```
#include "RAPID.H"
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#define    pi    3.1415926535897932

long int   bfacets, nfacets;
int     bcount=0, ncount=0,contact_set=1, loop_chk=0, third_pt = 0;
int     base=0, nsurf=0, n_edge=0, counter=0, flush_true;
double  midpoint[3], xc, yc, zc, nbz, nnz, b_cos_z, n_cos_z;
double  d1, d2, x_t[2], y_t[2], z_t[2];
double  C2_bolt[3],C2_fixed[3], Q[3][3], U[3], step=1.0, xroll=0.0;
double  x[2],y[2],z[2],P[3][3], pointX[5], pointY[5], pointZ[5];
double  BWt_i, BWt_j, BWt_k, NWt_i, NWt_j, NWt_k;
double  Com_Norm_i, Com_Norm_j, Com_Norm_k, B1[3][3], S[3];
double  gamma, M[3][3], C3_bolt[3], C3_fixed[3], XP[3][3];
double  BMt_i, BMt_j, BMt_k, BFt_i, BFt_j, BFt_k;
double  CN_b_i, CN_b_j, CN_b_k, NBt_i, NBt_j, NBt_k;
double  loc_xc, loc_yc, loc_zc, rate, psi_step, psi, g1, g2;
double  TP[] = { 0, 0, 0 };
double  alpha=0.0, beta=0.0, phi=0.0, R[3][3], pX_f, pY_f, pZ_f;;
double  pX_p0, pY_p0, pZ_p0, R1[3][3], T1[3], B2_fixed[3];

//Function Prototypes
void stl_reader(int, int, float (*array_bolt_tri)[13], float (*array_nut_tri)[13],
RAPID_model *bolt, RAPID_model *nut);

double init_collision_detect(double R1_b[][3], double T1_b[], RAPID_model *bolt_one,
double R2_n[][3], double T2_n[], RAPID_model *nut_one, int, float (*pb_init_tri)[13]);

double contact_point(int, double array[][3], double array2[], float (*array_b)[13],
float (*array_n)[13]);

double bolt_surf(double array[][3], double, double, double, double, int);

double nut_surf(double, double, double, int);

void second_model_input(double R1b_2[][3], double T1b_2[], RAPID_model *bolt_two,
float (*pb2_tri_2)[13], float (*pb_tri_2)[13], RAPID_model *nut_two, float
(*pn2_tri_2)[13], float (*pn_tri_2)[13]);

double mult_contact(int, double array_R[][3], double array_T[], float (*array_b2)[13],
float (*array_n2)[13], double R1_bolt[][3], double);

double tang_to_fixed(double vec_array[]);

double tang_to_bolt(double vec_arrayB[]);

double pivot_axes_theo(double phi);

void third_model_input(double fix_R[][3], double fix_T[], RAPID_model *bolt3, float
(*pb3_tri)[13], float (*pb_tri_3)[13], RAPID_model *nut3, float (*pn3_tri)[13], float
(*pn_tri_3)[13], double);

void special_third(double fx_sp_R[][3], double fx_sp_T[], RAPID_model *bolt3S, float
(*pb3S_tri)[13], float(*pb_tri_3S)[13], RAPID_model *nut3S, float (*pn3S_tri)[13],
float (*pn_tri_3S)[13], double, double);

double contact_group(int, float (*b3_ctct_tri)[13], float (*n3_ctct_tri)[13]);
```

```
double minimum(double, double, double);
double pivot_to_fixed(double, double, double);
double pivot_to_bolt(double, double, double);
double blt_vector(double, double, double);
double nut_vector(double, double, double);
double common_normal(double, double, double, double, double, double);
double fixed_to_bolt(double, double, double, double RT[][3]);
double bolt_to_fixed(double, double, double);
double bolt_to_pivot(double, double, double);
double bolt_to_FIXED(double, double, double);

int main()
{
   //Establish Initial RAPID models
   RAPID_model *bolt = new RAPID_model;
   RAPID_model *nut  = new RAPID_model;
   cout << "Loading tris into RAPID_model objects..." << flush;
   // Declare input file streams
   ifstream bolt_file;
   ifstream nut_file;
   // Open files in binary mode
   bolt_file.open("bv7.stl",ios::in|ios::binary);
   nut_file.open("nv6.stl",ios::in|ios::binary);
   if (bolt_file.bad()) {
      cerr << "Unable to open bv7.stl\n";
      exit (8);}
   if (nut_file.bad()) {
      cerr << "Unable to open nv6.stl\n";
      exit (8);}

   // Skip (seekg) 80 bytes from the beg of the file
   bolt_file.seekg(80, ios::beg);
   // Get the number of facets
   bolt_file.read((char*)&bfacets, sizeof(long int));
   bolt_file.close();
   //Create an 2D array in memory where bfacets
   //is the #rows, & 13 is the #of columns
   float (*pb_tri)[13] = new float[bfacets][13];
   nut_file.seekg(80, ios::beg);
   nut_file.read((char*)&nfacets, sizeof(long int));
   nut_file.close();
   float (*pn_tri)[13] = new float[nfacets][13];
   //Store Binary .STL files into memory pointers pb_tri & pn_tri
   stl_reader(bfacets, nfacets, pb_tri, pn_tri, bolt, nut);

   //The following arrays locate bolt with 3 DOF. We assume RCC corrects
   //the bolts lateral errors.  Nut is fixed in space and is not altered.
   double  R2[3][3], T2[3];

   R2[0][0] = R2[1][1] = R2[2][2] = 1.0;//Rotation of Nut
   R2[0][1] = R2[1][0] = R2[2][0] = 0.0;
   R2[0][2] = R2[1][2] = R2[2][1] = 0.0;
   T2[0] = T2[1] = T2[2] = 0.0;       //Translation of Nut
   T1[0] = T1[1] = T1[2] = 0.0;       //Translation of Bolt

   //This Loop Interates through all possible orientations
   ////////////////////
   for (int tilt=-20; tilt <=20; tilt=tilt+5) {
      for (int pitch=-20; pitch <=20; pitch=pitch+5) {
         for (int phase=0; phase <=359; phase=phase+1) {
            if ((tilt != 0) || (pitch != 0)) {
               alpha= (double)tilt*(1.0/5.0)*pi/180.0;
```

```
beta  = (double)pitch*(1.0/5.0)*pi/180.0;
phi   = (double)phase*pi/180.0;

//Transformation Matrix -- We already took the Transpose of [R]
R1[0][0] = cos(beta)*cos(phi);
R1[0][1] = sin(alpha)*sin(beta)*cos(phi)-cos(alpha)*sin(phi);
R1[0][2] = cos(alpha)*sin(beta)*cos(phi)+sin(alpha)*sin(phi);
R1[1][0] = cos(beta)*sin(phi);
R1[1][1] = sin(alpha)*sin(beta)*sin(phi)+cos(alpha)*cos(phi);
R1[1][2] = cos(alpha)*sin(beta)*sin(phi)-sin(alpha)*cos(phi);
R1[2][0] = -sin(beta);
R1[2][1] = sin(alpha)*cos(beta);
R1[2][2] = cos(alpha)*cos(beta);

//This function lowers the bolt onto the nut in dimishinging steps
//the step size begins at 0.001 inches and terminates at 1 micro inche
T1[2] = init_collision_detect(R1, T1, bolt, R2, T2, nut, bcount, pb_tri);
//This function calls the collision detection library and it returns
//the ID#'s of the contact pairs
RAPID_Collide(R1, T1, bolt, R2, T2, nut, RAPID_ALL_CONTACTS);

// Store the contact pairs into bolt and nut memory matrices
float (*bolt_facet_stor)[13] = new float[RAPID_num_contacts][13];
float (*nut_facet_stor)[13] = new float[RAPID_num_contacts][13];
for (int k=0; k < RAPID_num_contacts; k++) {
   for (int n=0; n <= 12; n++) {
      *(*(bolt_facet_stor+ counter) + n) =
                                  *(*(pb_tri+RAPID_contact[k].id1)+n);
      *(*(nut_facet_stor + counter) + n) =
                                  *(*(pn_tri+RAPID_contact[k].id2)+n);
   }
   counter++;
}

cout << setw(2) << alpha*180.0/pi << "  " << setw(2) << beta*180.0/pi
     << "  " << setw(3) << phi*180.0/pi << "\t" << counter << endl;

//Obtain the initial contact point -- needed to convert to C0 Ref Frame
//NOTE -- we do not check to see if there are multiple contacts at the
//intial config because that situation occurs only once
// -- when alpha=-1,-2,or-3, beta=0, phi=0
contact_point(counter, R1, T1, bolt_facet_stor, nut_facet_stor);
xc = midpoint[0];
yc = midpoint[1];
zc = midpoint[2];
//The following loop checks to see the facets normal vector
//If it is pointing down, a base tri on the bolt is involved in contact
//If it is pointing up, a surface tri on the nut is involved in contact
for (int bck=0; bck <= counter-1; bck++) {
   nbz   = (double)*(*(bolt_facet_stor + bck) +3);
   nnz   = (double)*(*(nut_facet_stor + bck) +3);
   b_cos_z = acos(nbz)*180.0/pi;
   n_cos_z = acos(nnz)*180.0/pi;
   if ((b_cos_z >= 170.0) && (b_cos_z <= 190.0)) {
      base  = base + 1;
   }
   if ((n_cos_z >= 0.0) && (n_cos_z <= 10.0)) {
      nsurf  = nsurf + 1;
   }
   if ((n_cos_z < 0.0) || (n_cos_z > 10.0)) {
      n_edge = n_edge + 1;
   }
}
```

```
            //This means that the interior edge of the NUT is in contact
            if (n_edge != 0) {
               bolt_surf(R1,xc,yc,zc,T1[2],base);
               nut_surf(xc,yc,zc,nsurf);
               common_normal(BWt_i, BWt_j, BWt_k, NWt_i, NWt_j, NWt_k);
            } else if (n_edge == 0) {
               //This occurs when the BOLT contacts the surface of the NUT
               Com_Norm_i = 0.0;
               Com_Norm_j = 0.0;
               Com_Norm_k = 1.0;
               //We still need to call bolt_surf so we can determine the C0 frame
               bolt_surf(R1,xc,yc,zc,T1[2],base);
               double zeta=0.0;
               //Here we find the dir. cosines for a vector tang. to the nut surface
               //when contact does not involve the interior edge of the nut.
