GBT MEMO 183

GBT Gregorian Focus Tracking in C

Don Wells*

June 19, 1998

Abstract

The GBT Gregorian subreflector, an off-axis portion of an ellipsoid, plus the feedroom with the feedhorn move relative to the prime focal point of the main paraboloidal mirror. The subreflector must be maneuvered relative to the prime focal point and the feedhorn to maintain nearly stigmatic imaging (maximum gain, minimum sidelobes). An algorithm is described which computes the required actuator motions as a function of elevation. Raytracing analysis shows that GBT wavefronts produced by this optical prescription exhibit *no* focus error, spherical abberation or coma; their only abberation is astigmatism, with amplitude 0.4 mm for $E = 0°$ and $E = 90°$.[1]

Contents

1 Introduction

S. Srikanth showed [Sri94] several years ago that the loss of gain at the Gregorian focus due to gravity-induced deflections of the GBT tipping structure can be compensated quite effectively by translation of the

*`mailto:dwells@nrao.edu`

[1] The source code described in this memo is available at `ftp://fits.cv.nrao.edu/pub/gbt_dwells_src.tar.gz`.

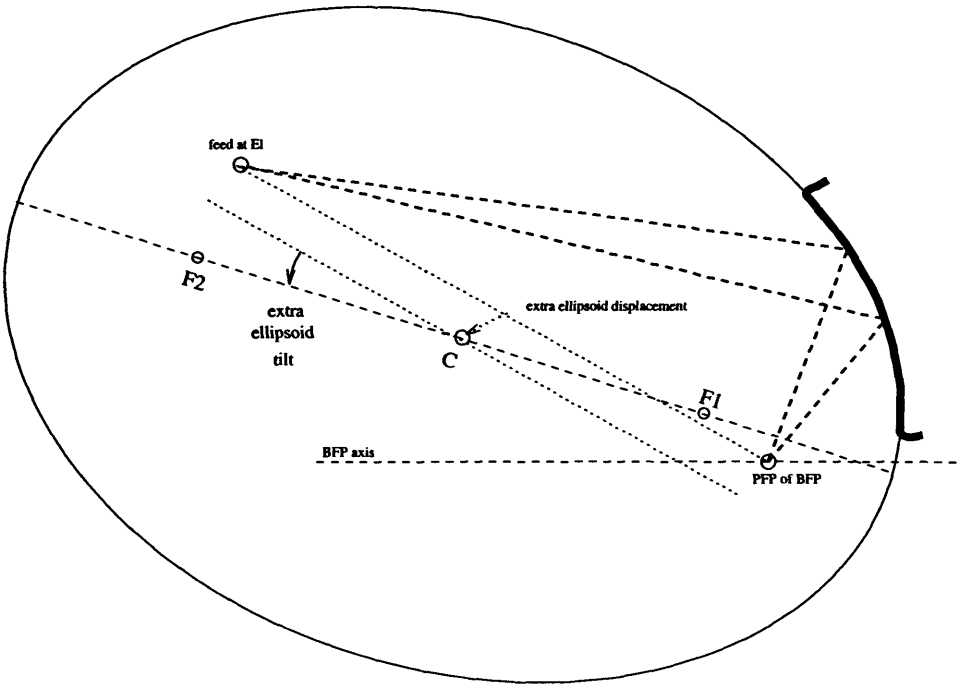

Figure 1: Extra Tilt and Translation of the Subreflector

subreflector: "The residual path length error after fitting a plane is 0.49 mm, giving a phase efficiciency of 84.54% at 20 Ghz." Srikanth found his compensation solution by a search procedure. In this report I show that the optimum translation can be computed by a geometric procedure which I have coded in C, and that the wavefront residuals are slightly improved if the subreflector is tilted by a small amount.

2 The geometry of GBT focus tracking

Before discussing the full geometry problem, we will define the terms "extra ellipsoid displacement" and "extra ellipsoid tilt". Figure 1 shows a schematic cross-section of the subreflector and the prime and Gregorian foci, in the plane of symmetry of the GBT, with the optical axis horizontal (as though the GBT were at $E = 0°$). The cross-section of the ellipsoid is the ellipse in the figure, with the off-axis segment occupied by the GBT subreflector in bold line. The point labelled "PFP of BFP" is the point of peak gain of the paraboloidal primary mirror, *i.e.*, the "best focus". The point labelled "feed at El" is the phase center of the Gregorian feedhorn. Two ray paths are shown as moderately-bold dashed lines from the PFP to the subreflector to the feed. The point labelled "C" is the center of the ellipsoid, which has two foci, "F1" and "F2", on its major axis; these foci are 11 meters apart. If the elevation angle of the GBT is its "rigging angle,"[2] the "feed" and "PFP" points will also be 11 meters apart—they will be adjusted to have this relationship during the alignment of the upper feedarm, near the end of the construction of the GBT. At the rigging angle the ellipsoid can be maneuvered until F1 coincides with the PFP and F2 coincides with the feed, and the design parameters of the GBT will then produce perfect imaging on axis. In this situation the center C will coincide with the bisection point of the PFP-to-feed line.

At elevations other than the rigging angle, the PFP and feed will be separated by distances greater or less than 11 meters, and F1 and F2 cannot be made to coincide with the PFP and feed. For these cases we must displace the center of the subreflector away from the bisection point of the PFP-to-feed line. In Figure 1 this

[2] In this memo the rigging angle is taken to be 44°, the value used in the GBT project during 1990-97. Recently the decision has been made (Lee King, private communication) to change to 50.8°, approximately the midpoint of the GBT's 5° $\mapsto$ 95° elevation range.

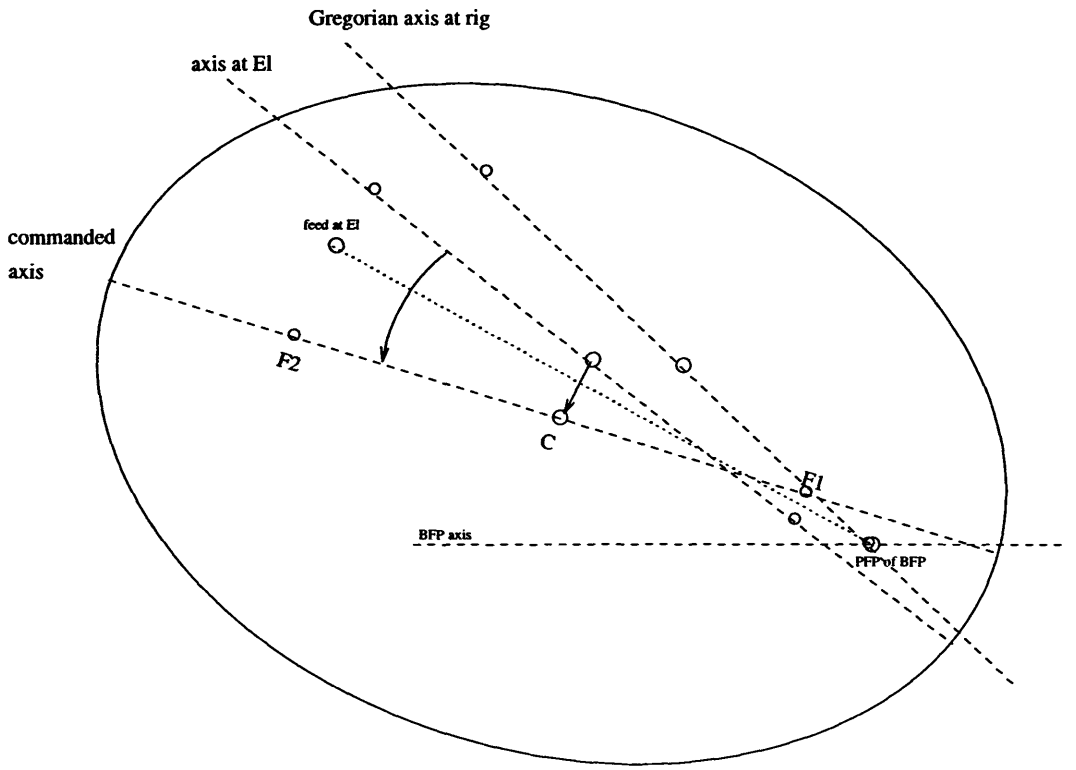

Figure 2: Subreflector Focus Tracking Actuator Motions

motion is labelled as "extra ellipsoid displacement"; it is relative to the bisection point. The RMS phase error analysis in [Wel98b] shows that we can get the best imaging performance by tilting the subreflector slightly; this motion is labelled "extra ellipsoid tilt" in the figure, and it is relative to the line connecting the PFP and feed. These motions will produce the closest approximation to stigmatic imaging as the feedhorn moves relative to the PFP due to the changing gravity vector.

