.89 | -0.22 | -0.779 | 0.477 |
| 40 | 0.69 | 0.20 | -0.00017 | -0.42 | -0.10 | -0.370 | 0.232 |
| 45 | -0.18 | -0.05 | 0.00005 | 0.11 | 0.01 | 0.084 | -0.069 |
| 50 | -1.07 | -0.28 | 0.00028 | 0.71 | 0.12 | 0.583 | -0.424 |
| 55 | -1.98 | -0.49 | 0.00053 | 1.38 | 0.22 | 1.121 | -0.829 |
| 60 | -2.91 | -0.68 | 0.00079 | 2.10 | 0.31 | 1.694 | -1.281 |
| 65 | -3.85 | -0.85 | 0.00106 | 2.88 | 0.40 | 2.298 | -1.778 |
| 70 | -4.79 | -0.99 | 0.00134 | 3.70 | 0.47 | 2.927 | -2.315 |
| 75 | -5.72 | -1.11 | 0.00163 | 4.57 | 0.54 | 3.578 | -2.887 |
| 80 | -6.64 | -1.21 | 0.00192 | 5.47 | 0.59 | 4.245 | -3.492 |
| 85 | -7.54 | -1.28 | 0.00221 | 6.39 | 0.63 | 4.924 | -4.125 |
| 90 | -8.41 | -1.33 | 0.00250 | 7.34 | 0.67 | 5.611 | -4.782 |

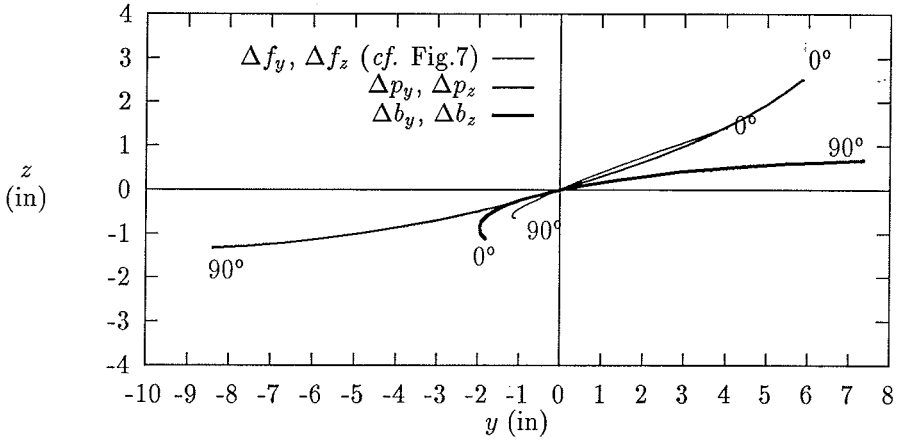


Figure 8: Trajectories of BFP, PF-boom and difference

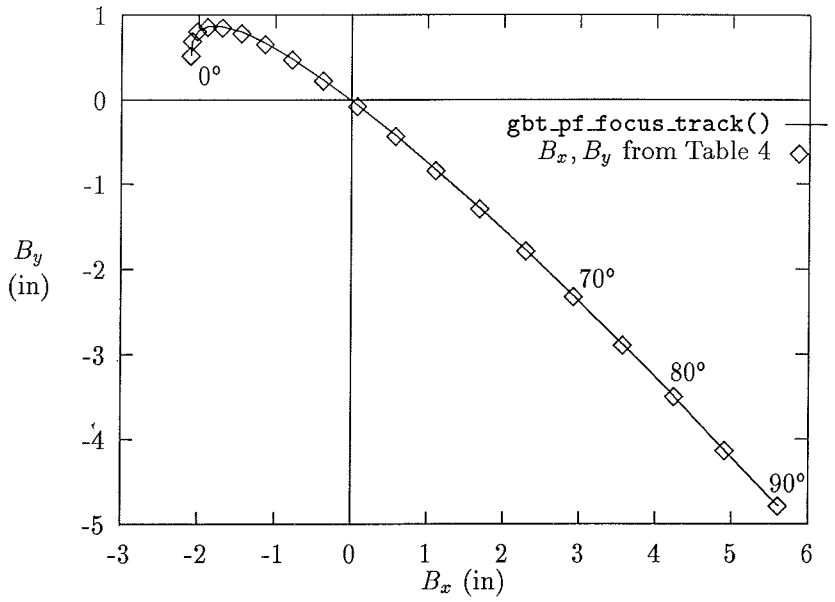


Figure 9: Prime-Focus-Box focus-tracking trajectory

deflection of the prime focus box (node 50000) at the tip of the prime focus boom. The program computes the difference between these as $\Delta b_y, \Delta b_z$ in Table 4.[3] These three trajectories are plotted in Figure 8.

The prime focus box actuators are oriented at an angle of 45.5° relative to the coordinate system of the tipping structure [Kin94b, KM93], so the $\Delta b_y, \Delta b_z$ trajectory must be rotated to the prime focus coordinate system. The appropriate coordinate rotation includes the gravitational tilt of the tip of the prime focus boom $t_x$ (see Table 4 and the expression for `box_rotation` in Table 3). For $E = 90°$, the 2.5 mrad value of $t_x$ displaces the trajectory by only about 0.02 inch (0.5 mm), negligible in comparison to wavelengths like 20 cm. The rotated coordinates are `box[i]`, $B_x, B_y$ in Table 4; note that the boom displacement is in $y, z$ coordinates, while the box actuators are called the $x, y$ axes. Figure 9 shows the focus-tracking trajectory of the prime focus box, *expressed in the prime-focus coordinate system* [Kin94b, KM93].

An AWK program reads Table 4 and fits Chebyshev polynomials to the $B_x$ and $B_y$ columns, and outputs the coefficients as a function `gbt_pf_focus_track()` expressed in the C language; see Appendix B (p.23). This module is ready to be installed in the prime focus tracking application of the M&C system. The trajectory of $B_x$ appears to be well matched to the required travel range of the X-axis of the prime focus feed positioning mechanism, ±7.22 inch. The Y-axis required range is ±20.71 inch, and $B_y$ fits into this with room to spare. The GBT elevation velocity limit is 0.25 deg/s and the $y_p$ ($B_y$) velocity is 6 in/min. For a slew from the rigging angle 44° to the zenith (the worst case), the elevation drive will need about 184 sec while the prime focus box will need only about 47 sec.