               if ((xc >= 0.0) && (yc >= 0.0)) {//Quadrant I
                  zeta = atan(yc/xc);
                  NWt_i = cos(pi/2.0 + zeta);
                  NWt_j = cos(zeta);
               } else if ((xc < 0.0) && (yc >= 0.0)) {//Quadrant II
                  zeta = atan(fabs(xc/yc));
                  NWt_i = cos(pi - zeta);
                  NWt_j = cos(pi/2.0 + zeta);
               } else if ((xc < 0.0) && (yc < 0.0)) {//Quadrant III
                  zeta = atan(fabs(yc/xc));
                  NWt_i = cos(pi/2.0 - zeta);
                  NWt_j = cos(pi - zeta);
               } else if ((xc >= 0.0) && (yc < 0.0)) {//Quadrant IV
                  zeta = atan(fabs(xc/yc));
                  NWt_i = cos(zeta);
                  NWt_j = cos(pi/2.0 - zeta);
               }
               NWt_k = 0.0;
            }

            cout << "INITIAL CONTACT POINT" << endl;
            cout << "X\t" << xc << "\tY\t" << yc << "\tZ\t" << zc << endl;
            cout << "T1[2]\t" << T1[2] << endl;
            base=0; nsurf=0; n_edge=0; counter=0;
            delete [] bolt_facet_stor;
            delete [] nut_facet_stor;

////////////////////////////////////////////////////////////////////////////
// This is where we convert BOLT and NUT vertices to the reference frame defined//
// by the tangent vector to the NUT parametric equation.  It is a first approx.  //
// as to the orientation of the axis by which the BOLT will rotate due to the   //
// influence of the RCC.  The tangent vector is labelled as the X-axis, and     //
// it is directed CCW.  The Y-axis points towards the center of the NUT          //
////////////////////////////////////////////////////////////////////////////

            //The following expressions determine the transformation
            //from the BOLT to the C0 frame
            NBt_i = R1[0][0]*NWt_i + R1[1][0]*NWt_j + R1[2][0]*NWt_k;
            NBt_j = R1[0][1]*NWt_i + R1[1][1]*NWt_j + R1[2][1]*NWt_k;
            NBt_k = R1[0][2]*NWt_i + R1[1][2]*NWt_j + R1[2][2]*NWt_k;
            if (((xc<=0.0) && (yc>0.0)) || ((xc<=0.0) && (yc<=0.0))) {
            //Quadrant II or III requires negative Z-axis rotation
               g1 = -acos(NBt_i/sin(acos(NBt_k)));
            } else if (((xc>0.0) && (yc>0.0)) || ((xc>0.0) && (yc<=0.0))) {
            //Quadrant I or IV requires positive Z-axis rotation
               g1 = acos(NBt_i/sin(acos(NBt_k)));
            }
            g2 = acos(NBt_k) - pi/2.0;
```

```
B1[0][0] = cos(g2)*cos(g1);  B1[0][1] = cos(g2)*sin(g1);
B1[0][2] = -sin(g2);         B1[1][0] = -sin(g1);
B1[1][1] = cos(g1);          B1[1][2] = 0.0;
B1[2][0] = sin(g2)*cos(g1);  B1[2][1] = sin(g2)*sin(g1);
B1[2][2] = cos(g2);

S[0] = R1[0][0]*xc + R1[1][0]*yc + R1[2][0]*zc - R1[2][0]*T1[2];
S[1] = R1[0][1]*xc + R1[1][1]*yc + R1[2][1]*zc - R1[2][1]*T1[2];
S[2] = R1[0][2]*xc + R1[1][2]*yc + R1[2][2]*zc - R1[2][2]*T1[2];
//d1 is a rot. about the Z-axis of the fixed frame
//and d2 is a rotation about the modified Y-axis
//The two rotations line up the X-axis with a general vector in space.
if (((xc<=0.0) && (yc>0.0)) || ((xc<=0.0) && (yc<=0.0))) {
//Quadrant II or III requires negative Z-axis rotation
   d1 = -acos(NWt_i/sin(acos(NWt_k)));
} else if (((xc>0.0) && (yc>0.0)) || ((xc>0.0) && (yc<=0.0))) {
//Quadrant I or IV requires positive Z-axis rotation
   d1 = acos(NWt_i/sin(acos(NWt_k)));
}
d2 = acos(NWt_k) - pi/2.0;

Q[0][0] = cos(d2)*cos(d1); Q[0][1] = cos(d2)*sin(d1); Q[0][2] = -sin(d2);
Q[1][0] = -sin(d1);        Q[1][1] = cos(d1);         Q[1][2] = 0.0;
Q[2][0] = sin(d2)*cos(d1); Q[2][1] = sin(d2)*sin(d1); Q[2][2] = cos(d2);

U[0] = xc;
U[1] = yc;
U[2] = zc;

//Declare New Models for the 2nd contact point analysis
RAPID_model *bolt2 = new RAPID_model;
RAPID_model *nut2  = new RAPID_model;
float (*pb2_tri)[13] = new float[bfacets][13];
float (*pn2_tri)[13] = new float[nfacets][13];
//Create New Model of BOLT & NUT w/origin at tangent ref frame
second_model_input(R1,T1,bolt2,pb2_tri, pb_tri, nut2, pn2_tri, pn_tri);

//**********************************//
//Perform 2nd Contact Point Analysis//
//**********************************//
double  R3[3][3], R4[3][3], T3[3], T4[3];
int     aa, qq, run_num=0;

R4[0][0] = 1.0; R4[0][1] = 0.0; R4[0][2] = 0.0;
R4[1][0] = 0.0; R4[1][1] = 1.0; R4[1][2] = 0.0;
R4[2][0] = 0.0; R4[2][1] = 0.0; R4[2][2] = 1.0;
T4[0] = 0.0;  T4[1] = 0.0;  T4[2] = 0.0;
T3[0] = 0.0;  T3[1] = 0.0;  T3[2] = 0.0;
int loop_chk=0;
//This guys checks to see if BOLT is flush with the NUT surface
flush_true = 0;
while (contact_set == 1) {
   //Transpose of the rotation matrix about the X-axis of the C0 frame
   R3[0][0] = 1.0; R3[0][1] = 0.0;         R3[0][2] = 0.0;
   R3[1][0] = 0.0; R3[1][1] = cos(xroll);  R3[1][2] = -sin(xroll);
   R3[2][0] = 0.0; R3[2][1] = sin(xroll);  R3[2][2] = cos(xroll);
   //Original rotation matrix about the X-axis of the C0 frame
   M[0][0] = 1.0;  M[0][1] = 0.0;          M[0][2] = 0.0;
   M[1][0] = 0.0;  M[1][1] = cos(xroll);   M[1][2] = sin(xroll);
   M[2][0] = 0.0;  M[2][1] = -sin(xroll);  M[2][2] = cos(xroll);

   RAPID_Collide(R3, T3, bolt2, R4, T4, nut2, RAPID_ALL_CONTACTS);
```

```
   //Store Tri Contact Pairs
   float (*bolt2_contact_tri)[13] = new float[RAPID_num_contacts][13];
   float (*nut2_contact_tri)[13] = new float[RAPID_num_contacts][13];
   for (qq=0; qq < RAPID_num_contacts; qq++) {
      for (aa=0; aa <= 12; aa++) {
         *(*(bolt2_contact_tri + qq) + aa) =
                              *(*(pb2_tri+RAPID_contact[qq].id1) + aa);
         *(*(nut2_contact_tri + qq) + aa) =
                              *(*(pn2_tri+RAPID_contact[qq].id2) + aa);
      }
   }
   //Determine if multiple contacts exist
   mult_contact(RAPID_num_contacts, R3, T3, bolt2_contact_tri,
                nut2_contact_tri, R1, phi);

   delete [] bolt2_contact_tri;
   delete [] nut2_contact_tri;
}//End of "2nd Contact Point" While Loop

delete bolt2;
delete nut2;
delete [] pb2_tri;
delete [] pn2_tri;

double  rad_to_C1, rad_to_C2, stor_x, stor_y, stor_z, stor_x_t, stor_y_t;
double  stor_z_t,TX,TY,TZ,CMAG,PTx,PTy,PTz,tP1,tP2,xf,m;
if (flush_true != 1) {//We maintain the first contact point
//Rearrange the contact information so X0, Y0, Z0
//represent the origin of the C0 frame
   rad_to_C1 = sqrt(pow(x_t[0],2) + pow(y_t[0],2) + pow(z_t[0],2));
   rad_to_C2 = sqrt(pow(x_t[1],2) + pow(y_t[1],2) + pow(z_t[1],2));
   if (rad_to_C1 > rad_to_C2) {
      stor_x_t = x_t[0];
      stor_y_t = y_t[0];
      stor_z_t = z_t[0];
      stor_x = x[0];
      stor_y = y[0];
      stor_z = z[0];
      x_t[0] = x_t[1];
      y_t[0] = y_t[1];
      z_t[0] = z_t[1];
      x[0] = x[1];
      y[0] = y[1];
      z[0] = z[1];
      x_t[1] = stor_x_t;
      y_t[1] = stor_y_t;
      z_t[1] = stor_z_t;
      x[1] = stor_x;
      y[1] = stor_y;
      z[1] = stor_z;
   }
   //Compute the direction cosines of the X-axis in the P0 frame
   CMAG = sqrt(pow(x_t[1],2) + pow(y_t[1],2) + pow(z_t[1],2));
   PTx = x_t[1]/CMAG;
   PTy = y_t[1]/CMAG;
   PTz = z_t[1]/CMAG;
} else if (flush_true == 1) {
//We are flush with surface and lose our first contact point
   TX = x[1] - x[0];
   TY = y[1] - y[0];
   TZ = z[1] - z[0];
   CMAG = sqrt(pow(TX,2) + pow(TY,2) + pow(TZ,2));
   PTx = TX/CMAG;
```

```
   PTy = TY/CMAG;
   PTz = TZ/CMAG;
}
//These two parameters are used to determine the direction of rotation
m  = (y[1] - y[0])/(x[1] - x[0]);
xf = x[0] - y[0]/m;

if (flush_true == 1) {//We lose first contact point
   if ((x[0] >= 0.0) && (y[0] >= 0.0)) {//CP0 is in Quadrant I
      tP1 = -acos(PTx/(sin(acos(PTz))));
      if (xf >= 0.0) {//Xp intersets pos Xf
         if (m < 0.0) {//slope of Xp line is neg
            if ((x[1] < 0.0) && (y[1] >= 0.0)) {//CP1 is in Quadrant II
               tP1 = acos(PTx/(sin(acos(PTz))));
            }
         }
      }
   } else if ((x[0] < 0.0) && (y[0] >= 0.0)) {//CP0 is in Quadrant II
      tP1 = -acos(PTx/(sin(acos(PTz))));
      if (xf < 0.0) {//Xp intersects the neg Xf
         if (m >= 0.0) {//slope of Xp line is pos
            if ((x[1] >= 0.0) && (y[1] >= 0.0)) {//CP1 is in Quadrant I
               tP1 = acos(PTx/(sin(acos(PTz))));
            }
         }
      }
   } else if ((x[0] < 0.0) && (y[0] < 0.0)) {//CP0 is in Quadrant III
      tP1 = acos(PTx/(sin(acos(PTz))));
      if (xf < 0.0) {//Xp intersects the neg Xf
         if (m < 0.0) {//slope of Xp line is neg
            if ((x[1] >= 0.0) && (y[1] < 0.0)) {//CP1 is in Quadrant IV
               tP1 = -acos(PTx/(sin(acos(PTz))));
            }
         }
      }
   } else if ((x[0] >= 0.0) && (y[0] < 0.0)) {//CP0 is in Quadrant IV
      tP1 = acos(PTx/(sin(acos(PTz))));
      if (xf >= 0.0) {//Xp intersects the pos Xf
         if (m >= 0.0) {//slope of Xp line is pos
            if ((x[1] >= 0.0) && (y[1] < 0.0)) {//CP1 is in Quadrant IV
               tP1 = -acos(PTx/(sin(acos(PTz))));
            }
         }
      }
   }
} else if (flush_true != 1) {//We maintain initial contact point
   //Calculate the two rotation angles to line
   //up the X-axis of the C0 frame with the X-axis of the P0 frame.