The terms "BFP axis" and "PFP of BFP" in the figure refer to function **`smGetNodeWrtBfp()`** [WK95a, Section 5.3 and Appendix C], which invokes function **`smGetNodeData()`** [WK95b] to obtain "grid" coordinates, displacements and tilts of structural model nodes and then uses results of function **`smParaboloid()`** [WK95a] to transform them to the "BFP" [Best-Fitting Paraboloid] coordinate system. The origin of this coordinate system is at the PFP [Prime Focal Point] and its Z-axis is coincident with the axis of the main mirror; the focal point origin and the axis move *wrt* the tipping structure coordinate system as the main mirror distorts homologously from paraboloid to paraboloid. Use of BFP coordinates simplifies Gregorian optics calculations.

The situation of the GBT has an additional level of complexity: the actuator attachment points of the subreflector are also translating and tilting due to the changing gravity vector! This situation is illustrated in Figure 2. The line labelled "Gregorian axis at rig" is the rigging angle geometry, with small circles to mark the position of the feedhorn and the bisection point. The point labelled "feed at El" and the dotted line connecting it to the PFP correspond to the "feed at El" dotted line in Figure 1. The ellipse, dashed-line major axis and points labelled F1, F2 and C correspond to the same entities in the previous figure; these items represent the desired position of the subreflector. The new element in Figure 2 is the line labelled "axis at El" and its three points; these items represent the position to which the subreflector moves for elevation "El" due to the changing gravity vector after it has been adjusted to have the "Gregorian axis at rig" position at the rigging angle. The translation vector and tilt angle in this figure illustrate the focus tracking motions which we must calculate; these motions are the motions illustrated in Figure 1 corrected

for the gravitational displacement of the subreflector (see Step 13 [p.10], Step 14 [p.10] and Step 15 [p.11] below).

3 Function srFocusTracking()

The interface to the function is:

```
int srFocusTracking (                    /* returns != 0 on error              */
                     double elev,        /* elevation                  [deg] */
                     double temp,        /* temperature              [deg_C] */
                     double F1_delta[3], /* first-focus offset           [m] */
                     double F2_delta[3], /* second-focus offset          [m] */
                     enum srFT_MODE mode,/* srFT_DEFAULT|srFT_NOEXTRA        */
                     double sub_trans[], /* XYZ subrefl backup         [in] */
                     double sub_tilts[], /* xyz subrefl tilts          [deg] */
                     struct node_data
                            *greg_feed,  /* feed coords/tilts wrt BFP        */
                     double *dS12,       /* PFP->feed - 11m              [m] */
                     double *extra_tilt, /* extra tilt subrefl          [mr] */
                     double S[],         /* subreflector optical vertex [in] */
                     double E[])         /* vertex tilt angles         [rad] */
```

The strategy of this algorithm is to first calculate the position and orientation of the ellipsoid vertex (`S[]` and `E[]`, Step 12 [p.9]); this facilitates checking the first part of the calculations by ray tracing. The algorithm then proceeds to compute the "center" position of the subreflector starting from `S[]`, `E[]` and `S_to_center[]`. Finally, the tilts to command are computed by compensating `E[]` for `subr_center.at_elev.tilts[]`.

The results which are tabulated and plotted below present the GBT as a symmetric structure, with the computed subreflector tilt angles all in the plane of symmetry. The actual GBT will be slightly asymmetric, due to the elevator which is attached to the right-hand half of the feedarm. This elevator asymmetry will cause the feedarm to move sideways by a small amount (± 0.83 inch), and ultimately (when `srFocusTracking()` calls an assymetric structural model) the subreflector tilts will be three angles, not just one. The present algorithm should be nearly sufficient to do this; in particular, the iterative algorithm used at Step 15 [p.11] below will invert the final rotation matrix into the necessary three angles. The author is grateful to Fred Schwab for developing the Mathematica program which computed the code of this iterative inversion algorithm symbolically.

The function argument `temp` is only partially implemented in the current version of the algorithm (see Step 1 [p.6]). Eventually the `smGetNodeData()` [WK95b] and `smParaboloid()` [WK95a] functions will return results which will scale with the thermal expansion of the materials. Also, eventually the `srDisplacementToLength()` function [Wel98a] will be modified to include calculation of the thermal expansion of the actuators and the small differential expansion of the magnetostrictive actuator encoders.

The algorithm makes extensive use of 3-vectors and 3×3 rotation matricies. The statement `VECTOR_ADD(TV, V, TA, A, TB, B)` computes vector `V` as the sum of vectors `A` and `B`; arguments `TV`, `TA` and `TB` specify the "types" of the vectors. The vector operator macro package (`mathVectorMatrix.h`) defines two types of vectors and matrices: (1) conventional indexed C arrays, with type codes `VC` and `MC`, and (2) sets of scalar elements with names like `myvector_0`, `myvector_1`, .., `mymatrix_02`, `mymatrix_12`, etc., using type codes `VS` and `MS`. Variables of the second type are declared by the `VECTOR_VS()` and `MATRIX_MS()` statements below. In this algorithm type (1) is used for function arguments and type (2) is used for internal variables. The hope is that scalar optimization will improve speed for types `VS` and `MS`.

```
/* srFocusTracking.c -- GBT Gregorian focus-tracking function
```

```
   D.Wells, NRAO-CV
   1995-09-??: original development
   1997-04-29: massive renaming of functions and includes
   1997-05-01: test-mode argument removed to simplify function
   1997-05-30: more renaming
   1997-07-23: change vectors from VC to VS notations
   1997-07-24: test macros MATRIX_TRANSPOSE and GET_SR_TILTS
   1997-11-21: minor changes for Vector-Matrix macro calls
   1998-04-14: debug print macros renamed
   1998-05-14: units indicated in comments systematically
   1998-05-20: sub_trans[] bug fix
   1998-06-11: removed delta_tilt, added additional 3-D rotation to tilts
   1998-06-18: flip Y_S
   */
```

[GNU GPL copyright notice omitted]

```
#include <stdlib.h>
#define SR_DEBUG 1
#include "srInclude.h"

int srFocusTracking (                    /* returns != 0 on error            */
                     double elev,        /* elevation                 [deg] */
                     double temp,        /* temperature              [deg_C] */
                     double F1_delta[3], /* first-focus offset           [m] */
                     double F2_delta[3], /* second-focus offset          [m] */
                     enum srFT_MODE mode, /* srFT_DEFAULT|srFT_NOEXTRA       */
                     double sub_trans[], /* XYZ subrefl backup         [in] */
                     double sub_tilts[], /* xyz subrefl tilts         [deg] */
                     struct node_data
                            *greg_feed,  /* feed coords/tilts wrt BFP       */
                     double *dS12,       /* PFP->feed - 11m             [m] */
                     double *extra_tilt, /* extra tilt subrefl         [mr] */
                     double S[],         /* subreflector optical vertex [in] */
                     double E[])         /* vertex tilt angles        [rad] */
{
    const int
      greg_feed_id = 40700, subr_center_id = 50005;
    int
      greg_feed_index,      subr_center_index,
      status;
    struct node_data        subr_center;
    double
      thermal_factor,       /* 1+eps                        [dimensionless] */
      FocalLength,          /* focal length of _aluminum_ ellipsoid      [m] */
      SemimajorAxis,                    /* _aluminum_ ellipsoid      [m] */
      rx,                   /* tilt of BFP axis                       [rad] */
      S12,                  /* distance from PFP to Greg feedhorn        [in] */
      dxyc[3], vxyz[3], fl, dxyz[3], xyz0[3], f2angle;
    double                  /* temp vars for Schwab inversion algorithm: */
      calpha, salpha, talpha, csalpha, c2alpha, s2alpha,
      cthx, sthx, cthy, sthy, cthz, sthz;
    VECTOR_VS(S_to_center);         VECTOR_VS(center_adjust);
    VECTOR_VS(feed_at_elev);
    VECTOR_VS(greg_feed_tilt);      VECTOR_VS(greg_feed_delta);
    VECTOR_VS(pfp_at_elev);         VECTOR_VS(pfp_delta);
    VECTOR_VS(pfp_to_feed);         VECTOR_VS(pfp_feed_unit);
```

```
VECTOR_VS(vertical);              VECTOR_VS(sideways);
VECTOR_VS(downward);              VECTOR_VS(C);
VECTOR_VS(dxyc_permuted);         VECTOR_VS(dxyc_rotated);
VECTOR_VS(feed_pfp_unit);         VECTOR_VS(ellipsoid_axis_unit);
MATRIX_MS(pfp_rotate);            MATRIX_MS(greg_feed_rotate);
MATRIX_MS(EllipsoidRotation);     MATRIX_MS(EllipsoidRotInv);
MATRIX_MS(ActuatorRotation);
MATRIX_MS(Rotator);               VECTOR_VS(S_center_rotated);
VECTOR_VS(sub_diff);
VECTOR_VS(center_target);         VECTOR_VS(center_at_elev);
MATRIX_MS(VertexTilt);            MATRIX_MS(VertexTiltInv);
MVM_PRIVATE_VARIABLES;  /* (used only by mathVectorMatrix.h macros) */
```