## 5.2 Relative Motions of the BFP, subreflector and Feedhorn

The Gregorian focus tracking subsystem will move the subreflector to correct for the relative deflections of the BFP, subreflector and Gregorian feedhorn. The program shown in Table 5 calculates these deflections;

[3] Displacement $\Delta p_x$ and tilts $t_y$ and $t_z$ are not shown because they are always zero for these nodes, which are in the meridional plane, the plane of symmetry of the GBT.

Table 5: Program which computes Table 6

```
/* Program paraboloid_tab_greg.c uses function gbt_paraboloid() plus
   the structural model functions to compute the
   subreflector/Gregorian_focus table of the BFP memo.
   D.Wells, NRAO-CV, Mar95 */
#include <stdlib.h>
#include <math.h>
#include "structural_model.h"
#include "paraboloid.h"
main() {
    const int svid = 50005,      /* subreflector-vertex  ID */
              gfid = 40700;      /* Gregorian-feed  node-ID */
    const double dtr = 0.017453293, mti = 39.37, rig_angle = 44.0;
    int svi, gfi, i, j, status;
    double start, el, vxyz[3], rx, fl, dl, dt, dt0,
           stilt, ctilt, rotate[3][3], svf1[3], svf1p[3],
           bfp_xyz[3], bfp_dxyz[3], f12[3], dp[3];
    struct node_data svi_data, gfi_data;

    if ((svi = get_index(svid)) == 0) exit(EXIT_FAILURE);
    if ((gfi = get_index(gfid)) == 0) exit(EXIT_FAILURE);
    for (el = start = -5.0; el < 90.1; el += 5.0) {
        if (el == start) { el = rig_angle; dt0 = 0.0; } /* initialization */
        if (el != rig_angle) printf("%2.0lf", el);
         bfp_dxyz[0] = bfp_xyz[0] = 0.0;
        gbt_paraboloid (el, vxyz, &rx, &fl, bfp_dxyz, bfp_xyz);
        if (get_node_data(svi, el, &svi_data)) exit(EXIT_FAILURE);
        if (el != rig_angle)
            printf("&%5.2f&%5.2f&%8.5f", svi_data.at_elev.delta[1],
                   svi_data.at_elev.delta[2], svi_data.at_elev.tilt[0]);
        if (get_node_data(gfi, el, &gfi_data)) exit(EXIT_FAILURE);
        if (el != rig_angle) printf("&%5.2f&%5.2f",
                    gfi_data.at_elev.delta[1], gfi_data.at_elev.delta[2]);
        for (i=0, dl=0.0; i<3; i++) {
            f12[i] = (bfp_xyz[i] + bfp_dxyz[i])
                   - ((double)gfi_data.grid[i] + gfi_data.at_elev.delta[i]);
            dl += f12[i]*f12[i];
        }
        dl = sqrt (dl) - (11.0 * mti);
        dt = atan2 (f12[2], f12[1]) - dt0;
        if (el == rig_angle) { dt0 = dt; dt = 0.0; }
        if (el != rig_angle) printf("&%5.2lf&%8.5lf", dl, dt);
        if (el == rig_angle)
            for (i = 0; i < 3; i++)
                svf1[i] = (bfp_xyz[i] + bfp_dxyz[i])
                    - ((double)svi_data.grid[i] + svi_data.at_elev.delta[i]);
        ctilt = cos(svi_data.at_elev.tilt[0]); /* rotation about X-axis */
        stilt = sin(svi_data.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, svf1p[i] = 0.0; j < 3; j++)
                svf1p[i] += svf1[j] * rotate[i][j];
            dp[i] = (bfp_xyz[i] + bfp_dxyz[i])
                - ((double)svi_data.grid[i] + svi_data.at_elev.delta[i]
                                            + svf1p[i]);
        }
        if (el != rig_angle) printf("&%6.3lf&%6.3lf", dp[1], dp[2]);
        if (el != rig_angle) printf("\\\\\n");
        if (el == rig_angle) el = start; /* undo initialization case */
    }
    exit(EXIT_SUCCESS);
}
```

Table 6: Displacements of subreflector and Gregorian feedhorn

| | Subreflèctor (50005) | | | Greg. Feed | | BFP↦Greg. Feed | | Subr.↦BFP | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Figures 10 and 11 | | Figure 12 | |
| $E$ (d) | $\Delta s_y$ (in) | $\Delta s_z$ (in) | $\delta s_x$ (rad) | $\Delta g_y$ (in) | $\Delta g_z$ (in) | $\Delta L_{12}$ (in) | $\Delta\theta_{12}$ (rad) | $\Delta P_y$ (in) | $\Delta P_z$ (in) |
| 0 | 6.19 | 2.92 | -0.00173 | 4.84 | 2.52 | -1.20 | 0.00162 | -1.908 | -1.237 |
| 5 | 5.74 | 2.59 | -0.00165 | 4.48 | 2.22 | -1.09 | 0.00207 | -2.012 | -1.084 |
| 10 | 5.22 | 2.24 | -0.00154 | 4.06 | 1.92 | -0.98 | 0.00236 | -2.034 | -0.933 |
| 15 | 4.62 | 1.90 | -0.00140 | 3.59 | 1.62 | -0.85 | 0.00248 | -1.972 | -0.783 |
| 20 | 3.96 | 1.56 | -0.00123 | 3.07 | 1.33 | -0.72 | 0.00244 | -1.825 | -0.636 |
| 25 | 3.23 | 1.22 | -0.00103 | 2.50 | 1.03 | -0.58 | 0.00224 | -1.596 | -0.492 |
| 30 | 2.45 | 0.88 | -0.00079 | 1.89 | 0.75 | -0.44 | 0.00187 | -1.284 | -0.354 |
| 35 | 1.61 | 0.56 | -0.00053 | 1.24 | 0.47 | -0.29 | 0.00134 | -0.893 | -0.221 |
| 40 | 0.73 | 0.24 | -0.00025 | 0.56 | 0.20 | -0.13 | 0.00066 | -0.426 | -0.095 |
| 45 | -0.19 | -0.06 | 0.00006 | -0.14 | -0.05 | 0.03 | -0.00018 | 0.113 | 0.023 |
| 50 | -1.13 | -0.35 | 0.00039 | -0.87 | -0.29 | 0.19 | -0.00116 | 0.721 | 0.132 |
| 55 | -2.11 | -0.61 | 0.00074 | -1.62 | -0.51 | 0.35 | -0.00227 | 1.391 | 0.232 |
| 60 | -3.10 | -0.86 | 0.00111 | -2.37 | -0.71 | 0.51 | -0.00351 | 2.120 | 0.321 |
| 65 | -4.09 | -1.09 | 0.00148 | -3.13 | -0.90 | 0.67 | -0.00486 | 2.900 | 0.400 |
| 70 | -5.09 | -1.29 | 0.00187 | -3.89 | -1.06 | 0.82 | -0.00632 | 3.726 | 0.466 |
| 75 | -6.08 | -1.47 | 0.00227 | -4.65 | -1.20 | 0.98 | -0.00787 | 4.592 | 0.521 |
| 80 | -7.06 | -1.62 | 0.00267 | -5.39 | -1.32 | 1.13 | -0.00951 | 5.492 | 0.563 |
| 85 | -8.02 | -1.75 | 0.00307 | -6.11 | -1.41 | 1.27 | -0.01122 | 6.419 | 0.592 |
| 90 | -8.95 | -1.85 | 0.00346 | -6.81 | -1.48 | 1.41 | -0.01299 | 7.368 | 0.608 |