   if (PTx == 0.0) {
      if ((acos(PTy) > 0.0) && (acos(PTy) < pi/2.0)) {
         tP1 = pi/2.0;
      } else if ((acos(PTy) > pi/2.0) && (acos(PTy) < pi)) {
         tP1 = -pi/2.0;
      }
   } else if (PTx != 0.0) {
      if (acos(PTy) > pi/2.0) {
         tP1 = -acos(PTx/(sin(acos(PTz))));
      } else if (acos(PTy) <= pi/2.0) {
         tP1 = acos(PTx/(sin(acos(PTz))));
      }
   }
}
tP2 = acos(PTz) - pi/2.0;
```

```
P[0][0] = cos(tP2)*cos(tP1); P[0][1] = cos(tP2)*sin(tP1);
P[0][2] = -sin(tP2);         P[1][0] = -sin(tP1);
P[1][1] = cos(tP1);          P[1][2] = 0.0;
P[2][0] = sin(tP2)*cos(tP1); P[2][1] = sin(tP2)*sin(tP1);
P[2][2] = cos(tP2);

//Create new models that define the current position
//of the BOLT and NUT as viewed from the P0 frame
RAPID_model *bolt3 = new RAPID_model;
RAPID_model *nut3  = new RAPID_model;
float (*pb3_tri)[13] = new float[bfacets][13];
float (*pn3_tri)[13] = new float[nfacets][13];
if (flush_true == 1) {
   //We are flush with the surface, hence TANG frame is not preserved
   special_third(R1, T1, bolt3, pb3_tri, pb_tri, nut3, pn3_tri, pn_tri,
                 xroll, phi);
} else if (flush_true != 1) {
   //We are not flush with the surface, hence TANG frame is preserved
   third_model_input(R1, T1, bolt3, pb3_tri, pb_tri, nut3, pn3_tri,
                     pn_tri, xroll);
}
////////////////////////////////////////////////////////////
//Welcome to the Pivot Axis Direction of Rotation Algorithm//
////////////////////////////////////////////////////////////
rate = 0.01;//Rate of rotation about the X-axis of the P0 frame per step
if ((x[0] >= 0.0) && (y[0] >= 0.0)) {//CP0 is in QUAD I
   if (xf >= 0.0) {//Xp-Axis INT POS Xf-Axis
      if (m < 0.0) {
      //Slope of Xp-Axis is negative as viewed from the FIXED-frame
         if ((x[1] < 0.0) && (y[1] >= 0.0)) {//NEG rot about the Xp-Axis
            psi_step = -rate;
         } else if ((x[1] >= 0.0) && (y[1] < 0.0)) {
         //POS rot about the Xp-Axis
            psi_step =  rate;
         }
      } else if (m >= 0.0) {
         if (y[1] < 0.0) {
            psi_step =  rate;
         }
      }
   } else if (xf < 0.0) {//Xp-Axis INT NEG Xf-Axis
      if (m >= 0.0) {
         if (x[1] < 0.0) {
            psi_step = -rate;
         }
      }
   }
} else if ((x[0] < 0.0) && (y[0] >= 0.0)) {//CP0 is in QUAD II
   if (xf >= 0.0) {
      if (m < 0.0) {
         if (x[1] >= 0.0) {
            psi_step =  rate;
         }
      }
   } else if (xf < 0.0) {
      if (m < 0.0) {
         if (y[1] < 0.0) {
            psi_step = -rate;
         }
      } else if (m >= 0.0) {
         if (x[1] < 0.0) {
            psi_step = -rate;
         } else if (x[1] >= 0.0) {
```

```
            psi_step =  rate;
         }
      }
   }
} else if ((x[0] < 0.0) && (y[0] < 0.0)) {//CP0 is in QUAD III
   if (xf >= 0.0) {
      if (m >= 0.0) {
         if (x[1] >= 0.0) {
            psi_step = -rate;
         }
      }
   } else if (xf < 0.0) {
      if (m < 0.0) {
         if ((x[1] < 0.0) && (y[1] >= 0.0)) {
            psi_step =  rate;
         } else if (y[1] < 0.0) {
            psi_step = -rate;
         }
      } else if (m >= 0.0) {
         if (y[1] >= 0.0) {
            psi_step =  rate;
         }
      }
   }
} else if ((x[0] >= 0.0) && (y[0] < 0.0)) {//CP0 is in QUAD IV
   if (xf >= 0.0) {
      if (m < 0.0) {
         if (y[1] >= 0.0) {
            psi_step = -rate;
         }
      } else if (m >= 0.0) {
         if ((x[1] >= 0.0) && (y[1] >= 0.0)) {
            psi_step = -rate;
         } else if (y[1] < 0.0) {
            psi_step =  rate;
         }
      }
   } else if (xf < 0.0) {
      if (m < 0.0) {
         if (x[1] < 0.0) {
            psi_step =  rate;
         }
      }
   }
}

//*********************************//
//Perform 3rd Contact Point Analysis//
//*********************************//
double  TP[] = { 0.0, 0.0, 0.0 };
psi = 0.0;
while (third_pt == 0) {
   psi = psi + psi_step*pi/180.0;
   //We already took the transpose of XP
   XP[0][0] = 1.0; XP[0][1] = 0.0;      XP[0][2] = 0.0;
   XP[1][0] = 0.0; XP[1][1] = cos(psi); XP[1][2] = -sin(psi);
   XP[2][0] = 0.0; XP[2][1] = sin(psi); XP[2][2] = cos(psi);

   RAPID_Collide(XP, TP, bolt3, R4, T4, nut3, RAPID_ALL_CONTACTS);

   //Store Tri Contact Pairs
   float (*bolt3_contact_tri)[13] = new float[RAPID_num_contacts][13];
   float (*nut3_contact_tri)[13] = new float[RAPID_num_contacts][13];
```

```
               for (qq=0; qq < RAPID_num_contacts; qq++) {
                  for (aa=0; aa <= 12; aa++) {
                     *(*(bolt3_contact_tri + qq) + aa) =
                                    *(*(pb3_tri+RAPID_contact[qq].id1) + aa);
                     *(*(nut3_contact_tri + qq) + aa) =
                                    *(*(pn3_tri+RAPID_contact[qq].id2) + aa);
                  }
               }
               //Group the triangle sets for the 3 Contact Points
               contact_group(RAPID_num_contacts,bolt3_contact_tri,nut3_contact_tri);
               delete [] bolt3_contact_tri;
               delete [] nut3_contact_tri;
            }//End of While Loop

            delete bolt3;
            delete nut3;
            delete [] pb3_tri;
            delete [] pn3_tri;

            //Reset our contact point flags for the next iteration
            contact_set = 1;
            xroll = 0.0;
            step = 0.1;
            third_pt = 0;
         }//end of IF loop that checks to see if TILT=PITCH=0
      } //PHASE
   } //PITCH
} //TILT

delete [] pb_tri;
delete [] pn_tri;
delete bolt;
delete nut;
return 0;
}

double minimum(double A, double B, double C)
{
   double min=0;

   if ((A <= B) && (A <= C)) {
      min = A;
   } else if ((B < A) && (B < C)) {
      min = B;
   } else if ((C < A) && (C < B)) {
      min = C;
   }
   return min;
}
```

```
void stl_reader(int bfacets, int nfacets, float (*pb_tri)[13], float
                            (*pn_tri)[13], RAPID_model *bolt, RAPID_model *nut)
{
   float xb[3],yb[3],zb[3],xn[3],yn[3],zn[3];
   float nxb, nyb, nzb, nxn, nyn, nzn;

   ifstream bolt_file;
   ifstream nut_file;
   bolt_file.open("bv7.stl",ios::in|ios::binary);
   nut_file.open("nv6.stl",ios::in|ios::binary);
   bolt_file.seekg(80, ios::beg);
   bolt_file.seekg(sizeof(long int), ios::cur);//Skip bfacets value
   nut_file.seekg(80, ios::beg);
   nut_file.seekg(sizeof(long int), ios::cur);//Skip nfacets value
   ////////////////////////////
   // Bolt Facet Input Area //
   ////////////////////////////
   bolt->BeginModel();
   for (int a=0; a <= (bfacets-1); a++) {
      bolt_file.read((char*)&nxb, sizeof(float));
      bolt_file.read((char*)&nyb, sizeof(float));
      bolt_file.read((char*)&nzb, sizeof(float));

      for (int b=0; b <= 2; b++) {
         bolt_file.read((char*)&xb[b], sizeof(float));
         bolt_file.read((char*)&yb[b], sizeof(float));
         bolt_file.read((char*)&zb[b], sizeof(float));
      }
      bolt_file.seekg(2, ios::cur); // Blank read the next 2 bytes so
                                    // we can skip to the next facet
      if ((zb[0] <= 0.0375) && (zb[1] <= 0.0375) && (zb[2] <= 0.0375)) {
         //Each pX[3] is a vertex
         double p0[3] = { xb[0], yb[0], zb[0] };
         double p1[3] = { xb[1], yb[1], zb[1] };
         double p2[3] = { xb[2], yb[2], zb[2] };

         bolt->AddTri(p0, p1, p2, bcount);    // Adding facet to RAPID

         *(*(pb_tri+bcount)+(0)) = (float)bcount;// Triangle ID#
         *(*(pb_tri+bcount)+(1)) = nxb;       // Normal X
         *(*(pb_tri+bcount)+(2)) = nyb;       // Normal Y
         *(*(pb_tri+bcount)+(3)) = nzb;       // Normal Z

         int e = 3;
         for (int d=0; d <= 2; d++) {
            *(*(pb_tri+bcount)+(1+e)) = xb[d]; // *(pb_tri+a) is the row
            *(*(pb_tri+bcount)+(2+e)) = yb[d]; // and (1+e) is the column
            *(*(pb_tri+bcount)+(3+e)) = zb[d]; // initially *pb_tri points to
            e += 3;                            // row zero.