Step 1 Get ellipsoid optical parameters

The ellipsoid subreflector backup structure is made of aluminum, unlike the paraboloid reflector backup structure and the other trusses of the GBT. Therefore its focal length scales differently with temperature from the scaling of the focal length of the paraboloid and the feedarm. The subreflector is attached at its center, and therefore scales about this point. The vector from the vertex to the center `S_to_center[]` given below has been computed by solving the equations $y = -(z + 2e)\tan\alpha$ (axis of feedhorn) and $(z + e)^2/a^2 + y^2/b^2 = 1$ (ellipse) for the vector from F_1 to the intercept point at the center of the ellipsoid and subtracting the vector from F_1 to the vertex; the result has been adjusted by ≈ 1.5 mm to make it agree with the grid coordinates used in Model 95b.

```
thermal_factor = (1. + ALUMINUM * (temp - RIGGING_TEMP));
FocalLength = GBT_GREG_FL * thermal_factor;                        /* [m] */
/* a=e/eps & e=f/2: */
SemimajorAxis = ((0.5 * FocalLength) / GBT_EPS) * thermal_factor; /* [m] */
/* S_to_center, as computed: */
VECTOR_INIT(VS, S_to_center,   0.000, -182.699, -60.957);          /* [in] */
/* Model 95b adjustment: */
VECTOR_INIT(VS, center_adjust, 0.000,   +0.031,  -0.056);          /* [in] */
VECTOR_ADD(VS, S_to_center, VS, S_to_center, VS, center_adjust);
VECTOR_SCALE(VS, S_to_center, thermal_factor, VS, S_to_center);    /* [in] */
```

Step 2 Get the positions of Gregorian feed and subreflector center

We get the displaced nodes expressed in "BFP" coordinates. The BFP coordinate system has the axis of the homologously-deforming paraboloid as its Z axis and the prime focal point as its origin. This coordinate system, which is rotated relative to the coordinate system of the tipping structure model, is convenient for optical calculations. The function calls return the "grid" (undeflected) positions expressed in the BFP coordinate system and the gravitational deflections (both translation and tilt), also expressed in the BFP system. The deflected positions of the nodes are the sums of the grid positions and translation components of gravity deflection. Note that, because the origin of the BFP system is at the prime focal point [PFP], the resulting node positions are the vectors from the prime focal point [PFP] to the feedhorn and center if the focal point offsets are zero.

```
if ((status = smGetNodeWrtBfp((greg_feed_index = smGetIndex(greg_feed_id)),
                              elev, greg_feed)) != 0) return(status);
VECTOR_ADD(VS, feed_at_elev,
           VC, (double)greg_feed->grid,
           VC, (double)greg_feed->at_elev.delta);                  /* [in] */
if ((status= smGetNodeWrtBfp((subr_center_index=smGetIndex(subr_center_id)),
```

```
                              elev, &subr_center)) != 0) return(status);
VECTOR_ADD(VS, center_at_elev,
           VC, (double)subr_center.grid,
           VC, (double)subr_center.at_elev.delta);                /* [in] */
```

Step 3 **Get the BFP axis-tilt**

The BFP axis-tilt `rx` [WK95a] is needed in order to rotate the focal point offset vector `F2_delta[]`.

```
smParaboloid(elev, vxyz, &rx, &fl, dxyz, xyz0);   /* [d,in,rad,in,in,in] */
```

Step 4 **Add `F1_delta[]` offset to PFP position**

The vector `F1_delta[]`, which is in prime focal plane coordinates [m], is rotated ≈45° to BFP coordinates and is added to the PFP position vector (which is identically zero because it is the origin of the BFP coordinate system).

```
ANGLES_2_MATRIX(MS, pfp_rotate,
                X, -GBT_PF_AXIS_TLT*DTR,
                NOP,NOP, NOP,NOP, NOP,NOP, NOP, NOP)
MATRIX_VECTOR_MULT(VS, pfp_delta, MS, pfp_rotate, VC, F1_delta); /* [in] */
VECTOR_SCALE(VS, pfp_delta, M2I, VS, pfp_delta);                  /* [m] */
VECTOR_FILL(VS, pfp_at_elev, 0.);                              /* at origin */
VECTOR_ADD(VS, pfp_at_elev, VS, pfp_at_elev, VS, pfp_delta);      /* [in] */
```

Step 5 **Add `F2_delta[]` offset to feedhorn position**

The vector `F2_delta[]`, which is in Gregorian feedhorn coordinates [m], is rotated to BFP coordinates and is then added to the feedhorn position vector [in]. The two angle symbols used here are defined in the file `gbtOpticalConstants.h` which is included by `srInclude.h`.

```
f2angle = ((-GBT_ALPHA_ANGLE + GBT_BETA_ANGLE) * DTR + rx);
ANGLES_2_MATRIX(MS, greg_feed_rotate,
                X, f2angle,
                NOP,NOP, NOP,NOP, NOP,NOP, NOP, NOP)
MATRIX_VECTOR_MULT(VS, greg_feed_delta, MS, greg_feed_rotate, VC, F2_delta);
VECTOR_SCALE(VS, greg_feed_delta, M2I, VS, greg_feed_delta);      /* [in] */
VECTOR_ADD(VS, feed_at_elev,
           VS, feed_at_elev, VS, greg_feed_delta);                /* [in] */
```

Step 6 **Get `S12` and `dS12`**

`S12` is the distance [in] from the PFP (origin) plus the first offset to the Gregorian feed plus the second offset. ΔS_{12}) is the change of separation between the PFP and the Greg feed wrt E=`RIGGING_ANGLE`.

```
VECTOR_SUB(VS, pfp_to_feed, VS, feed_at_elev, VS, pfp_at_elev);  /* [in] */
S12 = VECTOR_LENGTH(VS, pfp_to_feed);                            /* [in] */
*dS12 = (S12 * I2M) - FocalLength;                                /* [m] */
```

Step 7 **Get orientation of the GregFeed↦PFP line**

This is a unit vector pointing from the feedhorn to the PFP. We also obtain the reciprocal unit vector.

```
VECTOR_SCALE(VS, pfp_feed_unit, (1./S12), VS, pfp_to_feed);
VECTOR_SCALE(VS, feed_pfp_unit, -1.,      VS, pfp_feed_unit);
```

Step 8 **Compute the optimum tilt of the ellipsoid**

Tilting the ellipsoid by the optimum tilt will minimize the spherical abberation and coma (see [Wel98b, Section 6]). The `srFT_NOEXTRA` option is provided to test this minimization empirically.

```
switch (mode) {
case (srFT_DEFAULT):
  *extra_tilt = srEllipsoidBestRms(*dS12 * E3); break;          /* [mr] */
case (srFT_NOEXTRA):
  *extra_tilt = 0.0; break;
default:
  return(EXIT_FAILURE);
}
```

Step 9 **Get the center-point-offset dxyc[]**

We need this quantity [in] to produce stigmatic imaging from the PFP to the Feed:

```
srEllipsoid((*dS12 * E3), *extra_tilt, dxyc);            /* [mm, mr, mm] */
VECTOR_SCALE(VC, dxyc, (M2I / E3), VC, dxyc);                   /* [in] */
```