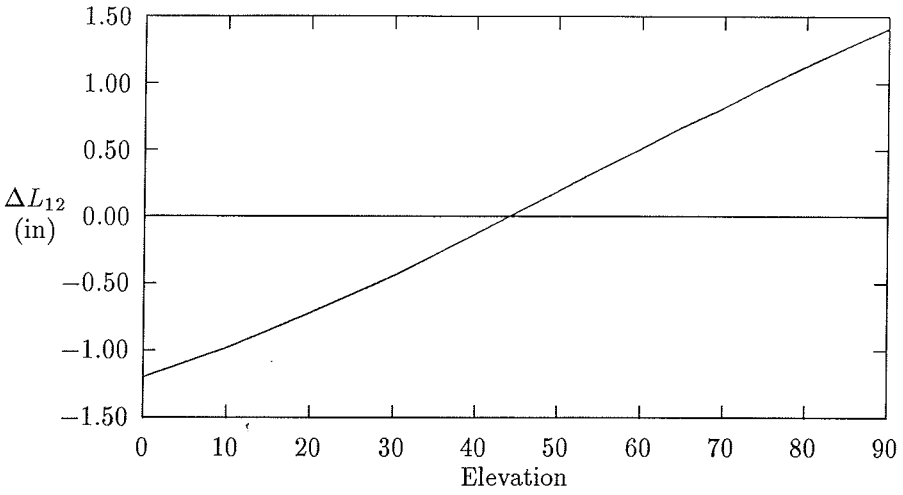


Figure 10: BFP-to-GregorianFeed distance minus 11 meters

its results are shown in Table 6.[4]

Program `paraboloid_tab_greg` also computes the length of the line connecting the prime focal point and Gregorian feedhorn. The difference between this distance and the 11 meter nominal focal length of the ellipsoid is plotted in Figure 10. The rotation of this line is also computed, and is plotted in Figure 11. Note that $\Delta\theta_{12} = -0.01259$ (-0.7 deg) means 4 inch (10 cm) displacement at the end of an 11 meter lever arm for $E = 90°$.

Program `paraboloid_tab_greg` also computes the displacement between the first-focal-point of the ellipsoidal subreflector and the prime-focal-point of the BFP; this is $\Delta P_y, \Delta P_z$ in Table 6. This computation is analogous to the "rigid rod" calculation of the displacement of the second-focal-point relative to the feedhorn in Section 3.2 of [WK95]. It accounts for the tilt of the subreflector as well as its translation in forming the difference, which is plotted in Figure 12.[5] The ellipsoidal subreflector images the region around its first-focal-point onto the region around its second-focal-point (the feedhorn). The mapping involves longitudinal and lateral magnification. The displacement of the BFP prime-focal-point relative to the first-focal-point is thus *magnified* at the feedhorn, producing de-focus (loss of gain) and a pointing offset. Because of the magnification, the displacement shown in Figure 12 is the major factor in Gregorian focus tracking.

## 5.3 Displacements with-respect-to the BFP Axis

The 'prescription' for the Gregorian optics is simplified if it is expressed in a coordinate system attached to the BFP. Such results have been distributed previously by King and used for analyses of the Gregorian optics [Sri94, Table 1(p.4)]. We present a new version of these calculations in Table 7, which has been produced by the program shown in Table 9. King recently distributed a different version of these calculations [Kin95], selected portions of which we show in Table 8. The version shown in Table 7 covers the elevation range in 5° steps, is based on function `get_node_wrt_bfp()` and is relative to the new focal point of the BFP rather than the nominal focal point. The agreement between the two sets of results is good; for example,

[4]Displacement $\Delta s_x$ and tilts $\delta s_y$ and $\delta s_z$ are not shown because they are always zero for these nodes, which are in the meridional plane, the plane of symmetry of the GBT.

[5]This trajectory is expressed in the *elevation coordinate system*, not in the *subreflector coordinate system*, which is rotated to an angle of 36.7° relative to the elevation system [Kin94b, KM93].