         }
         bcount++;
      }
   }

   ////////////////////////////
   // Nut Facet Input Area //
   ////////////////////////////
   nut->BeginModel();
   for (int f=0; f <= (nfacets-1); f++){
      nut_file.read((char*)&nxn, sizeof(float));
      nut_file.read((char*)&nyn, sizeof(float));
      nut_file.read((char*)&nzn, sizeof(float));
```

```
      for (int g=0; g <= 2; g++){
         nut_file.read((char*)&xn[g], sizeof(float));
         nut_file.read((char*)&yn[g], sizeof(float));
         nut_file.read((char*)&zn[g], sizeof(float));
      }
      nut_file.seekg(2, ios::cur);

      if ((zn[0] >= 0.025) && (zn[1] >= 0.025) && (zn[2] >= 0.025)) {
         double p3[3]  = { xn[0], yn[0], zn[0] };
         double p4[3]  = { xn[1], yn[1], zn[1] };
         double p5[3]  = { xn[2], yn[2], zn[2] };

         nut->AddTri(p3, p4, p5, ncount);
         *(*(pn_tri+ncount)+(0)) = (float)ncount;// Triangle ID#
         *(*(pn_tri+ncount)+(1)) = nxn;       // Normal X
         *(*(pn_tri+ncount)+(2)) = nyn;       // Normal Y
         *(*(pn_tri+ncount)+(3)) = nzn;       // Normal Z

         int j = 3;
         for (int d=0; d <= 2; d++) {
            *(*(pn_tri+ncount)+(1+j)) = xn[d];
            *(*(pn_tri+ncount)+(2+j)) = yn[d];
            *(*(pn_tri+ncount)+(3+j)) = zn[d];
            j += 3;
         }
         ncount++;
      }
   }

   cout << "done\n" << flush;
   cout << "Bolt has " << bfacets << " and the Nut has " << nfacets
        << " facets\n" << flush;
   cout << "Building hierarchies..." << flush;
   bolt->EndModel();
   nut->EndModel();
   cout << "done.\n" << flush;
   cout << bcount << " " << ncount << endl;
   bolt_file.close();
   nut_file.close();
}
```

```
double init_collision_detect(double Rb[][3], double Tb[], RAPID_model *bolt,
                             double Rn[][3], double Tn[], RAPID_model *nut, int
                             bcount, float (*pb_tri)[13])
{
   double ep=0.01, ZMIN=1.0, z1, z2, z3;
   for (int row=0; row <= bcount-1; row++) {
      z1 = *(*(pb_tri+row)+(4))*Rb[2][0] + *(*(pb_tri+row)+(5))*Rb[2][1] +
            *(*(pb_tri+row)+(6))*Rb[2][2];
      z2 = *(*(pb_tri+row)+(7))*Rb[2][0] + *(*(pb_tri+row)+(8))*Rb[2][1] +
            *(*(pb_tri+row)+(9))*Rb[2][2];
      z3 = *(*(pb_tri+row)+(10))*Rb[2][0] + *(*(pb_tri+row)+(11))*Rb[2][1] +
            *(*(pb_tri+row)+(12)) * Rb[2][2];
      if ((z1 <= z2) && (z1 <= z3) && (z1 <= ZMIN)) {
         ZMIN = z1;
      } else if ((z2 < z1) && (z2 <= z3) && (z2 <= ZMIN)) {
         ZMIN = z2;
      } else if ((z3 < z1) && (z3 < z2) && (z3 <= ZMIN)) {
         ZMIN = z3;
      }
   }
   Tb[2] = 0.050001 - ZMIN;  //Solve for T1[2] in transformation equation

   //Find translation in the pos. Z-axis that minimizes the # of contact pairs
   for (int i=0; i<=3; i++) {
      ep = ep/10.0;
      RAPID_Collide(Rb, Tb, bolt, Rn, Tn, nut, RAPID_FIRST_CONTACT);
      while (RAPID_num_contacts == 0) {
         Tb[2] = Tb[2] - ep;
         RAPID_Collide(Rb, Tb, bolt, Rn, Tn, nut, RAPID_FIRST_CONTACT);
      }
      Tb[2] = Tb[2] + ep;
   }
   Tb[2] = Tb[2] - ep;
   return Tb[2];
}
```

```
double contact_point(int counter, double R[][3], double T[], float (*bolt)[13],
                     float (*nut)[13])
{
   double  BX1,BX2,BX3,BY1,BY2,BY3,BZ1,BZ2,BZ3;
   double  NX1,NX2,NX3,NY1,NY2,NY3,NZ1,NZ2,NZ3;
   double  x1cmp,x2cmp,x3cmp,y1cmp,y2cmp,y3cmp,z1cmp,z2cmp,z3cmp;
   double  b_icrd, b_jcrd, b_kcrd,MAG1,MAG2,MAG3,MAGC1,MAGC2,MAGC3;
   double  n_icrd, n_jcrd, n_kcrd,nMAG1,nMAG2,nMAG3,nMAGC1,nMAGC2,nMAGC3;
   double  line_mag1_0,line_mag2_0,line_mag1_1,line_mag2_1,bolt_mag;
   double  lc_theta1_0,lc_theta2_0,lc_theta1_1,lc_theta2_1;
   double  Xi,Zi,Kb,Kn,Mx,My,Mz,BCX[2],BCY[2],BCZ[2],NCX[2],NCY[2],NCZ[2];
   double  XC1,YC1,ZC1,XC2,YC2,ZC2,XC3,YC3,ZC3;
   double  nXC1,nYC1,nZC1,nXC2,nYC2,nZC2,nXC3,nYC3,nZC3;
   double  n1Mx,n2Mx,n3Mx;
   double  c_theta_1,c_theta_2,c_theta_3;
   double  nc_theta_1,nc_theta_2,nc_theta_3;
   double  midpoint_x=0.0,midpoint_y=0.0,midpoint_z=0.0;
   int     num_tri=counter;//counter is the #of contact pairs

   for (int h=0; h<=counter-1; h++) {//start at 0, so we end at counter-1
      x1cmp = *(*(bolt + h) +4);
      y1cmp = *(*(bolt + h) +5);
      z1cmp = *(*(bolt + h) +6);
      x2cmp = *(*(bolt + h) +7);
      y2cmp = *(*(bolt + h) +8);
      z2cmp = *(*(bolt + h) +9);
      x3cmp = *(*(bolt + h) +10);
      y3cmp = *(*(bolt + h) +11);
      z3cmp = *(*(bolt + h) +12);
      // Convert BOLT tri coords into the NUT ref. frame
      BX1 = x1cmp*R[0][0] + y1cmp*R[0][1] + z1cmp*R[0][2] + T[0];
      BX2 = x2cmp*R[0][0] + y2cmp*R[0][1] + z2cmp*R[0][2] + T[0];
      BX3 = x3cmp*R[0][0] + y3cmp*R[0][1] + z3cmp*R[0][2] + T[0];
      BY1 = x1cmp*R[1][0] + y1cmp*R[1][1] + z1cmp*R[1][2] + T[1];
      BY2 = x2cmp*R[1][0] + y2cmp*R[1][1] + z2cmp*R[1][2] + T[1];
      BY3 = x3cmp*R[1][0] + y3cmp*R[1][1] + z3cmp*R[1][2] + T[1];
      BZ1 = x1cmp*R[2][0] + y1cmp*R[2][1] + z1cmp*R[2][2] + T[2];
      BZ2 = x2cmp*R[2][0] + y2cmp*R[2][1] + z2cmp*R[2][2] + T[2];
      BZ3 = x3cmp*R[2][0] + y3cmp*R[2][1] + z3cmp*R[2][2] + T[2];

      NX1 = *(*(nut + h) +4);
      NY1 = *(*(nut + h) +5);
      NZ1 = *(*(nut + h) +6);
      NX2 = *(*(nut + h) +7);
      NY2 = *(*(nut + h) +8);
      NZ2 = *(*(nut + h) +9);
      NX3 = *(*(nut + h) +10);
      NY3 = *(*(nut + h) +11);
      NZ3 = *(*(nut + h) +12);
      //Obtain the normal vector to the Bolt and Nut via Cross Product
      b_icrd = (BY2-BY1)*(BZ3-BZ1) - (BY3-BY1)*(BZ2-BZ1);
      b_jcrd = (BX3-BX1)*(BZ2-BZ1) - (BX2-BX1)*(BZ3-BZ1);
      b_kcrd = (BX2-BX1)*(BY3-BY1) - (BX3-BX1)*(BY2-BY1);

      n_icrd = (NY2-NY1)*(NZ3-NZ1) - (NY3-NY1)*(NZ2-NZ1);
      n_jcrd = (NX3-NX1)*(NZ2-NZ1) - (NX2-NX1)*(NZ3-NZ1);
      n_kcrd = (NX2-NX1)*(NY3-NY1) - (NX3-NX1)*(NY2-NY1);
      //Some constants to make life easier
      Kb = b_icrd * BX1 + b_jcrd * BY1 + b_kcrd * BZ1;
      Kn = n_icrd * NX1 + n_jcrd * NY1 + n_kcrd * NZ1;
      //Components of the direction vector M
      Mx = b_jcrd * n_kcrd - b_kcrd * n_jcrd;
      My = b_kcrd * n_icrd - b_icrd * n_kcrd;
```

```
Mz = b_icrd * n_jcrd - b_jcrd * n_icrd;
//More constants -- yet life isn't any easier
Zi = (n_icrd*Kb - b_icrd*Kn)/My;
Xi = (b_kcrd*Kn - n_kcrd*Kb)/(n_icrd*b_kcrd - b_icrd*n_kcrd);

//The 3 contact points that occur between the edges of the BOLT tri and //the
line.  Also known as bc1, bc2 and bc3
ZC1 = ((Mx*(BZ3-BZ1))/(Mx*(BZ3-BZ1)-Mz*(BX3-BX1))) *
                      (Zi + (Mz/Mx)*(BX1-Xi-BZ1*(BX3-BX1)/(BZ3-BZ1)));
YC1 = BY1 + (BY3-BY1)*(ZC1-BZ1)/(BZ3-BZ1);
XC1 = BX1 + (BX3-BX1)*(ZC1-BZ1)/(BZ3-BZ1);

ZC2 = ((Mx*(BZ3-BZ2))/(Mx*(BZ3-BZ2)-Mz*(BX3-BX2))) *
                      (Zi + (Mz/Mx)*(BX2-Xi-BZ2*(BX3-BX2)/(BZ3-BZ2)));
YC2 = BY2 + (BY3-BY2)*(ZC2-BZ2)/(BZ3-BZ2);
XC2 = BX2 + (BX3-BX2)*(ZC2-BZ2)/(BZ3-BZ2);

ZC3 = ((Mx*(BZ2-BZ1))/(Mx*(BZ2-BZ1)-Mz*(BX2-BX1))) *
                      (Zi + (Mz/Mx)*(BX1-Xi-BZ1*(BX2-BX1)/(BZ2-BZ1)));
YC3 = BY1 + (BY2-BY1)*(ZC3-BZ1)/(BZ2-BZ1);
XC3 = BX1 + (BX2-BX1)*(ZC3-BZ1)/(BZ2-BZ1);
//Magnitudes of the vectors that make up each side of the bolt tri
MAG1 = sqrt(pow((BX3-BX1),2) + pow((BY3-BY1),2) + pow((BZ3-BZ1),2));
MAG2 = sqrt(pow((BX3-BX2),2) + pow((BY3-BY2),2) + pow((BZ3-BZ2),2));
MAG3 = sqrt(pow((BX2-BX1),2) + pow((BY2-BY1),2) + pow((BZ2-BZ1),2));
//Magnitudes of the vectors parallel to the side of the bolt tri
//but drawn to the intersection point, be it bc1, bc2, or bc3
MAGC1 = sqrt(pow((XC1-BX1),2) + pow((YC1-BY1),2) + pow((ZC1-BZ1),2));
MAGC2 = sqrt(pow((XC2-BX2),2) + pow((YC2-BY2),2) + pow((ZC2-BZ2),2));
MAGC3 = sqrt(pow((XC3-BX1),2) + pow((YC3-BY1),2) + pow((ZC3-BZ1),2));
//The angle between the side of the bolt tri and the intersection point
//If c_theta_* is zero, the intersection point occurs in the same
//direction as the vector which defined the side of the BOLT tri
c_theta_1 = ((BX3-BX1)*(XC1-BX1)+(BY3-BY1)*(YC1-BY1)+(BZ3-BZ1)*(ZC1-BZ1))
           / (MAG1*MAGC1);
c_theta_2 = ((BX3-BX2)*(XC2-BX2)+(BY3-BY2)*(YC2-BY2)+(BZ3-BZ2)*(ZC2-BZ2))
           / (MAG2*MAGC2);
c_theta_3 = ((BX2-BX1)*(XC3-BX1)+(BY2-BY1)*(YC3-BY1)+(BZ2-BZ1)*(ZC3-BZ1))
           / (MAG3*MAGC3);

//The intersection point may not be on the boundary of the BOLT tri
int i=0, line_check=0;
//Check magnitudes of the vectors to isolate the two intersection points.