Step 10 **Get new axes of ellipsoid system**

We can compute orthogonal unit vectors for the axes of the desired ellipsoid orientation by taking cross-products of various vectors. We generate a vertical vector and compute the cross-product of it with the `pfp_to_feed` vector to get a "sideways" vector. The cross-product of `sideways` with the `pfp_to_feed` vector produces a vector orthogonal to both, which we call "downward" (actually, it points sideways when the GBT points to the zenith!) These vectors will be used below to define the rotation matrix for orienting the subreflector. The choice of `downward` used at this step implies that the current algorithm cannot "roll" the subreflector whenever the Y_s rotation is non-zero; such a roll might potentially be used to minimize spillover changes without affecting the stigmatism of the imaging.

```
VECTOR_INIT(VS,  vertical, 0., 0., 1.);
VECTOR_CROSS(VS, sideways, VS, feed_pfp_unit, VS, vertical);
VECTOR_UNIT(VS,  sideways, VS, sideways);
VECTOR_CROSS(VS, downward, VS, feed_pfp_unit, VS, sideways);
VECTOR_UNIT(VS,  downward, VS, downward);
```

Step 11 **Compute the optimum center point**

We want the vector sum of the bisector point of the Feed↦PFP line plus the optimum offset for the ellipsoid center (see Figure 3). The point `C` is the bisector of the line. Vector `dxyc[]` is multiplied by the three unit vectors computed above, and the result is subtracted from `C`.

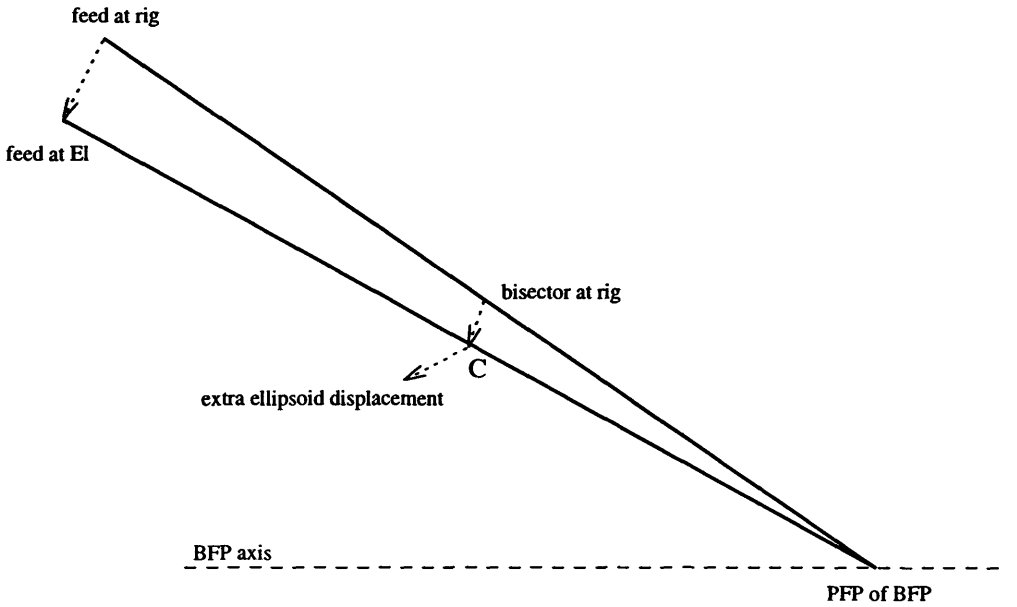

Figure 3: Center-Offset of the Subreflector [Step 11]

```
VECTOR_SCALE(VS, C, 0.5, VS, pfp_to_feed);                    /* [in] */
VECTOR_ADD(VS, C, VS, C, VS, pfp_at_elev);                    /* [in] */
/* X-->+Z, Y-->-Y, Z-->+X: */
VECTOR_PERMUTE(VS, dxyc_permuted, 2, 1, 0, 1., -1., 1., VC, dxyc);
VECTOR_INIT(VS,   dxyc_rotated, 0., 0., 0.);
VECTOR_EXTEND(VS, dxyc_rotated, VS, sideways,     VS(dxyc_permuted,0));
VECTOR_EXTEND(VS, dxyc_rotated, VS, downward,     VS(dxyc_permuted,1));
VECTOR_EXTEND(VS, dxyc_rotated, VS,feed_pfp_unit, VS(dxyc_permuted,2));
VECTOR_SUB(VS, C, VS, C, VS, dxyc_rotated);                   /* [in] */
```

Step 12 **Compute the optical parameters**

The vertex position `S[]` and axis tilts `E[]` are needed for the raytrace analysis. In this calculation we rotate the ellipsoid axis by the `extra_tilt`, which corrects spherical abberation and coma, and then calculate the *vertex* of the ellipsoid using the distance from the center point `C` (which we calculated above) to the vertex to extend the axis vector. We then convert the three unit vectors to angles for the raytracing. The cross-product of `ellipsoid_axis_unit` with `vertical` implies that the current algorithm does not support any extra "yaw" motions (rotation about Y_s). A future release of this code will probably include an additional function argument `delta_yaw` which will be applied at this point to support moving the beam in cross-elevation for vibration compensation.

```
VECTOR_COPY(VC, S, VS, C);                                    /* [in] */
ANGLES_2_MATRIX(MS, Rotator,
                X, (*extra_tilt / E3), NOP,NOP, NOP,NOP, NOP,NOP, NOP,NOP);
MATRIX_VECTOR_MULT(VS, ellipsoid_axis_unit, MS, Rotator, VS, feed_pfp_unit);
VECTOR_EXTEND(VC, S,
              VS, ellipsoid_axis_unit, SemimajorAxis*M2I);     /* [in] */
/* OK, we have S[] (ellipsoid vertex point), now to get E[] (tilts). */
/* We must recompute unit vectors with extra_tilt included (the
   previous set of unit vectors were used for application of dxyc[]). */
VECTOR_CROSS(VS, sideways, VS, ellipsoid_axis_unit, VS, vertical);
```

```
VECTOR_UNIT(VS,  sideways, VS, sideways);
VECTOR_CROSS(VS, downward, VS, ellipsoid_axis_unit, VS, sideways);
VECTOR_UNIT(VS,  downward, VS, downward);
MATRIX_INIT(MS, EllipsoidRotation, /* compose matrix from unit vectors */
            VS(sideways,0), VS(downward,0), VS(ellipsoid_axis_unit,0),
            VS(sideways,1), VS(downward,1), VS(ellipsoid_axis_unit,1),
            VS(sideways,2), VS(downward,2), VS(ellipsoid_axis_unit,2));
MATRIX_2_SMALL(VC, E, MS, EllipsoidRotation);                    /* [r] */
```

Step 13 **Get subreflector "center" translation**

Using the optical parameters as input, compute required displacement of subreflector in subreflector coordinates, by computing difference wrt deflected center of home position of subreflector. Note that the last statement of this step introduces a *reflection* in the Y_S axis; the direction sense of this axis as given in [KM93] is contradicted by the conceptual drawing in Figure 1 of [Wel98a], and it appears that the latter definition corresponds to the derivation of [Nan91], which is the basis for the implementation of function `srDisplacementToLength()` [Wel98a].

```
MATRIX_TRANSPOSE(MS, EllipsoidRotInv, MS, EllipsoidRotation);
MATRIX_VECTOR_MULT(VS, S_center_rotated,
                   MS, EllipsoidRotInv, VS, S_to_center);        /* [in] */
VECTOR_ADD(VS, center_target, VC, S, VS, S_center_rotated);      /* [in] */
VECTOR_SUB(VS, sub_diff, VS, center_target, VS, center_at_elev); /* [in] */
ANGLES_2_MATRIX(MS, Rotator, X, (+GBT_SR_AXIS_TLT * DTR),
                NOP,NOP, NOP,NOP, NOP,NOP, NOP,NOP);
MATRIX_VECTOR_MULT(VC, sub_trans, MS, Rotator, VS, sub_diff);    /* [in] */
/* Change (+X_bfp,+Y_bfp,+Z_bfp) to (+X_s,-Y_s,+Z_s): */
VECTOR_PERMUTE(VC, sub_trans, 1, 2, 0, +1., -1., +1., VC, sub_trans);
```

Step 14 **Get rotation matrix of the required subreflector tilts**

We must compute differential tilts. First we take out the 5.57° tilt of the ellipsoid axis[3] which compensates the cross-polarization of the off-axis system. Then we multiply the rotation matrix of the target ellipsoid differential orientation by the transpose (inverse[4]) of the gravitational deflection rotation matrix to get matrix `Rotator`. We then permute `Rotator` from the axis order of tipping coordinates to the axis order for subreflector coordinates. The last operation of the step is to "rotate" the axes of `Rotator` by the 36.7° tilt of the subreflector focussing axis. This is done by starting with an identity matrix in the subreflector coordinates, rotating to tipping coordinates, multiplying by `Rotator` and finally rotating back to subreflector coordinates to get `ActuatorRotation`.[5]

```
ABOUT_X(MS, EllipsoidRotation, (+GBT_BETA_ANGLE * DTR));
ANGLES_2_MATRIX(MS, VertexTilt,
                X, (double)subr_center.at_elev.tilt[0],
                Y, (double)subr_center.at_elev.tilt[1],
                Z, (double)subr_center.at_elev.tilt[2],
                NOP,NOP, NOP,NOP);
```

[3] `GBT_BETA_ANGLE`, defined in `gbtOpticalConstants.h`.