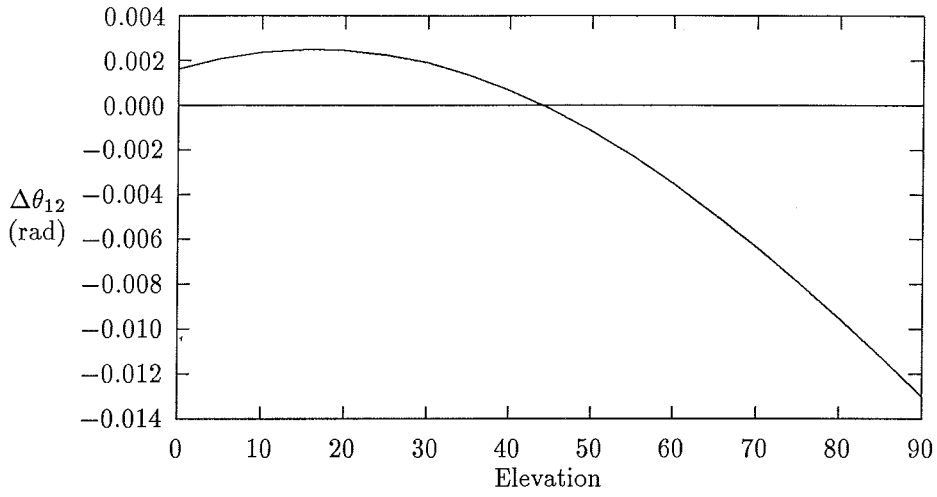


Figure 11: Rotation of the BFP-to-GregorianFeed line

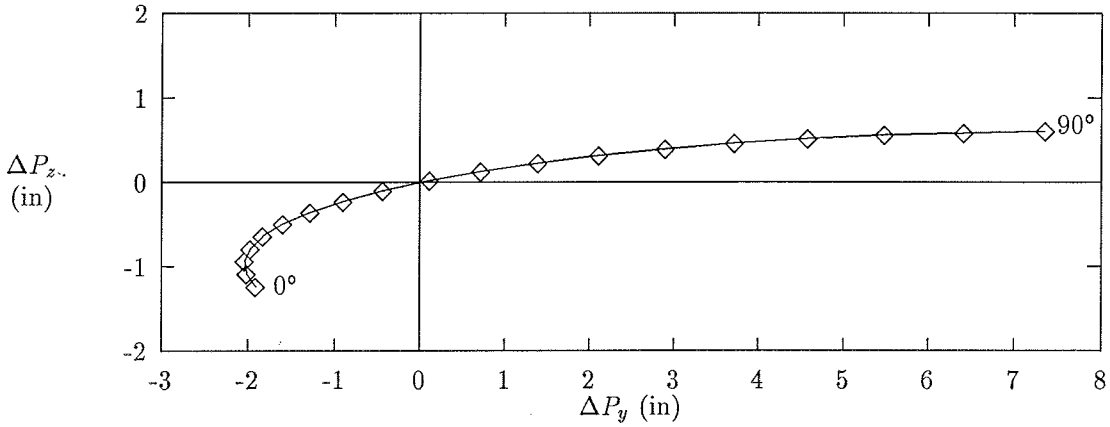


Figure 12: Trajectory of BFP relative to first-focal-point

Table 7: Gravity-induced Deformations wrt the BFP Axis

| | | | Greg. Feed (40700) | | | Subreflector (50005) | | |
|---|---|---|---|---|---|---|---|---|
| $E$ (d) | $\Delta F$ (in) | $r_x$ (rad) | $\Delta g_y$ (in) | $\Delta g_z$ (in) | $\delta g_x$ (rad) | $\Delta s_y$ (in) | $\Delta s_z$ (in) | $\delta s_x$ (rad) |
| 0 | 0.183 | 0.000800 | 0.473 | 1.153 | -0.00295 | 2.291 | 1.662 | -0.00253 |
| 5 | 0.133 | 0.000648 | 0.720 | 1.024 | -0.00265 | 2.361 | 1.471 | -0.00230 |
| 10 | 0.090 | 0.000510 | 0.893 | 0.891 | -0.00234 | 2.345 | 1.279 | -0.00205 |
| 15 | 0.054 | 0.000387 | 0.988 | 0.758 | -0.00202 | 2.244 | 1.084 | -0.00179 |
| 20 | 0.026 | 0.000279 | 1.005 | 0.624 | -0.00168 | 2.055 | 0.890 | -0.00151 |
| 25 | 0.005 | 0.000188 | 0.943 | 0.490 | -0.00134 | 1.781 | 0.697 | -0.00121 |
| 30 | -0.008 | 0.000114 | 0.802 | 0.357 | -0.00099 | 1.423 | 0.507 | -0.00091 |
| 35 | -0.012 | 0.000058 | 0.584 | 0.226 | -0.00063 | 0.985 | 0.321 | -0.00059 |
| 40 | -0.009 | 0.000019 | 0.291 | 0.099 | -0.00028 | 0.469 | 0.140 | -0.00026 |
| 45 | 0.003 | -0.000001 | -0.076 | -0.024 | 0.00007 | -0.120 | -0.034 | 0.00006 |
| 50 | 0.022 | -0.000002 | -0.513 | -0.142 | 0.00041 | -0.777 | -0.199 | 0.00040 |
| 55 | 0.050 | 0.000015 | -1.017 | -0.253 | 0.00075 | -1.497 | -0.355 | 0.00073 |
| 60 | 0.085 | 0.000052 | -1.582 | -0.358 | 0.00108 | -2.274 | -0.499 | 0.00106 |
| 65 | 0.127 | 0.000107 | -2.206 | -0.455 | 0.00139 | -3.102 | -0.632 | 0.00138 |
| 70 | 0.176 | 0.000180 | -2.884 | -0.543 | 0.00169 | -3.975 | -0.751 | 0.00169 |
| 75 | 0.232 | 0.000271 | -3.609 | -0.622 | 0.00197 | -4.887 | -0.857 | 0.00200 |
| 80 | 0.294 | 0.000379 | -4.378 | -0.690 | 0.00223 | -5.830 | -0.947 | 0.00229 |
| 85 | 0.362 | 0.000503 | -5.185 | -0.749 | 0.00247 | -6.799 | -1.022 | 0.00256 |
| 90 | 0.435 | 0.000640 | -6.024 | -0.796 | 0.00268 | -7.787 | -1.081 | 0.00282 |

Table 8: Optics movements under gravity loads wrt BFP (from [Kin95])

| | | | Node 40700 | | Node 50005 | |
|---|---|---|---|---|---|---|
| $E$ (d) | $\Delta F$ (in) | $r_x$ (rad) | $\Delta g_y$ (in) | $\Delta g_z$ (in) | $\Delta s_y$ (in) | $\Delta s_z$ (in) |
| 0 | 0.183 | 0.000802 | 0.477 | 1.337 | 2.296 | 1.845 |
| 90 | 0.435 | 0.000635 | -6.033 | -0.361 | -7.799 | -0.646 |

$\Delta g_z = 1.337$ for $E = 0$ in Table 8 should be compared with $\Delta g_z + \Delta F = 0.183 + 1.153 = 1.336$ in Table 7 and $\Delta g_y = 0.477$ should be compared with 0.473.