//Only two points are valid, so line check will be equal to at least 1
//if there is a numerical error, line_check is incremented past 1 and this
//triangle is dropped from the list of contact pairs.
if ((c_theta_1 >= 0.0) && (MAG1 >= MAGC1)) {
   BCX[i] = XC1;
   BCY[i] = YC1;
   BCZ[i] = ZC1;
   i++;
} else if ((c_theta_1 < 0.0) || (MAG1 < MAGC1)) {
   line_check++;
}
if ((c_theta_2 >= 0.0) && (MAG2 >= MAGC2)) {
   BCX[i] = XC2;
   BCY[i] = YC2;
   BCZ[i] = ZC2;
   i++;
} else if ((c_theta_2 < 0.0) || (MAG2 < MAGC2)) {
   line_check++;
}
if ((c_theta_3 >= 0.0) && (MAG3 >= MAGC3)) {
```

```
   BCX[i] = XC3;
   BCY[i] = YC3;
   BCZ[i] = ZC3;
   i++;
} else if ((c_theta_3 < 0.0) || (MAG3 < MAGC3)) {
   line_check++;
}

if (line_check >= 2) {
   num_tri = num_tri - 1;
} else if (line_check < 2) {
   // NUT REDUCTION //
   if (fabs(Mz) >= 1e-17) {
      //We had to check the Z-component of the nut tri, because if the //contact
      occurs at the surface, NZ3-NZ1=0 and we will wind up
      //dividing by zero
      if (fabs(NZ3-NZ1) <= 1.0e-17) {
         nZC1 = NZ1;
         nXC1 = Xi + (Mx/Mz)*(NZ1-Zi);
         nYC1 = (1.0/(Mz*(NX3-NX1)))*(Mz*(NX3-NX1)*NY1 + (NY3-NY1)*
                                        (Mz*(Xi-NX1)+Mx*(NZ1-Zi)));
      } else if (fabs(NZ3-NZ1) > 1.0e-17) {
         nZC1 = ((Mx*(NZ3-NZ1))/(Mx*(NZ3-NZ1)-Mz*(NX3-NX1)))*(Zi + (Mz/Mx)
                                  *(NX1-Xi-NZ1*(NX3-NX1)/(NZ3-NZ1)));
         nYC1 = NY1 + (NY3-NY1)*(nZC1-NZ1)/(NZ3-NZ1);
         nXC1 = NX1 + (NX3-NX1)*(nZC1-NZ1)/(NZ3-NZ1);
      }
      if (fabs(NZ3-NZ2) <= 1.0e-17) {
         nZC2 = NZ2;
         nXC2 = Xi + (Mx/Mz)*(NZ2-Zi);
         nYC2 = (1.0/(Mz*(NX3-NX2)))*(Mz*(NX3-NX2)*NY2 + (NY3-NY2)*
                                        (Mz*(Xi-NX2)+Mx*(NZ2-Zi)));
      } else if (fabs(NZ3-NZ2) > 1.0e-17) {
         nZC2 = ((Mx*(NZ3-NZ2))/(Mx*(NZ3-NZ2)-Mz*(NX3-NX2)))*(Zi + (Mz/Mx)
                                  *(NX2-Xi-NZ2*(NX3-NX2)/(NZ3-NZ2)));
         nYC2 = NY2 + (NY3-NY2)*(nZC2-NZ2)/(NZ3-NZ2);
         nXC2 = NX2 + (NX3-NX2)*(nZC2-NZ2)/(NZ3-NZ2);
      }
      if (fabs(NZ2-NZ1) <= 1.0e-17) {
         nZC3 = NZ1;
         nXC3 = Xi + (Mx/Mz)*(NZ1-Zi);
         nYC3 = (1.0/(Mz*(NX2-NX1)))*(Mz*(NX2-NX1)*NY1 + (NY2-NY1)*
                                           (Mz*(Xi-NX1)+Mx*(NZ1-Zi)));
      } else if (fabs(NZ2-NZ1) > 1.0e-17) {
         nZC3 = ((Mx*(NZ2-NZ1))/(Mx*(NZ2-NZ1)-Mz*(NX2-NX1)))*(Zi + (Mz/Mx)
                                     *(NX1-Xi-NZ1*(NX2-NX1)/(NZ2-NZ1)));
         nYC3 = NY1 + (NY2-NY1)*(nZC3-NZ1)/(NZ2-NZ1);
         nXC3 = NX1 + (NX2-NX1)*(nZC3-NZ1)/(NZ2-NZ1);
      }
   } else if (fabs(Mz) < 1e-17) {
      n1Mx = (NX3-NX1);
      if (fabs(n1Mx) <= 1.0e-17) {
         nYC1 = (My/(My*(NX3-NX1)-Mx*(NY3-NY1)))*(NY1*(NX3-NX1)+(NY3-NY1)
                                                              *(Xi-NX1));
         nXC1 = Xi + (Mx/My)*nYC1;
         nZC1 = NZ1;
      } else if (fabs(n1Mx) > 1.0e-17) {
         nYC1 = (My/(My*(NX3-NX1)-Mx*(NY3-NY1)))*(NY1*(NX3-NX1)+(NY3-NY1)
                                                              *(Xi-NX1));
         nXC1 = Xi + (Mx/My)*nYC1;
         nZC1 = NZ1 + ((NZ3-NZ1)/(NX3-NX1))*(nXC1-NX1);
      }
      n2Mx = (NX3-NX2);
```

```
   if (fabs(n2Mx) <= 1.0e-17) {
      nYC2 = (My/(My*(NX3-NX2)-Mx*(NY3-NY2)))*(NY2*(NX3-NX2)+(NY3-NY2)
                                                          *(Xi-NX2));
      nXC2 = Xi + (Mx/My)*nYC2;
      nZC2 = NZ2;
   } else if (fabs(n2Mx) > 1.0e-17) {
      nYC2 = (My/(My*(NX3-NX2)-Mx*(NY3-NY2)))*(NY2*(NX3-NX2)+(NY3-NY2)
                                                          *(Xi-NX2));
      nXC2 = Xi + (Mx/My)*nYC2;
      nZC2 = NZ2 + ((NZ3-NZ2)/(NX3-NX2))*(nXC2-NX2);
   }
   n3Mx = (NX2-NX1);
   if (fabs(n3Mx) <= 1.0e-17) {
      nYC3 = (My/(My*(NX2-NX1)-Mx*(NY2-NY1)))*(NY1*(NX2-NX1)+(NY2-NY1)
                                                          *(Xi-NX1));
      nXC3 = Xi + (Mx/My)*nYC3;
      nZC3 = NZ1;
   } else if (fabs(n3Mx) > 1.0e-17) {
      nYC3 = (My/(My*(NX2-NX1)-Mx*(NY2-NY1)))*(NY1*(NX2-NX1)+(NY2-NY1)
                                                          *(Xi-NX1));
      nXC3 = Xi + (Mx/My)*nYC3;
      nZC3 = NZ1 + ((NZ2-NZ1)/(NX2-NX1))*(nXC3-NX1);
   }
}
//Magnitudes of the vectors that make up each side of the NUT tri
nMAG1 = sqrt(pow((NX3-NX1),2) + pow((NY3-NY1),2) + pow((NZ3-NZ1),2));
nMAG2 = sqrt(pow((NX3-NX2),2) + pow((NY3-NY2),2) + pow((NZ3-NZ2),2));
nMAG3 = sqrt(pow((NX2-NX1),2) + pow((NY2-NY1),2) + pow((NZ2-NZ1),2));
//Magnitude of the vectors parallel to the side of the NUT tri but
//drawn to the intersection point
nMAGC1 = sqrt(pow((nXC1-NX1),2) + pow((nYC1-NY1),2) +
          pow((nZC1-NZ1),2));
nMAGC2 = sqrt(pow((nXC2-NX2),2) + pow((nYC2-NY2),2) +
          pow((nZC2-NZ2),2));
nMAGC3 = sqrt(pow((nXC3-NX1),2) + pow((nYC3-NY1),2) +
          pow((nZC3-NZ1),2));
//Angles that locate the orientation of the intersection point and the
//vector which defines the side of the NUT tri (same as BOLT analysis).