[4] The tilts are small (a few milliradians), so the small angle approximation applies: the inverse could be approximated by the simple negation of the angles. Indeed, the structural model computes the tilts on the assumption that the small angle approximation is valid. At this step of the algorithm, numerical errors will cause displacements of the second focal point. The lever arm is roughly ten meters, and the author's dimensional tolerance for this work is 0.1 mm, so 10^{-5} radian is significant, a value slightly larger than the likely error committed. The author chooses to do explicit inversion at this step as a reminder that it is mathematically correct.

[5] The author is grateful to Fred Schwab for his advice on this step.

```
    MATRIX_TRANSPOSE(MS, VertexTiltInv, MS, VertexTilt);
    MATRIX_MULT(MS, Rotator, MS, EllipsoidRotation, MS, VertexTiltInv);
    MATRIX_PERMUTE(MS, Rotator, 1, 2, 0, MS, Rotator);
    /* 'Rotator' is now permuted to subreflector axis order */
    MATRIX_INIT(MS, ActuatorRotation, 1.,0.,0.,0.,1.,0.,0.,0.,1.);
    ABOUT_Z(MS, ActuatorRotation, (-GBT_SR_AXIS_TLT*DTR));
    /* 'ActuatorRotation' now in tipping system, same as 'Rotator': */
    MATRIX_MULT(MS, ActuatorRotation, MS, ActuatorRotation, MS, Rotator);
    ABOUT_Z(MS, ActuatorRotation, (+GBT_SR_AXIS_TLT*DTR));
    /* 'ActuatorRotation' is now back in subreflector coordinates. */
```

Step 15 **Get subreflector tilt angles from computed rotation matrix**

The matrix computed in the previous step now needs to be inverted to get the tilt angles to command. As discussed previously by Wells [Wel98a, Section 1.1], the three angles which are passed to function `srDisplacementToLength()` to tilt the subreflector are defined to rotate about a *skewed* set of axes; the equivalent rotation matrix is computed as the product of *five* rotations, not three. An analytic inversion is not possible. The iterative inversion algorithm used here was derived in 1997 by F.Schwab using a Mathematica program. It is interesting that the algorithm only uses three out of the nine cells of the rotation matrix. This implementation consists of a macro which is defined and then invoked three times. Numerical experiments show that three iterations will converge to less than $1\mu r$ (0.2 arcsec).

```
    VECTOR_INIT(VC, sub_tilts, 0., 0., 0.);
    calpha = cos(GBT_SR_AXIS_TLT*DTR);
    salpha = sin(GBT_SR_AXIS_TLT*DTR);
    talpha  = salpha / calpha;
    csalpha = calpha * salpha;
    c2alpha = calpha * calpha;
    s2alpha = salpha * salpha;

#define ITERATE_SR_TILTS(TT,T,TA,A)                                    \
    cthx = cos(TT(T,0)); sthx = sin(TT(T,0));                          \
    cthy = cos(TT(T,1)); sthy = sin(TT(T,1));                          \
    cthz = cos(TT(T,2)); sthz = sin(TT(T,2));                          \
    TT(T,0) -= ( TA(A,1,2) + calpha*cthy*sthx                          \
                 - sthy*(csalpha*cthx - csalpha) ) / calpha;           \
    TT(T,1) -= ( (talpha*(TA(A,1,2) + calpha*cthy*sthx                 \
                          -sthy*(csalpha*cthx - csalpha)))             \
                 -(TA(A,0,2) + cthy*salpha*sthx                        \
                   -sthy*(c2alpha + cthx*s2alpha)));                   \
    TT(T,2) -= ( TA(A,0,1) -cthz*(csalpha*cthx - csalpha)              \
                 +sthz*( cthy*(c2alpha + cthx*s2alpha) + salpha*sthx*sthy) );

    ITERATE_SR_TILTS(VC, sub_tilts, MS, ActuatorRotation);
    ITERATE_SR_TILTS(VC, sub_tilts, MS, ActuatorRotation);
    ITERATE_SR_TILTS(VC, sub_tilts, MS, ActuatorRotation);          /* [r] */
    VECTOR_SCALE(VC, sub_tilts, (1. / DTR), VC, sub_tilts);        /* [deg] */

    return(EXIT_SUCCESS);
}
```

3.1 Optimum focus tracking of gravity-induced deflections

Table 1 gives the optical prescription computed by `srFocusTracking()` as a function of elevation, expressed in a coordinate convention consistent with the `ray` package [Wel98c], in which the X-axis is coincident with

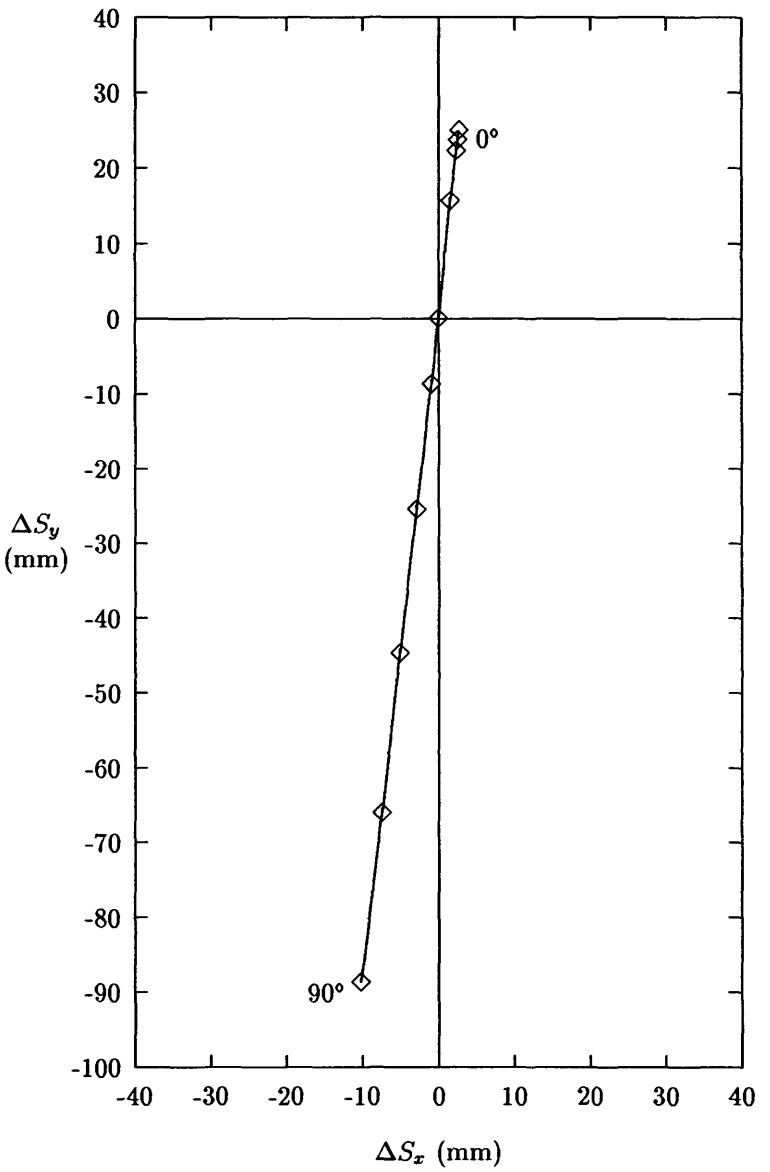

Figure 4: Focus tracking 'center' trajectory in ray-tracing coordinates (Table 1) [1:1 scale]