The results in Table 7 are ready to use in optical calculations, with one exception: node 50005 is the *center* of the off-axis ellipsoid, not its vertex. The optical prescription needs the displaced location of the ellipsoid vertex and the tilt of the ellipsoid axis. The tilt is $\delta s_x$ and the trajectory of the ellipsoid vertex can be obtained by combining $\Delta s_y$, $\Delta s_z$ and $\delta s_x$ in a calculation analogous to the calculation of the trajectory of the first focal point, as discussed in Section 5.2, which uses code shown in Table 5.

# 6 Availability

The complete package of code for the BFP is available at the URL

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

Table 9: Program which computes Table 7

```
/* Program paraboloid_tab_wrt.c uses structural model function
   get_node_wrt_bfp() to compute a table of trajectories of two nodes
   with respect to the BFP axis.  D.Wells, NRAO-CV, May95 */

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

main()
{
    const int svid = 50005, gfid = 40700;
    int svi, gfi;
    double el, vxyz[3], rx, fl, bfp_dxyz[3], bfp_xyz[3];
    char form[100];
    struct node_data svi_data, gfi_data;

    if ((svi = get_index(svid)) == 0) exit(EXIT_FAILURE);
    if ((gfi = get_index(gfid)) == 0) exit(EXIT_FAILURE);
    for (el = 0.0; el < 90.1; el += 5.0) {
        gbt_paraboloid (el, vxyz, &rx, &fl, bfp_dxyz, bfp_xyz);
        if (get_node_wrt_bfp(svi, el, &svi_data)) exit(EXIT_FAILURE);
        if (get_node_wrt_bfp(gfi, el, &gfi_data)) exit(EXIT_FAILURE);
        printf("%2.0lf&%5.3lf&%8.6lf&%6.3lf&%6.3lf&%8.5lf&%6.3lf&%6.3lf&%8.5lf \\\\\n",
               el, (fl - 60.0*39.37), rx, gfi_data.at_elev.delta[1],
               gfi_data.at_elev.delta[2], gfi_data.at_elev.tilt[0],
               svi_data.at_elev.delta[1],
               svi_data.at_elev.delta[2], svi_data.at_elev.tilt[0]);
    }
    exit(EXIT_SUCCESS);
}
```

This compressed "tar" file is about 120 kilobytes in length. Use "`gunzip`" to decompress it. In addition to the code, it contains a Makefile and several testcases for compiling and verifying the code. It also contains a copy of this GBT memo in Postscript.

# References


[BBRS92] Paul T. Boggs, Richard H. Byrd, Janet E. Rogers, and Robert B. Schnabel. *User's Reference Guide for ODRPACK Version 2.01—Software for Weighted Orthogonal Distance Regression.* National Institute of Standards and Technology, Gaithersburg, MD 20899, June 1992. ODRPACK is a software package for *weighted orthogonal distance regression*, i.e., for finding the set of parameters that minimize the sum of the weighted orthogonal distances from a set of observations to the curve or surface determined by the parameters. It can also be used to solve the nonlinear ordinary least squares problem. See URL=`http://netlib.att.com/magic/netlib_find?db=0&pat=odrpack` (or URL=`ftp://netlib.att.com/netlib/odrpack/`). Also see [BBS87] and [BDBS89]. This is report NISTIR 92-4834.

[BBS87] Paul T. Boggs, Richard H. Byrd, and Robert B. Schnabel. A stable and efficient algorithm for nonlinear orthogonal distance regression. *SIAM Journal on Scientific and Statistical Computing*, 8(6):1052–1078, November 1987.

[BDBS89] P. T. Boggs, J. R. Donaldson, R. H. Byrd, and R. B. Snabel. Algorithm 676, ODRPACK – software for weighted orthogonal distance regression. *ACM Transactions On Mathematical Software*, 15:348–364, 1989.

[D'A89] L. R. D'Addario. Active surface adjustment to nominal vs. nearby paraboloid. GBT Memo 20, National Radio Astronomy Observatory, October 1989.

[GKHK75] W. Gellert, H. Küstner, M. Hellwich, and H. Kästner. *The VNR Concise Encyclopedia of Mathematics.* Van Nostrand Reinhold Company, 1975.

[JFMM88] William H. Jefferys, Michael J. Fitzpatrick, Barbara E. McArthur, and James E. McCartney. *User's Manual – GaussFit: A system for least squares and robust estimation.* The University of Texas at Austin, 1.0-12/2/88 edition, December 1988. See URL=`ftp://clyde.as.utexas.edu/pub/gaussfit/`.

[Kin94a] Lee King. GBT BFP parameters. Email memo to `gbt.dis`. This summary of the BFP (Best Fitted Paraboloid) parameters was taken directly from output of King's surface-fitting program `rms`, and was requested by R. Norrod for distribution at a 10/24/94 GBT meeting, October 1994.

[Kin94b] Lee King. GBT coordinate systems. Limited-distribution memo, January 1994.

[Kin95] Lee King. GBT optics movements under gravity loads. The relative distances of the turret, the prime focus and the subreflector vertex with respect to the best-fitted paraboloids are calculated for gravity loadings. This Email summary memo was sent to seven people on 5/19/95. Details of the calculations, including source code, are in URL=`ftp://gbtsp20.cv.nrao.edu/pub/optics.defl/`, May 1995.

[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.

[KS66] M. S. Katow and L. W. Schmele. Antenna structures: Evaluation techniques of reflector distortions. *JPL Space Programs Summary*, 37-40(IV):176–184, August 1966. This report is in the NRAO-Charlottesville library under "US/PASA/JPL/". The original version of this program was described in [UB63], and references to other versions are given in this paper.

[MAY76] Y. Mizugutch, M. Akagawa, and H. Yokoi. Offset dual reflector antenna. In *Int. IEEE/AP-S Symp. Digest*, pages 2–4. IEEE, 1976. This is the original paper for the "Mizugutch condition", in which cross-polarization is minimized in an off-axis dual antenna if the axis of the secondary is tilted by a small angle (5.57° in the GBT).

[Nor95] Roger D. Norrod. GBT surface accuracy. GBT Memo 119, National Radio Astronomy Observatory, January 1995. This discussion of GBT surface accuracy uses results from [Kin94a] and [Sri94].