nc_theta_1 = ((NX3-NX1)*(nXC1-NX1) + (NY3-NY1)*(nYC1-NY1) + (NZ3-NZ1)
                                          *(nZC1-NZ1))/(nMAG1*nMAGC1);
nc_theta_2 = ((NX3-NX2)*(nXC2-NX2) + (NY3-NY2)*(nYC2-NY2) + (NZ3-NZ2)
                                          *(nZC2-NZ2))/(nMAG2*nMAGC2);
nc_theta_3 = ((NX2-NX1)*(nXC3-NX1) + (NY2-NY1)*(nYC3-NY1) + (NZ2-NZ1)
                                          *(nZC3-NZ1))/(nMAG3*nMAGC3);

int j=0;
//Only two points will be valid, thus one of these statements will fail
if ((nc_theta_1 >= 0.0) && (nMAG1 >= nMAGC1)) {
   NCX[j] = nXC1;
   NCY[j] = nYC1;
   NCZ[j] = nZC1;
   j++;
}
if ((nc_theta_2 >= 0.0) && (nMAG2 >= nMAGC2)) {
   NCX[j] = nXC2;
   NCY[j] = nYC2;
   NCZ[j] = nZC2;
   j++;
}
if ((nc_theta_3 >= 0.0) && (nMAG3 >= nMAGC3)) {
   NCX[j] = nXC3;
   NCY[j] = nYC3;
   NCZ[j] = nZC3;
```

```
   j++;
}

//Magnitude of the line segment bounded by the sides of the BOLT tri
bolt_mag = sqrt(pow((BCX[1]-BCX[0]),2) + pow((BCY[1]-BCY[0]),2) +
                              pow((BCZ[1]-BCZ[0]),2));
//Defines the mag. of the vector drawn from bc1 to nc1
line_mag1_0 = sqrt(pow((NCX[0]-BCX[0]),2) + pow((NCY[0]-BCY[0]),2) +
                                pow((NCZ[0]-BCZ[0]),2));
//Defines the mag. of the vector drawn from bc1 to nc2
line_mag2_0 = sqrt(pow((NCX[1]-BCX[0]),2) + pow((NCY[1]-BCY[0]),2) +
                                pow((NCZ[1]-BCZ[0]),2));
//Angle between the vector from bc1 to nc1 and vector from bc1 to bc2
lc_theta1_0 = ((BCX[1]-BCX[0])*(NCX[0]-BCX[0]) + (BCY[1]-BCY[0])
                          *(NCY[0]-BCY[0]) + (BCZ[1]-BCZ[0])
                       *(NCZ[0]-BCZ[0]))/(bolt_mag*line_mag1_0);
//Angle between the vector from bc1 to nc2 and vector from bc1 to bc2
lc_theta2_0 = ((BCX[1]-BCX[0])*(NCX[1]-BCX[0]) + (BCY[1]-BCY[0])
                           *(NCY[1]-BCY[0]) + (BCZ[1]-BCZ[0])
                          *(NCZ[1]-BCZ[0]))/(bolt_mag*line_mag2_0);
//Defines mag of the line segment drawn from bc2 to nc1
line_mag1_1 = sqrt(pow((NCX[0]-BCX[1]),2) + pow((NCY[0]-BCY[1]),2) +
                                             pow((NCZ[0]-BCZ[1]),2));
//Defines mag of the line segment drawn from bc2 to nc2
line_mag2_1 = sqrt(pow((NCX[1]-BCX[1]),2) + pow((NCY[1]-BCY[1]),2) +
                                             pow((NCZ[1]-BCZ[1]),2));
//Angle between the vector from bc2 to nc1 and vector from bc1 to bc2
lc_theta1_1 = ((BCX[0]-BCX[1])*(NCX[0]-BCX[1]) + (BCY[0]-BCY[1])
                            *(NCY[0]-BCY[1]) + (BCZ[0]-BCZ[1])
                           *(NCZ[0]-BCZ[1]))/(bolt_mag*line_mag1_1);
//Angle between the vector from bc2 to nc2 and vector from bc1 to bc2
lc_theta2_1 = ((BCX[0]-BCX[1])*(NCX[1]-BCX[1]) + (BCY[0]-BCY[1])
                            *(NCY[1]-BCY[1]) + (BCZ[0]-BCZ[1])
                           *(NCZ[1]-BCZ[1]))/(bolt_mag*line_mag2_1);

//The following minimizes the length of the line segment
//First up is the Type I BOLT/NUT contact configuration
if ((lc_theta1_0 <= 0.0) && (lc_theta2_0 > 0.0)) {
   if ((bolt_mag >= line_mag2_0)) {
      midpoint_x = midpoint_x + (NCX[1] + BCX[0])/2.0;
      midpoint_y = midpoint_y + (NCY[1] + BCY[0])/2.0;
      midpoint_z = midpoint_z + (NCZ[1] + BCZ[0])/2.0;
   }
} else if ((lc_theta2_0 <= 0.0) && (lc_theta1_0 > 0.0)) {
   if ((bolt_mag >= line_mag1_0)) {
      midpoint_x = midpoint_x + (NCX[0] + BCX[0])/2.0;
      midpoint_y = midpoint_y + (NCY[0] + BCY[0])/2.0;
      midpoint_z = midpoint_z + (NCZ[0] + BCZ[0])/2.0;
   }
} else if ((lc_theta1_1 <= 0.0) && (lc_theta2_1 > 0.0)) {
   if ((bolt_mag >= line_mag2_1)) {
      midpoint_x = midpoint_x + (NCX[1] + BCX[1])/2.0;
      midpoint_y = midpoint_y + (NCY[1] + BCY[1])/2.0;
      midpoint_z = midpoint_z + (NCZ[1] + BCZ[1])/2.0;
   }
} else if ((lc_theta2_1 <= 0.0) && (lc_theta1_1 > 0.0)) {
   if ((bolt_mag >= line_mag1_1)) {
      midpoint_x = midpoint_x + (NCX[0] + BCX[1])/2.0;
      midpoint_y = midpoint_y + (NCY[0] + BCY[1])/2.0;
      midpoint_z = midpoint_z + (NCZ[0] + BCZ[1])/2.0;
   }
}
//Type II BOLT/NUT contact configuration
```

```
        if ((lc_theta1_0 > 0.0) && (lc_theta2_0 > 0.0) &&
                    (bolt_mag >= line_mag1_0) && (bolt_mag >= line_mag2_0)) {
           midpoint_x = midpoint_x + (NCX[0] + NCX[1])/2.0;
           midpoint_y = midpoint_y + (NCY[0] + NCY[1])/2.0;
           midpoint_z = midpoint_z + (NCZ[0] + NCZ[1])/2.0;
        }
        //Type III BOLT/NUT contact configuration
        if ((lc_theta1_0 < 0.0) && (lc_theta2_0 > 0.0) &&
                              (lc_theta1_1 > 0.0) && (lc_theta2_1 < 0.0)) {
           midpoint_x = midpoint_x + (BCX[0] + BCX[1])/2.0;
           midpoint_y = midpoint_y + (BCY[0] + BCY[1])/2.0;
           midpoint_z = midpoint_z + (BCZ[0] + BCZ[1])/2.0;
        } else if ((lc_theta1_0 > 0.0) && (lc_theta2_0 < 0.0) &&
                              (lc_theta1_1 < 0.0) && (lc_theta2_1 > 0.0)) {
           midpoint_x = midpoint_x + (BCX[0] + BCX[1])/2.0;
           midpoint_y = midpoint_y + (BCY[0] + BCY[1])/2.0;
           midpoint_z = midpoint_z + (BCZ[0] + BCZ[1])/2.0;
        }
        //Type IV BOLT/NUT contact configuration
        if ((lc_theta1_0 > 0.0) && (lc_theta2_0 > 0.0) &&
                              (lc_theta1_1 < 0.0) && (lc_theta2_1 < 0.0)) {
           midpoint_x = midpoint_x + (BCX[0] + BCX[1])/2.0;
           midpoint_y = midpoint_y + (BCY[0] + BCY[1])/2.0;
           midpoint_z = midpoint_z + (BCZ[0] + BCZ[1])/2.0;
        } else if ((lc_theta1_0 < 0.0) && (lc_theta2_0 < 0.0) &&
                              (lc_theta1_1 > 0.0) && (lc_theta2_1 > 0.0)) {
           midpoint_x = midpoint_x + (BCX[0] + BCX[1])/2.0;
           midpoint_y = midpoint_y + (BCY[0] + BCY[1])/2.0;
           midpoint_z = midpoint_z + (BCZ[0] + BCZ[1])/2.0;
        }
      }// end of else if in line_check
    } // end of for loop
    //Store the midpoint of the line segment in memory
    midpoint[0] = midpoint_x/(double)num_tri;
    midpoint[1] = midpoint_y/(double)num_tri;
    midpoint[2] = midpoint_z/(double)num_tri;
    return 0;
}
```

```
double bolt_surf(double R1[][3], double xc, double yc, double zc, double zb,
                 int b)
{
   double  b_max = 0.12445, rr=0.005, crr=0.0025;//MAX BOLT radius
   double  b_min = 0.12445 - 5.0*(0.05)*sqrt(3.0)/16.0;//MIN BOLT radius
   double  K = sqrt(3.0)*0.05/(2.0*pi), L = 0.05/(2.0*pi);//Constants
   double  Tb_i, Tb_j, Tb_k, mag_Tb;//Vector Components of BOLT tangent vector
   double  theta, angle, ang_ini, theta_ini=0.0;

   //First, convert contact point into the BOLT reference frame
   loc_xc = R1[0][0]*xc + R1[1][0]*yc + R1[2][0]*zc - R1[2][0]*zb;
   loc_yc = R1[0][1]*xc + R1[1][1]*yc + R1[2][1]*zc - R1[2][1]*zb;
   loc_zc = R1[0][2]*xc + R1[1][2]*yc + R1[2][2]*zc - R1[2][2]*zb;

   //Compute the angular location of the contact
   if ((loc_xc >= 0.0) && (loc_yc >= 0.0)) {       // Quadrant I
      angle = atan(loc_yc/loc_xc);
   } else if ((loc_xc < 0.0) && (loc_yc >= 0.0)) {    // Quadrant II
      angle = pi + atan(loc_yc/loc_xc);
   } else if ((loc_xc < 0.0) && (loc_yc < 0.0)) {     // Quadrant III
      angle = pi + atan(loc_yc/loc_xc);
   } else if ((loc_xc >= 0.0) && (loc_yc < 0.0)) {    // Quadrant IV
      angle = 2.0*pi + atan(loc_yc/loc_xc);
   }
   if (b == 0) { // Contact occurs above the base of the bolt