	PF $\mapsto F_2$ line		Feedhorn		Subreflector			Main
E	ΔL_{12}	$\Delta\theta$	ΔW_x	ΔW_y	ΔS_x	ΔS_y	$\Delta\phi$	ΔF
deg	mm	mr	mm	mm	mm	mm	mr	mm
0	-30.3	-2.5	29.3	-12.0	2.6	23.6	3.4	4.6
10	-24.7	-2.1	22.6	-22.7	2.7	24.9	3.9	2.3
20	-18.2	-1.5	15.8	-25.5	2.4	22.2	3.7	0.7
30	-11.0	-0.9	9.1	-20.3	1.6	15.6	2.7	-0.2
44	-0.0	-0.0	-0.0	0.0	-0.0	-0.0	-0.0	-0.0
50	4.9	0.4	-3.6	13.1	-0.9	-8.7	-1.6	0.6
60	13.0	1.1	-9.1	40.2	-2.8	-25.5	-4.6	2.2
70	21.1	1.7	-13.8	73.3	-5.0	-44.8	-8.2	4.5
80	28.8	2.3	-17.5	111.2	-7.4	-66.1	-12.2	7.5
90	36.0	2.9	-20.2	153.0	-10.2	-88.7	-16.5	11.0
Rigging-Angle Values								
44	11000.0		-10948.1	1067.7	4893.5	-477.2	-97.2	60000.0

Table 1: Focus tracking optical prescription which corrects gravity-induced deflections

the best-fitting optical axis of the paraboloid and the origin is coincident with the best-fitting focal point of the paraboloid. This convention is a permutation of the axes of the BFP coordinates: X (e.g. ΔS_x) is equivalent to $Z_{\rm BFP}$, Y (e.g. ΔS_y) to $Y_{\rm BFP}$ and Z to $X_{\rm BFP}$.

- ΔL_{12} is the change of the distance between the prime focal point and the feedhorn phase center from 11 meters, the focal length of the ellipsoid. This means that the ellipsoid must operate away from its focal points whenever $E \neq$ `RIGGING_ANGLE` (currently 44°), and can produce only an approximation to stigmatic imaging. $\Delta\theta$ is the `extra_tilt` which is calculated by `srEllipsoidBestRms()` at Step 8 [p.8].

- ΔW_x and ΔW_y, under the "Feedhorn" heading, are the gravitational displacement[6] of the Gregorian feedhorn from its position (-10.9481 m, +1.0677 m) *wrt* the BFP focal point and axis at the rigging angle. These values are computed by function `get_node_wrt_bfp()`.

- ΔS_x, ΔS_y and $\Delta\phi$ are the recipe for the subreflector. The ΔS values are the translation of the vertex of the ellipsoid[7] from its position at the rigging angle (+4.8935 m, -0.4772 m). $\Delta\phi$ is the tilt of the ellipsoid axis relative to the -0.097214 radian (5.57°) value it has at the rigging angle.

- ΔF is the change of paraboloid focal length as a function of elevation, relative to the 60-meter design value.

The focus tracking prescription given in Table 1 has been raytraced. The pathlength differences of rays across the 100-meter aperture have been fitted with Zernike polynomials, which are the third-order (Seidel) abberations of the system. The Seidel abberation coefficients of the GBT wavefront with this prescription for focus tracking are shown in Table 2. The algorithm presented here produces nearly zero amplitude (i.e., complete correction) of the curvature (focus), spherical abberation and coma terms of the system. There are tilt and coma wavefront residuals at some elevations, but they are quite small ($\approx 1\ \mu r$ [0.2 arcsec], ≈ 0.1 mm); currently the author suspects that these are due to minor numerical errors in the `srEllipsoidBestRms()`

[6]The symbol W is used for this displacement because the feedhorn position is the "`wave_point`" argument of the `rayGenerator()` reference in the analysis program `srFocusTrackingTable1.c`, in which a spherical wave originates from the phase center of the feedhorn.

[7]The central point of the GBT's off-axis segment of the ellipsoid has been called the "vertex" in some GBT discussions; it is different from the mathematical vertex of the ellipsoid which we are discussing here. The central point is also called the "center" in other discussions (e.g., comments and variable names at Step 13 [p.10]), but it is different from the mathematical center of the ellipsoid which is discussed in Section 2 of this memo.

		Wavefront Errors					
E	ΔP	*Curv*	*SphAb*	*Tilt*	*Coma*	*Astm*	σ
d	mm	mm	mm	μr	mm	mm	μm
0	-19.6	-0.0	0.0	-2	-0.0	-0.4	12
10	-19.0	-0.0	0.0	-1	-0.0	-0.3	9
20	-16.0	-0.0	0.0	-1	-0.0	-0.2	7
30	-10.8	-0.0	0.0	-0	-0.0	-0.1	4
44	-0.0	-0.0	0.0	-0	-0.0	-0.0	0
50	5.7	0.0	-0.0	0	0.0	0.1	2
60	16.6	0.0	-0.0	0	0.0	0.2	5
70	28.8	0.0	-0.0	-0	0.0	0.3	7
80	42.1	0.0	-0.0	-1	0.0	0.3	10
90	56.0	0.0	-0.0	-1	0.1	0.4	12

Table 2: Wavefront abberations produced by the focus-tracking prescription

solution [Wel98b, Section 6]. The significant residual optical error produced by the algorithm is astigmatism ($A_{2020}\rho^2 \cos 2\theta$), with maximum wavefront amplitude $A_{2020} \approx 0.4$ mm.

ΔP is the change of mean pathlength through the telescope; this is relevant for VLBI phase corrections. Examination of Tables 1 and 2 shows that

$$\Delta P \approx \Delta L_{12} + 2\Delta F.$$

I.e., the change of pathlength through the telescope is approximately the sum of the change of separation of between the prime focal point and Gregorian feedhorn plus twice the change of focal length of the primary mirror. The σ column in the table is the RMS error of the fit of the Zernike polynomials to the wavefront. The author presumes that this 12μm error is the sum of the 5$^{\text{th}}$-order abberation terms which are not included in the nearly-plane wavefront fitting procedure (`rayGetPlanes()` [Wel98c]). It is clear that the GBT optics are modelled well by a 3$^{\text{rd}}$-order analysis.

The author considers the existence of an algorithm which produces such small wavefront errors in the presence of $\approx$ 65 mm of defocus to be remarkable; the fact that this algorithm produces *cancellation* of both spherical abberation and coma is a strong hint that the extra ellipsoid tilt and translation used in the algorithm must be derivable from 3$^{\text{rd}}$-order optics theory. The author expects that the derivation will involve translating the coordinate origin from the axis of the parent ellipsoid to the "vertex" (center) of the off-axis segment of the ellipsoid and rotating the axes by some angle, and then discovering that this substitution of variables cancels the terms.

The author intends to correct the $A_{2020}\rho^2 \cos 2\theta$ astigmatism wavefront error of the ellipsoid by including it in the GBT's open-loop surface servo. This should essentially eliminate variation of gain with elevation, except for spillover changing as the structure distorts due to gravity.

Table 3 shows the subreflector actuator trajectory which will implement the optical prescription which was shown in Table 1. The trajectory is presented in two equivalent forms, (1) six displacements and (2) six actuator lengths. The (ΔX_S,ΔY_S) trajectory from this table is shown in Figure 5.