[Sch92] Fred Schwab. FITREFL: A computer program to calculate Best-Fitting Paraboloids and Ellipsoids by ordinary least squares or by orthogonal distance regression (version 0). This memo documents Schwab's implementation based on the ODRPACK library from NIST [BBRS92], January 1992.

[Sri92] S. Srikanth. Correcting for gravity induced deformations. GBT Memo 78, National Radio Astronomy Observatory, May 1992. Deformations extracted from NASTRAN models. *Note:* this memo has been (mostly) superceded by [Sri94].

[Sri94] S. Srikanth. Gain reduction due to gravity-induced deflections of the GBT tipping structure (model 95, version B) and its compensation. GBT Memo 115, National Radio Astronomy Observatory, September 1994. See Table 1 on page 4, "Gravity-induced deformations". See the addendum [Sri95] to this memo; an earlier version of this memo is [Sri92].).

[Sri95] S. Srikanth. Addendum to GBT memo No. 115. Technical Report 121, National Radio Astronomy Observatory, September 1995.

[Tho89] A. R. Thompson. Comments on 'active surface adjustment to nominal vs. nearby paraboloid'. GBT Memo 21, National Radio Astronomy Observatory, October 1989.

[UB63] S. Utku and S. M. Barondess. Computation of weighted root-mean-square of path length changes caused by the deformations and imperfections of rotational paraboloidal antennas. Technical Memorandum 33-118, Jet Propulsion Laboratory, Pasadena, CA, March 1963.

[vH93] Sebastian von Hoerner. Shift of prime focus location by small surface adjustments. GBT Memo 100, National Radio Astronomy Observatory, January 1993.

[WK95] Don Wells and Lee King. The GBT Tipping-Structure Model in C. GBT Memo 124, National Radio Astronomy Observatory, March 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. See URL=`http://info.gb.nrao.edu/GBT/memos/memo124.ps`.

# A The BFP C-function

This function is in file `gbt_paraboloid.c`:

```
/* GBT Commanded_Paraboloid as function of Elevation
   Computed Tue May 30 18:18:38 EDT 1995
   from least-squares fits to Model 95b
   Don Wells <dwells@nrao.edu>, Dec94-May95 */

char BFP_SELECT[]="tipping0 rms";

#include "math.h"
#include "structural_model.h"
```

```
#include "paraboloid.h"

void gbt_paraboloid (double el,    /* Elevation       (degrees) */
                     double vxyz[], /* Vertex_Offset   (inches)  */
                     double *rx,    /* Tilt_about_X    (radians) */
                     double *fl,    /* Focal_Length    (inches)  */
                     double dxyz[], /* Focus_Offset    (inches)  */
                     double xyz0[]) /* Rigging_Focus   (inches)  */
{
    const double fl0 = (60.0 * 39.37); /* fl=60_m at rigging angle */

    double *vy, *vz;
    double x = (el - 45.0) / 45.0; /* Elevation scaled to [-1,+1]
                                      for Chebyshev polynomial    */
    double t0 =    1.0;
    double t1 =      x;
    double t2 =  2.0*x*x        -1.0;
    double t3 =  4.0*x*x*x      -3.0*x;
    double t4 =  8.0*x*x*x*x    -8.0*x*x   +1.0;
    double t5 = 16.0*x*x*x*x*x -20.0*x*x*x +5.0*x;

    vxyz[0] = 0.0;
    vy = &vxyz[1];
    vz = &vxyz[2];

   *vy = ( 1.5767) * t0
       + (-2.8064) * t1
       + ( 1.6245) * t2
       + ( 0.0751) * t3
       + (-0.0213) * t4
       + (-0.0006) * t5;
     /* ( 0.0003)   sigma-of-fit for Chebyshev polynomial */

   *vz = ( 0.01778) * t0
       + (-1.18147) * t1
       + ( 0.04579) * t2
       + ( 0.03169) * t3
       + (-0.00054) * t4
       + (-0.00022) * t5;
     /* ( 0.00026)   sigma-of-fit for Chebyshev polynomial */

   *rx = ( 0.0003644) * t0
       + (-0.0000827) * t1
       + ( 0.0003606) * t2
       + ( 0.0000034) * t3
       + (-0.0000048) * t4
       + (-0.0000007) * t5;
     /* ( 0.0000024)   sigma-of-fit for Chebyshev polynomial */

   *fl = (2362.3580) * t0
       + (   0.1293) * t1
       + (   0.1531) * t2
       + (  -0.0034) * t3
       + (  -0.0021) * t4
       + (   0.0002) * t5;
     /* (   0.0003)   sigma-of-fit for Chebyshev polynomial */
```

```
    xyz0[0] =     0.0;
    xyz0[1] = -2159.02;       /* offset vertex wrt El axis  */
    xyz0[2] =   196.85        /* offset vertex wrt El axis  */
              +1900.0         /* offset El axis wrt alidade */
              + fl0;          /* plus nominal focal length  */

    dxyz[0] = 0.0;
    dxyz[1] = *vy - *fl * sin(*rx);
    dxyz[2] = *vz + *fl * cos(*rx) - fl0;
}
```

# B The Prime Focus focus-tracking function

This function is in file `gbt_pf_focus_track.c`:

```
/* GBT Prime-Focus Focus-Tracking as function of Elevation
   Computed Tue May 30 18:19:35 EDT 1995
   from least-squares fits to Model 95b.
   Results bx,by,bz are in prime focus coordinate system.
   Don Wells <dwells@nrao.edu>, March 1995 */

#include "structural_model.h"
#include "paraboloid.h"

void gbt_pf_focus_track (double  el,     /* Elevation    (degrees) */
                         double *bx,     /* Box_Offset_X (inches)  */
                         double *by,     /* Box_Offset_Y (inches)  */
                         double *bz)     /* Box_Offset_Z (inches)  */
{
    double x = (el - 45.0) / 45.0; /* Elevation scaled to [-1,+1]
                                      for Chebyshev polynomial    */
    double t0 =    1.0;
    double t1 =      x;
    double t2 =  2.0*x*x        -1.0;
    double t3 =  4.0*x*x*x      -3.0*x;
    double t4 =  8.0*x*x*x*x    -8.0*x*x   +1.0;
    double t5 = 16.0*x*x*x*x*x -20.0*x*x*x +5.0*x;

   *bx = (  0.933) * t0
       + (  3.959) * t1
       + (  0.837) * t2
       + ( -0.109) * t3
       + ( -0.011) * t4
       + (  0.002) * t5;
      /* (  0.000)   sigma-of-fit for Chebyshev polynomial */

   *by = ( -1.112) * t0
       + ( -2.726) * t1
       + ( -1.030) * t2
       + (  0.074) * t3
       + (  0.013) * t4
       + ( -0.002) * t5;
      /* (  0.000)   sigma-of-fit for Chebyshev polynomial */

   *bz =    0.000;  /* no Z box-offset for symmetric model */
}
```