      cout << "No BASE triangles" << endl;
      //Rotate cross-section of the BOLT counter-clockwise
      //because we are going up a right-handed thread
      angle = angle + loc_zc*2.0*pi/0.05;
      if (angle > 2.0*pi) {
         angle = angle - 2.0*pi;
      }
      Tb_k = 0.05/(2.0*pi);
   } else if (b != 0) {//Contact occurs at an edge with includes the Base
      Tb_k = 0.0;
   }
   //Welcome to the tangent vector determination based on the cross sections
   //defined in Chapter 2
   theta = angle*180.0/pi;
   if ((theta >= 0.0) && (theta <= 34.14202146)) {//+ Flank
      ang_ini = 62.76952127*pi/180.0;
      Tb_i = -K*cos(angle) - (b_max - 0.5*crr - K*(angle+ang_ini))*sin(angle);
      Tb_j = -K*sin(angle) + (b_max - 0.5*crr - K*(angle+ang_ini))*cos(angle);
   } else if ((theta > 34.14202146) && (theta <= 65.31893600)) {//+ Root Fillet
      ang_ini = 34.14202146*pi/180.0;
      Tb_i = -(L*(sqrt(3)*rr/2.0 -L*(angle-ang_ini))*cos(angle))/sqrt(pow(rr,2)-
              pow((sqrt(3)*rr/2.0 -L*(angle-ang_ini)),2)) - (b_min + rr –
              sqrt(pow(rr,2)-pow((sqrt(3)*rr/2.0 -L*(angle-
               ang_ini)),2)))*sin(angle);
      Tb_j = -(L*(sqrt(3)*rr/2.0 -L*(angle-ang_ini))*sin(angle))/sqrt(pow(rr,2)-
              pow((sqrt(3)*rr/2.0 -L*(angle-ang_ini)),2)) + (b_min + rr –
              sqrt(pow(rr,2)-pow((sqrt(3)*rr/2.0 -L*(angle-
              ang_ini)),2)))*cos(angle);
   } else if ((theta > 65.31893600) && (theta <= 113.74971662)) {// Root
      Tb_i = -b_min*sin(angle);
      Tb_j =  b_min*cos(angle);
   } else if ((theta > 113.74971662) && (theta <= 144.92663116)) {//- RF
      ang_ini = 113.74971662*pi/180.0;
      Tb_i = (pow(L,2)*(angle-ang_ini)*cos(angle))/sqrt(pow(rr,2)
              -pow(L,2)*pow((angle-ang_ini),2)) - (b_min + rr - sqrt(pow(rr,2) –
              pow(L,2)*pow((angle-ang_ini),2)))*sin(angle);
      Tb_j = (pow(L,2)*(angle-ang_ini)*sin(angle))/sqrt(pow(rr,2)-
              pow(L,2)*pow((angle-ang_ini),2)) + (b_min + rr - sqrt(pow(rr,2) –
```

```
            pow(L,2)*pow((angle-ang_ini),2)))*cos(angle);
   } else if ((theta > 144.92663116) && (theta <= 241.83817389)) {//- Flank
      ang_ini = 144.92663116*pi/180.0;
      Tb_i = K*cos(angle) - (b_min + 0.5*rr + K*(angle-ang_ini))*sin(angle);
      Tb_j = K*sin(angle) + (b_min + 0.5*rr + K*(angle-ang_ini))*cos(angle);
   } else if ((theta > 241.83817389) && (theta <= 257.42663116)) {//- CF
      ang_ini = 241.83817389*pi/180.0;
      Tb_i = (L*(sqrt(3)*crr/2.0 -L*(angle-ang_ini))*cos(angle))
             /sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 -L*(angle-ang_ini)),2)) –
            (b_max - crr + sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 -L*(angle-
            ang_ini)),2)))*sin(angle);
      Tb_j = (L*(sqrt(3)*crr/2.0 -L*(angle-ang_ini))*sin(angle))
            /sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 -L*(angle-ang_ini)),2)) +
           (b_max - crr + sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 -L*(angle-
           ang_ini)),2)))*cos(angle);
   } else if ((theta > 257.42663116) && (theta <= 281.64202146)) {// Crest
      Tb_i = -b_max*sin(angle);
      Tb_j =  b_max*cos(angle);
   } else if ((theta > 281.64202146) && (theta <= 297.23047873)) {//+ CF
      ang_ini = 281.64202146*pi/180.0;
      Tb_i = -(pow(L,2)*(angle-ang_ini)*cos(angle))/sqrt(pow(crr,2)
             -pow(L,2)*pow((angle-ang_ini),2)) - (b_max - crr + sqrt(pow(crr,2)
             - pow(L,2)*pow((angle-ang_ini),2)))*sin(angle);
      Tb_j = -(pow(L,2)*(angle-ang_ini)*sin(angle))/sqrt(pow(crr,2)-
             pow(L,2)*pow((angle-ang_ini),2)) + (b_max - crr + sqrt(pow(crr,2)
             - pow(L,2)*pow((angle-ang_ini),2)))*cos(angle);
   } else if ((theta > 297.23047873) && (theta <= 360.0)) {//+ Flank
      ang_ini = 297.23047873*pi/180.0;
      Tb_i = -K*cos(angle) - (b_max - 0.5*crr - K*(angle-ang_ini))*sin(angle);
      Tb_j = -K*sin(angle) + (b_max - 0.5*crr - K*(angle-ang_ini))*cos(angle);
   }
   mag_Tb = sqrt(pow(Tb_i,2) + pow(Tb_j,2) + pow(Tb_k,2));
   //Normalized Tangent Vector Components, a.k.a. direction cosines
   BMt_i = Tb_i/mag_Tb;
   BMt_j = Tb_j/mag_Tb;
   BMt_k = Tb_k/mag_Tb;
   //Convert Direction Cosines to FIXED frame
   BWt_i = R1[0][0]*BMt_i + R1[0][1]*BMt_j + R1[0][2]*BMt_k;
   BWt_j = R1[1][0]*BMt_i + R1[1][1]*BMt_j + R1[1][2]*BMt_k;
   BWt_k = R1[2][0]*BMt_i + R1[2][1]*BMt_j + R1[2][2]*BMt_k;
   return 0;
}
```

```
double nut_surf(double xc, double yc, double zc, int ns)
{
   double  n_max = 0.125, rr=0.0025, crr=0.005;//MAX NUT radius
   double  n_min = 0.125 - 5.0*(0.05)*sqrt(3.0)/16.0;//MIN NUT radius
   double  K = sqrt(3.0)*0.05/(2.0*pi), L = 0.05/(2.0*pi);//Constants
   double  Tn_i, Tn_j, Tn_k, mag_Tn;//NUT Tangent Vector Components
   double  theta, nang, ang_ini;

   //Compute the angular location of the contact as viewed in the NUT frame
   if ((xc >= 0.0) && (yc >= 0.0)) {       // Quadrant I
      nang = atan(yc/xc);
   } else if ((xc < 0.0) && (yc >= 0.0)) {    // Quadrant II
      nang = pi + atan(yc/xc);
   } else if ((xc < 0.0) && (yc < 0.0)) {     // Quadrant III
      nang = pi + atan(yc/xc);
   } else if ((xc >= 0.0) && (yc < 0.0)) {    // Quadrant IV
      nang = 2.0*pi + atan(yc/xc);
   }
   if (ns == 0) {//Contact occurs below the surface of the NUT
      cout << "No SURF triangles" << endl;
      //Rotate cross section of the NUT clockwise
      //because we are going down a right-handed thread
      nang = nang - (0.05-zc)*2.0*pi/0.05;
      if (nang < 0.0) {
         nang = nang + 2.0*pi;
      }
      Tn_k = 0.05/(2.0*pi);
   } else if (ns != 0) {//Contact occurs at an edge OR on the surface of the NUT
      Tn_k = 0.0;
   }
   //Equations for the components of the vector tangent to the NUT
   theta = nang*180.0/pi;
   if ((theta >= 0.0) && (theta <= 60.27536811)) {//- Flank
      ang_ini = 36.63617462*pi/180.0;
      Tn_i = K*cos(nang) - (n_min + 0.5*crr + K*(nang+ang_ini))*sin(nang);
      Tn_j = K*sin(nang) + (n_min + 0.5*crr + K*(nang+ang_ini))*cos(nang);
   } else if ((theta > 60.27536811) && (theta <= 75.86382538)) {//- Root Fillet
      ang_ini = 60.27536811*pi/180.0;
      Tn_i = (L*(sqrt(3)*rr/2.0 - L*(nang-ang_ini))*cos(nang))/sqrt(pow(rr,2)-
              pow((sqrt(3)*rr/2.0 -L*(nang-ang_ini)),2)) - (n_max - rr +
              sqrt(pow(rr,2)-pow((sqrt(3)*rr/2.0 -L*(nang-
              ang_ini)),2)))*sin(nang);
      Tn_j = (L*(sqrt(3)*rr/2.0 - L*(nang-ang_ini))*sin(nang))/sqrt(pow(rr,2)-
              pow((sqrt(3)*rr/2.0 -L*(nang-ang_ini)),2)) + (n_max - rr +
              sqrt(pow(rr,2)-pow((sqrt(3)*rr/2.0 -L*(nang-
              ang_ini)),2)))*cos(nang);
   } else if ((theta > 75.86382538) && (theta <= 100.07921568)) {// Root
      Tn_i = -n_max*sin(nang);
      Tn_j =  n_max*cos(nang);
   } else if ((theta > 100.07921568) && (theta <= 115.66767295)) {//+ RF
      ang_ini = 100.07921568*pi/180.0;
      Tn_i = -(pow(L,2)*(nang-ang_ini)*cos(nang))/sqrt(pow(rr,2)-
              pow(L,2)*pow((nang-ang_ini),2)) - (n_max - rr + sqrt(pow(rr,2)-
              pow(L,2)*pow((nang-ang_ini),2)))*sin(nang);
      Tn_j = -(pow(L,2)*(nang-ang_ini)*sin(nang))/sqrt(pow(rr,2)-
              pow(L,2)*pow((nang-ang_ini),2)) + (n_max - rr + sqrt(pow(rr,2)-
              pow(L,2)*pow((nang-ang_ini),2)))*cos(nang);