The direction sense of the subreflector translation coordinates is a potential source of confusion. In the current implementation the author has flipped the Y_S axis (see comment at Step 13 [p.10]) in order to make the commanded lengths of the actuators correspond to his intuitive sense of how the GBT feedarm deflects as a function of elevation. At $E = 0°$ (horizon), the vertical feedarm is horizontal and sags downward due to gravity. This will mean that the X1 and X2 actuators must shorten. The force on the vertical feedarm will also bend the horizontal feedarm (which is vertical), and it will move in the $+Z_t$ direction, which means that Y1, Y2, Y2 must lengthen. At $E = 90°$ (zenith), the situation is reversed: the vertical feedarm springs back, so X1 and X2 must extend, while the horizontal feedarm sags under the weight of the vertical feedarm,

	Displacements						Actuator Lengths					
	Translations			Tilts			Y1	Y2	Y3	X1	X2	Z1
E	ΔX_S	ΔY_S	ΔZ_S	$\Delta\theta_{X_n}$	$\Delta\theta_{Y_S}$	$\Delta\theta_{Z_S}$	ΔL_0	ΔL_1	ΔL_2	ΔL_3	ΔL_4	ΔL_5
d	in	in	in	d	d	d	in	in	in	in	in	in
0	-3.772	-0.081	0.000	0.00	0.00	0.34	-0.14	0.35	0.35	-3.81	-3.81	0.25
10	-3.672	-0.358	0.000	0.00	0.00	0.34	0.13	0.63	0.63	-3.64	-3.64	0.34
20	-3.111	-0.464	0.000	0.00	0.00	0.30	0.25	0.68	0.68	-3.05	-3.05	0.33
30	-2.101	-0.392	0.000	0.00	0.00	0.20	0.23	0.53	0.53	-2.04	-2.04	0.23
44	0.001	-0.000	0.000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	1.118	0.268	0.000	0.00	0.00	-0.11	-0.16	-0.32	-0.32	1.08	1.08	-0.11
60	3.224	0.836	0.000	0.00	0.00	-0.32	-0.49	-0.96	-0.96	3.10	3.10	-0.26
70	5.578	1.543	0.000	0.00	0.00	-0.57	-0.88	-1.69	-1.69	5.36	5.36	-0.35
80	8.111	2.371	0.000	0.00	0.00	-0.83	-1.29	-2.47	-2.47	7.80	7.80	-0.33
90	10.758	3.298	0.000	0.00	0.00	-1.11	-1.70	-3.26	-3.26	10.36	10.36	-0.18
Rigging-Angle Values												
44							110.95	110.98	110.98	127.24	127.24	61.58

Table 3: Subreflector trajectory for gravity-correcting focus tracking

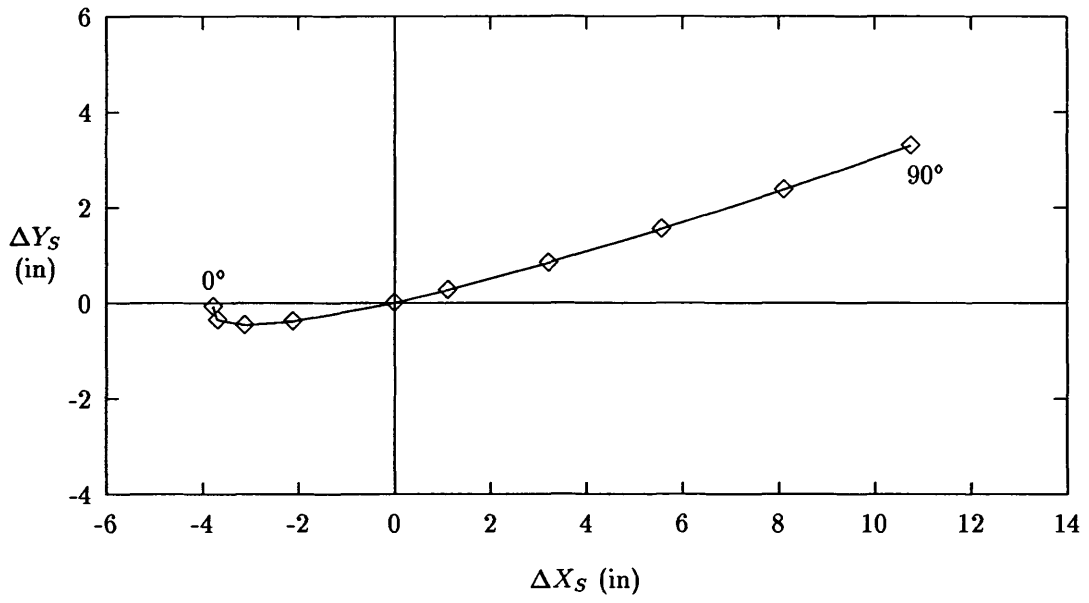

Figure 5: Gregorian focus tracking actuator trajectory (Table 3) [1:4 scale]

ΔZ_{F_2}	ΔP	Wavefront Errors					σ
		Curv	*SphAb*	*Tilt*	*Coma*	*Astm*	
mm	mm	mm	mm	μr	mm	mm	μm
0	-9.9	-0.1	0.0	2	-0.2	0.3	12
10	-11.1	-0.1	0.0	2	-0.1	0.3	10
20	-10.2	-0.0	0.0	1	-0.1	0.2	7
30	-7.3	-0.0	0.0	1	-0.1	0.1	4
44	-0.0	-0.0	0.0	0	-0.0	0.0	0
50	4.2	0.0	-0.0	-0	0.0	-0.1	2
60	12.6	0.0	-0.0	-1	0.1	-0.1	5
70	22.3	0.0	-0.0	-1	0.1	-0.2	9
80	33.2	0.1	-0.0	-2	0.2	-0.3	12
90	44.9	0.1	-0.0	-3	0.2	-0.4	15

Table 4: Wavefront abberations with no extra-tilt

so Y1, Y2, Y3 must shorten. The reader can verify by inspection that these properties hold in Table 3. The signs of $\Delta\theta_{Z_S}$ in Table 3, judging by the differential motions of Y1, Y2 and Y3, also appear to be plausible.

3.2 Abberations caused by focus-tracking without "extra-tilt"

Table 4 shows the Seidel abberation coefficients which this algorithm produces if its "extra-tilt" term is disabled. The interesting consequence is 200 microns of coma.

3.3 Comparison with previous work

There have been two prior analyses of Gregorian focus tracking:

- Enterline [Zai92, p.A2] presented a table of his "Subreflector gravity load travel range calculation." His objectives were (1) to decide the travel range which should be supported by the subreflector actuators and (2) to give preliminary numbers for the final alignment plan. It is not clear how his structural model differed from the model 95B which is used in the present work. His table does not seem to have a concept of "rigging angle". The direction sense of his X_S is flipped relative to the author's intuition. His total travel range is significantly different from that shown here in Table 3. The most significant problem for the author in comparing Enterline's table with the present work is that Enterline presents his results with no explanation of his optical analysis.

- Srikanth [Sri94] showed that focus tracking can be accomplished rather well by translation alone. His coordinate conventions are sufficiently different from those used in this work that exact numerical comparison is difficult. In particular, his version of BFP coordinates is referred to the position of the Gregorian feed at the rigging angle, rather than to the prime focal point as used in this work. In addition, his BFP coordinate system is rotated so that Z is collinear with a line connecting the vertex of the primary mirror with the secondary focal point. This is an alternative definition of a BFP coordinate system, but it is sufficiently different that the author has hesitated to attempt exact comparison of solutions. It appears that Srikanth's search procedure did not assure that the first focus of the ellipsoid was nearly coincident with the PFP. Not only does this produce a tilt, it means that abberations of the primary contribute to the optical system's properties. Srikanth's demonstration that good solutions of this kind are possible implies that the interesting solution space is larger than that covered by the present `srFocusTracking()` algorithm.

4 Sources of confusion: multiple conventions

The focus tracking function depends on and interacts with many other functions. Its computations involve physical quantities which are expressed in several coordinate systems and units, with multiple angle and axis-permutation conventions. These differences produce a situation in which errors (software bugs) due to confusion of conventions, coordinates and units are likely to occur. Anyone who is going to maintain this code, or even to attempt to understand it, must be constantly aware of these differences, which sometimes occur in the same C statement! It is pointless to criticize or even to complain about this situation: the problem is inherent in the fact that we are integrating a variety of subsystems and components which were developed independently over a period of time by different people, and in the fact that we are doing this project in the USA which has not yet fully adopted SI units conventions. Sorry about that...

- **Dimensional Units**

  Both metric units and inches are used in the GBT project: the structural model is expressed in inches, but the focal lengths of the optical elements are specified in meters. Conversions between these units are pervasive in the code shown above; multiplying a number by the symbolic constant `M2I` (meters-to-inches) converts it from meters to inches (e.g., Step 9 [p.8], Step 5 [p.7].)