# C Function to retrieve data for nodes *wrt* BFP

This function is in file `get_node_wrt_bfp.c`:

```
/* Function get_node_wrt_bfp() is similar to function get_node().  The
   difference is that, rather than delivering a struct node_data in
   the 'elevation' coordinate system (plus 1900 inches in Z), it
   delivers the struct with_respect_to the axis of the best fitting
   paraboloid for the requested elevation. The node is computed in the
   elevation system, and then is transformed to be relative to the BFP
   vertex, and then is rotated by the BFP rotation. The BFP focal
   length is subtracted from the new Z so that the coordinates will be
   relative to the prime focal point of the BFP at the elevation. This
   version of the node_data struct facilitates optical calculations:
   gbt_paraboloid() gives the focal length of the primary paraboloid,
   while wrt_bfp->grid[] plus wrt_bfp->at_elev.delta[] gives the
   position of the Gregorian feedhorn (or prime focus box or
   subreflector 'vertex') relative to the focal point of the primary
   paraboloid. The tilts relative to the BFP axis for these nodes are
   returned in wrt_bfp->at_elev.tilt[]. D.Wells, NRAO-CV, May95. */

#include "math.h"
#include "structural_model.h"
#include "paraboloid.h"

int get_node_wrt_bfp(int i_node,                /* tipping_model[] index */
                     double elev,               /* elevation       (deg) */
                     struct node_data *wrt_bfp) /* results returned here */
{

    const double refl_orig[3] = {0.0, -2159.02, +2096.85}; /*reflector origin*/
    const double fl0 = (60.0 * 39.37);
    int i, j;
    double refl[3], wrt_untilted[3], ctilt, stilt, rotate[3][3], wrt_tilted[3];
    double vxyz[3], rx, fl, bfp_dxyz[3], bfp_xyz[3];
    struct node_data tipping_node;

    gbt_paraboloid (elev, vxyz, &rx, &fl, bfp_dxyz, bfp_xyz);
    if (get_node_data(i_node, elev, &tipping_node)) return(13);
    wrt_bfp->node_id   = tipping_node.node_id;
    wrt_bfp->elevation = tipping_node.elevation;
    for (i = 0; i < 3; i++) {
        refl[i] =
            (  (double)tipping_node.grid[i]
             - refl_orig[i] );
        wrt_untilted[i] =
            (  refl[i]
             + (double)tipping_node.at_elev.delta[i]
             - vxyz[i] );
    }
    ctilt = cos(-rx); stilt = sin(-rx);
    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, wrt_tilted[i] = 0.0; j < 3; j++)
        wrt_tilted[i] += wrt_untilted[j] * rotate[i][j];
    for (i = 0; i < 3; i++) {
```

```
        wrt_bfp->grid[i] = tipping_node.grid[i];
        wrt_bfp->at_elev.delta[i] = wrt_tilted[i] - refl[i];
        wrt_bfp->at_elev.tilt[i]  = tipping_node.at_elev.tilt[i];
    }
    wrt_bfp->grid[2] -= fl0;          /* new origin is at prime_focal_point */
    wrt_bfp->at_elev.delta[2] -= (fl - fl0);
    wrt_bfp->at_elev.tilt[0]  -= rx;          /* new coor system is tilted */
    return(0);
}
```

# D The "include" file

The text reproduced below is the file `paraboloid.h`, which contains the ANSI-C prototypes for the BFP and PF-focus-tracking functions:

```
/* Include for the best-fitting-paraboloid functions
   D.Wells, NRAO-CV, Dec94-May95 */

void gbt_paraboloid (double el,    /* Elevation       (degrees) */
                     double vxyz[], /* Vertex_Offset   (inches)  */
                     double *rx,    /* Tilt_about_X    (radians) */
                     double *fl,    /* Focal_Length    (inches)  */
                     double dxyz[], /* Focus_Offset    (inches)  */
                     double xyz0[]) /* Rigging_Focus   (inches)  */
;

void gbt_pf_focus_track (double  el,     /* Elevation    (degrees) */
                         double *bx,     /* Box_Offset_X (inches)  */
                         double *by,     /* Box_Offset_Y (inches)  */
                         double *bz)     /* Box_Offset_Z (inches)  */
;

/* The 'node_data' reference below means that structural_model.h
   should be included before this include: */

int get_node_wrt_bfp(int i,                     /* tipping_model[] index */
                     double elev,               /* elevation       (deg) */
                     struct node_data *wrt_bfp) /* results returned here */
;
```

# E The Gaussfit "program" for fitting conic sections

In this section, we reproduce the source code file `model.gf`, which GaussFit [JFMM88] uses to fit conic sections to data. The equations of condition for the regression are the argument of the `export()` function call in the program. The equation is based on the common vertex equation of the conic sections [GKHK75, p.315], for the case of the axis of the conic along the Z-axis:

$$x^2 + y^2 = 2pz - (1 - \varepsilon^2)z^2, \tag{3}$$

where $\varepsilon$ is the numerical eccentricity and $p$ is the "semiparameter"[6] of the conic. For a paraboloid, we have $\varepsilon = 1$ and $p = 2F$, and so Eq.(3) reduces to

$$z = \frac{x^2 + y^2}{4F}, \tag{4}$$

where $F$ is the focal length (60 meters for the GBT primary mirror). The substitution $2p = 4F$ has been made in the `model.gf` shown below, as a convenience for the BFP analysis.