- **Temperature Units**

  The Centigrade versus Fahrenheit distinction is a trap set for anybody who tries calculating thermal expansion. For example, at Step 1 [p.6] the symbol `RIGGING_TEMP` is in Centigrade and has value 21.1°C (*not* 20° as you might expect), because the GBT official project specifications give the rigging temperature as 70°F. Expansion coefficients are given in some handbooks per degree Centigrade and in other sources per degree Fahrenheit—caveat emptor! The author may decide to define macros `C2F()` and `F2C()` in the future to minimize the confusion.

- **Angle Units**

  The classic programming problem of conversion between degrees and radians pervades this code; multiplication by the symbol `DTR` (degrees-to-radians) does this (e.g., see Step 5 [p.7] and Step 15 [p.11]). Note the radians-to-degrees conversion at the end of Step 15 [p.11]—function `srDisplacementToLength()` wants *degrees* for this vector. Some small angles are computed in milliradians for convenience; this necessitates conversion to radians by dividing by `E3` (1000) in other places (e.g., see Step 12 [p.9]).

- **Coordinate Systems**

  The basic structural model [WK95b] is expressed in the "tipping" coordinate system [KM93], which has Z pointing upward, Y pointing from the feedarm to the outer edge of the primary mirror and X pointing from the GBT plane of symmetry toward the right when viewed along the Y axis. The origin is at the pintle bearing. The "BFP" coordinate system (defined in [WK95a], not [KM93]) is similar to the tipping system except that the origin is moved to the prime focal point and the Y and Z axes are rotated about the X axis by a few milliradians due to the rotation of the paraboloid which best-fits the homologously-deforming primary mirror as a function of elevation. The real confusion occurs when we consider subreflector coordinates [KM93, Wel98a], in which the origin is at the "vertex" (center) position of the subreflector at the rigging angle and the axes are permuted ($X_t \mapsto Z_s$, $Y_t \mapsto X_s$, $Z_t \mapsto Y_s$) and axes X_s and Y_s are rotated about Z_s by `GBT_SR_AXIS_TLT` (36.7° [Sri90]).[8] The axis convention used by the `ray` package [Wel98c] is different from the tipping and BFP conventions—`ray` traces the system with the optical axis parallel to the X coordinate, while the GBT optical axis is parallel to the Z_t and Z_{BFP} axes.

[8] The X rotation axis of the subreflector is called X_n, rather than X_s, because it is perpendicular to Z_t (the axis of the paraboloid) and rotation about it accomplishes the beam "nutation" motion of the subreflector. The (X_n, Y_s, Z_s) *rotation* axes are *skewed*.

- **Rotation Angles**

  Confusion about the signs of rotations is inevitable in calculations of this type; the most basic case is the question of whether we are rotating an object or rotating the axes to which the object coordinates are referred. For rotations by sets of angles, we have two possibilities: Euler angles and small-angle rotations. Both types are used in various places in the GBT project. For the small-angle case, if the angles are truly small, the order of the rotations is immaterial; for example, the tilts computed by the GBT structural model are typically of order one milliradian, so cross-product errors will be about one microradian (0.2 arcsec), which we can take to be negligible. However, the angles of the `tilts[]` argument of function `srDisplacementToLength()` are *not* negligible—they may be more than one degree, so the order in which they are applied does matter. The axis permutations which were discussed in the **Coordinate Systems** section above apply to rotation angles also, of course. All of these issues arise in the conversion of rotation matricies back into rotation angles, which occurs in several places in the code. The nasty case of subreflector coordinates is especially interesting, as computation of the rotation matrix takes five rotations, not three [Wel98a] and its inversion must be interative. This problem arises at Step 15 [p.11].

5 Features expected to be added in future versions

- The rigging angle will be changed to 50.8°
- Assymetric structural models will become available
- Temperature compensation will be fully implemented
- Function output `E[]` will be changed to use an Euler angle convention when the `ray` package [Wel98c] makes this change.
- An optimization strategy option argument will probably be added to the function. The default mode (the current implementation) will be gain optimization. The most likely alternative strategy will be cross-polarization minimization as a function of elevation; for Zeeman observations it might be better to trade a coma lobe for cross-pol control.
- The author believes that the readability of the vector-matrix operations in this algorithm would be improved by translation to C++; however, he expects that the performance would suffer, especially if the compiler has good scalar dataflow analysis in its optimization repertoire. Currently the author does not think that the tradeoffs favor making the language change, but this opinion is subject to change.

References

[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.

[Nan91] P. B. Nanavati. Equations of motion – subreflector positioner. Loral GBT Technical Memo 19, Loral Western Development Labs, September 1991. The "inverse kinematic" algorithm for calculating actuator lengths from displacements is derived by both matrix/vector and trigonometric approaches. Appendix A includes a copy of NRAO drawing B35102M007 which specifies the "nutation" axis X_n.

[Sri90] S. Srikanth. Axial focussing. GBT Memo 49, NRAO, April 1990. "..the axis along which the [subreflector is] to be moved (without the telescope beam shifting in the sky) for axial focussing and the amount of movement required [are] determined..; the axis of focussing is 36.73° from the axis of the.. main reflector.. [and] 24.4° from the secondary focus feed axis..".

[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.

[Wel98a] Don Wells. GBT subreflector actuator functions in C. GBT Memo 175, NRAO, January 1998. Two ANSI-C functions are described: `srDisplacementToLength()`, which accepts displacements (three linear, three angular) from the "home" position of the subreflector and computes six actuator lengths, and `srLengthToDisplacement()`, which performs an iterative numerical inversion of function `srDisplacementToLength()` (i.e., it accepts actuator lengths and produces displacements). These two functions will be used in the GBT Monitor and Control software to transform Gregorian focus tracking outputs into actuator commands. Partial derivatives of the actuator lengths *wrt* translation and tilt are tabulated.

[Wel98b] Don Wells. Imaging properties of the GBT subreflector in C. GBT Memo 179, NRAO, March 1998. The.. subreflector.. images points in the neighborhood of its first focus onto points in the neighborhood of its second focus.. nearly-stigmatic imaging.. can be obtained for a variety of tilts and displacements.. cases.. computed by ray tracing.. minimum phase error.. if.. ellipsoid is tilted slightly.. results.. fitted with polynomials.. expressed in C. *(The current version of this memo is 179.2, dated 1998-04-28; the changes are that the best-RMS function is now constrained to zero for* $\Delta S_{12} = 0$ *and that* `srEllipsoidBestRms()` *is fitted over a wider range.).*

[Wel98c] Don Wells. The "`ray`" ray tracing package. GBT Memo 178, NRAO, March 1998. Abstract: "The '`ray`' package and program `rayMain` trace sets of rays (representing wavefronts) through systems of rotationally-symmetric aspheric optical elements. The starting sets of rays can represent either plane or spherical wavefronts, with feedhorn tapering. The optical elements can be de-centered and/or tilted conic sections (planes, spheres, ellipsoids, paraboloids, hyperboloids) with additional superimposed radially-symmetric aspheric terms, and they can be mirrors as well as refracting surfaces. Both foci and nearly-plane wavefronts can be analyzed." See `ftp://fits.cv.nrao.edu/pub/gbt_dwells_ray.tar.gz` (155 KB).

[WK95a] Don Wells and Lee King. GBT Best-Fitting Paraboloid [BFP] in C. GBT Memo 131, NRAO, June 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. See `ftp://fits.cv.nrao.edu/pub/gbt_dwells_doc.tar.gz` for the current revision of this memo (131.5 as of 1998-04-20).

[WK95b] Don Wells and Lee King. The GBT Tipping-Structure Model in C. GBT Memo 124, NRAO, 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 `ftp://fits.cv.nrao.edu/pub/gbt_dwells_doc.tar.gz` for the current revision of this memo (124.3 as of 1997-06-23).

[Zai92] J. Zaine. Mechanical analysis S/R positioner. Loral GBT Technical Memo 46, Loral Western Development Labs, October 1992. This comprehensive report discusses operational, peak operational and survival loadings, required motor torques, lost motion (backlash) of the U-joints and possible structural interference and U-joint angle problems of the subreflector actuators. Portions of Loral Interoffice Memorandum 3WL110-DLE-107, by D. L. Enterline and titled "NRAO 100m GBT Prime Focus Feed & Subreflector Positioner Gravity Correction Travel Requirements", are included in Appendix A (pp.A1-A13). The asymmetric structural model which was used enabled calculation of the required out-of-plane Z_s translation ± 0.83in.