The numerical eccentricity variable is still present to enable the code to fit ellipsoids.[7] For an ellipsoid, we have $p = b^2/a$, where $a$ is the semi-major axis and $b$ is the semi-minor axis, so that the `model.gf` code shown here is able to solve for an "$F$" which is $b^2/2a$. We have $a^2 - e^2 = b^2$ and $\varepsilon = e/a$; it follows that `model.gf` will solve for an ellipsoid size parameter

$$\frac{b^2}{2a} = e(\frac{1}{2\varepsilon} - \frac{\varepsilon}{2}), \tag{5}$$

where $e$ is the "linear eccentricity" (one-half the desired inter-focal distance) and $\varepsilon$ is the "numerical eccentricity". The nominal values for the GBT are $e = 5.5$ meters (11 meters between the foci) and $\varepsilon = 0.528$). If the declaration "`constant eps;`" is changed to "`parameter eps;`", `model.gf` will be able to fit for the numerical eccentricity as well as the ellipsoid size parameter.

```
/* GaussFit program to fit paraboloids to the distorted GBT surface */
/*      Don Wells, NRAO-CV, 08-December-1994,3/16/95 */

observation     x;      /* + = away from plane of symmetry */
observation     y;      /* + = away from feed arm */
observation     z;      /* + = up, height of surface */

parameter       vy;     /* vertex position */
parameter       vz;
parameter       rx;     /* paraboloid tilt about x-axis (radians) */
parameter       fl;     /* focal length */
constant        eps;    /* eccentricity */

main()
{
        variable x0, y0, z0, srx, crx, xp, yp, zp, xpp, ypp, zpp;

        x0    = 0.0;
        y0    = vy;
        z0    = vz;
        srx  = sin(rx);
        crx  = cos(rx);

        /* COMPUTE PREDICTED VALUES */
        while (import()) {
                xp = x - x0;
                yp = y - y0;
                zp = z - z0;
                /* positive rx is right-hand-rule about +X: */
                xpp = xp ;
                ypp =  +crx * yp  +srx * zp;
                zpp =  -srx * yp  +crx * zp;
```

[6] "The parameter $2p$ of a parabola $y^2 = 2px$ in the vertex position is defined as the length of the chord of the parabola perpendicular to the axis through the focus; it measures the width, so to speak, of the parabola at the focus. This definition can be carried over to the other conics: *The parameter of a conic is defined as the length of the chord perpendicular to the principal axis through a focus.*" [GKHK75, p.314]

[7] The original version [Sch92] of `fitrefl` is also able to fit ellipsoids.

```
                /* Common vertex equation of the conics (VNR, p.315): */
                export( (xpp^2 + ypp^2)
                        - (((4.0 * fl) - (1.0 - eps^2) * zpp) * zpp) );
        }
}
```

# F The surface-data generator

This program is file `get_surface.c`:

```
/* get_surface.c --- compute actuator coors as fn of Elevation
   D.Wells, NRAO-CV, Dec94-Mar95 */
#include <stdlib.h>
#include <math.h>
#include "structural_model.h"
#define MAX_SURF 2300
main() {
    int
        num_nodes, i, j, pass, node_count, full_flag, id, aid, num_surf,
        first = 1, diff_count, surf_index[MAX_SURF], surf_id[MAX_SURF];
    const double
        surface_rig_angle = 44.0, /* degrees */
        fl = (60.0 * 39.37), /* 60m in inches */
        translate_to_vertex[] = {0.0, 2159.020, -2096.85},
        tolerance = 4.0; /* inches */
    double
        elev, /* degrees */ xyz[3], rad2, zp, diff;
    char line[100], name[20];
    struct node_data surf_node;
    FILE *f2;

    /* make a list of the surface (actuator) nodes: */
    if ((num_nodes = get_index(0)) == 0) exit(EXIT_FAILURE);
    for (i = 1, num_surf = 0; i <= num_nodes; i++) {
        if (get_node_data(i, elev, &surf_node)) exit(EXIT_FAILURE);
        id = surf_node.node_id;
        aid = id > 0 ? id : -id;
        if ((aid >=  700001) && (aid <=  768012)) {
            surf_index[num_surf] = i;
            surf_id[num_surf] = id;
            if ((num_surf++) > MAX_SURF) exit(EXIT_FAILURE);
        }
    }
    f2 = fopen("surf_diffs.out", "w"); /* for unusual actuator Z-coors */
    if (gets(line) == NULL) { printf("EOF?!?\n"); exit(EXIT_FAILURE); }
    sscanf(line, "%d", &full_flag);
    while (gets(line) != NULL) {
        sscanf(line, "%lf", &elev); if (elev > 90.0) break;
        strcpy(name, "half"); if (full_flag) strcpy(name, "full");
        printf ("tipping%1d %s tipping_model at el_angle wrt surface_rig_angle\n",
                full_flag, name);
        for (pass = 1, node_count = 0, diff_count = 0;
             pass <= 2; pass++) {
            for (i = 0; i < num_surf; i++) {
                if (get_node_data(surf_index[i], elev, &surf_node)) exit(EXIT_FAILURE);
```

```
                if ((id = surf_node.node_id) != surf_id[i]) exit(EXIT_FAILURE);
                if (full_flag || (!full_flag && (id > 0))) {
                    if (pass == 1) {
                        node_count++; /* how many in dataset? */
                    } else { /* pass == 2: */
                        for (j = 0; j < 3; j++) xyz[j] = (double)surf_node.grid[j]
                                                    + translate_to_vertex[j];
                        rad2 = (xyz[0]*xyz[0] + xyz[1]*xyz[1]);
                        zp = rad2 / (4.0 * fl);
                        /* check difference btw grid coors and paraboloid: */
                        if (first && fabs(diff = xyz[2] - zp) > tolerance) {
                            diff_count++;
                            fprintf(f2,
                                    "n=%3d: id=%7d, r=%6.1lf, d=%5.2lf (=%7.2lf-%7.2lf)\n",
                                    diff_count, id, sqrt(rad2), diff, zp, xyz[2]);
                        }
                        /* The GBT surface will be adjusted to 60m paraboloid
                           at surface_rig_angle, so we overwrite xyz[2]: */
                        xyz[2] = zp;
                        printf ("%8d%8.2lf%8.2lf%8.2lf%8.2lf",
                                id, 1.0, xyz[0], xyz[1], xyz[2]);
                        printf ("%8.5lf%8.5lf%8.5lf\n", surf_node.at_elev.delta[0],
                                surf_node.at_elev.delta[1],
                                surf_node.at_elev.delta[2]);
                    }

                }
            }
            if (pass == 1) printf ("%6d%8.2f\n", node_count, elev);
        }
        first = 0;
    }
    fclose(f2); exit(EXIT_SUCCESS);
}
```