   } else if ((theta > 115.66767295) && (theta <= 212.57921568)) {//+ Flank
      ang_ini = 115.66767295*pi/180.0;
      Tn_i = -K*cos(nang) - (n_max - 0.5*rr - K*(nang-ang_ini))*sin(nang);
      Tn_j = -K*sin(nang) + (n_max - 0.5*rr - K*(nang-ang_ini))*cos(nang);
   } else if ((theta > 212.57921568) && (theta <= 243.75613022)) {//+ CF
      ang_ini = 212.57921568*pi/180.0;
```

```
      Tn_i = -(L*(sqrt(3)*crr/2.0 -L*(nang-ang_ini))*cos(nang))/sqrt(pow(crr,2)-
              pow((sqrt(3)*crr/2.0 -L*(nang-ang_ini)),2)) - (n_min + crr –
              sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 -L*(nang-
              ang_ini)),2)))*sin(nang);
      Tn_j = -(L*(sqrt(3)*crr/2.0 -L*(nang-ang_ini))*sin(nang))/sqrt(pow(crr,2)-
              pow((sqrt(3)*crr/2.0 -L*(nang-ang_ini)),2)) + (n_min + crr –
              sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 -L*(nang-
              ang_ini)),2)))*cos(nang);
   } else if ((theta > 243.75613022) && (theta <= 292.18691084)) {// Crest
      Tn_i = -n_min*sin(nang);
      Tn_j =  n_min*cos(nang);
   } else if ((theta > 292.18691084) && (theta <= 323.36382538)) {//-CF
      ang_ini = 292.18691084*pi/180.0;
      Tn_i = (pow(L,2)*(nang-ang_ini)*cos(nang))/sqrt(pow(crr,2)
              -pow(L,2)*pow((nang-ang_ini),2)) - (n_min + crr - sqrt(pow(crr,2)
              - pow(L,2)*pow((nang-ang_ini),2)))*sin(nang);
      Tn_j = (pow(L,2)*(nang-ang_ini)*sin(nang))/sqrt(pow(crr,2)-
              pow(L,2)*pow((nang-ang_ini),2)) + (n_min + crr - sqrt(pow(crr,2) –
              pow(L,2)*pow((nang-ang_ini),2)))*cos(nang);
   } else if ((theta > 323.36382538) && (theta <= 360.0)) {//- Flank
      ang_ini = 323.36382538*pi/180.0;
      Tn_i = K*cos(nang) - (n_min + 0.5*crr + K*(nang-ang_ini))*sin(nang);
      Tn_j = K*sin(nang) + (n_min + 0.5*crr + K*(nang-ang_ini))*cos(nang);
   }
   mag_Tn = sqrt(pow(Tn_i,2) + pow(Tn_j,2) + pow(Tn_k,2));
   //Normalized components of the vector tangent to the NUT edge
   NWt_i = Tn_i/mag_Tn;
   NWt_j = Tn_j/mag_Tn;
   NWt_k = Tn_k/mag_Tn;
   return 0;
}
```

```
void second_model_input(double R1[][3], double T1[], RAPID_model *bolt2, float
                        (*pb2_tri)[13], float (*pb_tri)[13], RAPID_model *nut2,
                        float (*pn2_tri)[13], float (*pn_tri)[13])
{
   int     mm,nn,blt_cnt=0,nut_cnt=0;
   float fixed_ID;
   double  xf_b1,yf_b1,zf_b1,xf_n2,yf_n2,zf_n2;
   double  xt_b1[3],yt_b1[3],zt_b1[3],xt_n2[3],yt_n2[3],zt_n2[3];
   double  theta_X, theta_Y, theta_Z;

   //BOLT2 Facet Input Area
   bolt2->BeginModel();
   for (mm=0; mm<=bcount-1; mm++) {
      //Convert BOLT dir cosines from the BOLT (b) frame to the FIXED (F) frame
      fixed_ID = *(*(pb_tri+blt_cnt)+(0));
      theta_X  = *(*(pb_tri+blt_cnt)+(1)) * R1[0][0] + *(*(pb_tri+blt_cnt)+(2))
                 * R1[0][1] + *(*(pb_tri+blt_cnt)+(3)) * R1[0][2];
      theta_Y  = *(*(pb_tri+blt_cnt)+(1)) * R1[1][0] + *(*(pb_tri+blt_cnt)+(2))
                 * R1[1][1] + *(*(pb_tri+blt_cnt)+(3)) * R1[1][2];
      theta_Z  = *(*(pb_tri+blt_cnt)+(1)) * R1[2][0] + *(*(pb_tri+blt_cnt)+(2))
                 * R1[2][1] + *(*(pb_tri+blt_cnt)+(3)) * R1[2][2];
      //Convert BOLT dir cosines from the FIXED (F) frame to the C0 (tang) frame
      *(*(pb2_tri+blt_cnt)+(0)) = fixed_ID;
      *(*(pb2_tri+blt_cnt)+(1)) = (float)(theta_X * Q[0][0] + theta_Y * Q[0][1]
                                  + theta_Z * Q[0][2]);
      *(*(pb2_tri+blt_cnt)+(2)) = (float)(theta_X * Q[1][0] + theta_Y * Q[1][1]
                                  + theta_Z * Q[1][2]);
      *(*(pb2_tri+blt_cnt)+(3)) = (float)(theta_X * Q[2][0] + theta_Y * Q[2][1]
                                  + theta_Z * Q[2][2]);

      int ee=3;
      for (nn=0; nn<=2; nn++) {
         //Convert BOLT coords from the BOLT (b) frame to the FIXED (F) frame
         xf_b1 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R1[0][0] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R1[0][1] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R1[0][2] + T1[0];
         yf_b1 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R1[1][0] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R1[1][1] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R1[1][2] + T1[1];
         zf_b1 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R1[2][0] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R1[2][1] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R1[2][2] + T1[2];
         //Convert BOLT coords from the FIXED (F) frame to the C0 (tang) frame
         xt_b1[nn] = xf_b1 * Q[0][0] + yf_b1 * Q[0][1] + zf_b1 * Q[0][2] –
                     (Q[0][0]*U[0] + Q[0][1]*U[1] + Q[0][2]*U[2]);
         yt_b1[nn] = xf_b1 * Q[1][0] + yf_b1 * Q[1][1] + zf_b1 * Q[1][2] –
                     (Q[1][0]*U[0] + Q[1][1]*U[1] + Q[1][2]*U[2]);
         zt_b1[nn] = xf_b1 * Q[2][0] + yf_b1 * Q[2][1] + zf_b1 * Q[2][2] –
                     (Q[2][0]*U[0] + Q[2][1]*U[1] + Q[2][2]*U[2]);
         *(*(pb2_tri+blt_cnt)+(1+ee)) = (float)xt_b1[nn];
         *(*(pb2_tri+blt_cnt)+(2+ee)) = (float)yt_b1[nn];
         *(*(pb2_tri+blt_cnt)+(3+ee)) = (float)zt_b1[nn];
         ee += 3;
      }
      double b1p0[3] = { xt_b1[0], yt_b1[0], zt_b1[0] };
      double b1p1[3] = { xt_b1[1], yt_b1[1], zt_b1[1] };
      double b1p2[3] = { xt_b1[2], yt_b1[2], zt_b1[2] };
      bolt2->AddTri(b1p0, b1p1, b1p2, blt_cnt);
      blt_cnt++;
   }
   bolt2->EndModel();
```

```
   //NUT2 Facet Input Area
   double  Ntheta_X, Ntheta_Y, Ntheta_Z;
   float Nfixed_ID;
   nut2->BeginModel();
   for (mm=0; mm<=ncount-1; mm++) {
      Nfixed_ID = *(*(pn_tri+nut_cnt)+(0));
      Ntheta_X  = *(*(pn_tri+nut_cnt)+(1));
      Ntheta_Y  = *(*(pn_tri+nut_cnt)+(2));
      Ntheta_Z  = *(*(pn_tri+nut_cnt)+(3));

      *(*(pn2_tri+nut_cnt)+(0)) = Nfixed_ID;
      *(*(pn2_tri+nut_cnt)+(1)) = (float)(Ntheta_X * Q[0][0] + Ntheta_Y *
                                  Q[0][1] + Ntheta_Z * Q[0][2]);
      *(*(pn2_tri+nut_cnt)+(2)) = (float)(Ntheta_X * Q[1][0] + Ntheta_Y *
                                 Q[1][1] + Ntheta_Z * Q[1][2]);
      *(*(pn2_tri+nut_cnt)+(3)) = (float)(Ntheta_X * Q[2][0] + Ntheta_Y *
                                 Q[2][1] + Ntheta_Z * Q[2][2]);

      int ef=3;
      for (nn=0; nn<=2; nn++) {
         //NUT coords are given in the FIXED frame
         xf_n2 = *(*(pn_tri+nut_cnt)+(1+ef));
         yf_n2 = *(*(pn_tri+nut_cnt)+(2+ef));
         zf_n2 = *(*(pn_tri+nut_cnt)+(3+ef));
         //Convert coords from FIXED (f) to CO (tang) frames
         xt_n2[nn] = xf_n2 * Q[0][0] + yf_n2 * Q[0][1] + zf_n2 * Q[0][2] –
                     (Q[0][0]*U[0] + Q[0][1]*U[1] + Q[0][2]*U[2]);
         yt_n2[nn] = xf_n2 * Q[1][0] + yf_n2 * Q[1][1] + zf_n2 * Q[1][2] –
                     (Q[1][0]*U[0] + Q[1][1]*U[1] + Q[1][2]*U[2]);
         zt_n2[nn] = xf_n2 * Q[2][0] + yf_n2 * Q[2][1] + zf_n2 * Q[2][2] –
                     (Q[2][0]*U[0] + Q[2][1]*U[1] + Q[2][2]*U[2]);
         *(*(pn2_tri+nut_cnt)+(1+ef)) = (float)xt_n2[nn];
         *(*(pn2_tri+nut_cnt)+(2+ef)) = (float)yt_n2[nn];
         *(*(pn2_tri+nut_cnt)+(3+ef)) = (float)zt_n2[nn];
         ef += 3;
      }
      double n2p0[3] = { xt_n2[0], yt_n2[0], zt_n2[0] };
      double n2p1[3] = { xt_n2[1], yt_n2[1], zt_n2[1] };
      double n2p2[3] = { xt_n2[2], yt_n2[2], zt_n2[2] };
      nut2->AddTri(n2p0, n2p1, n2p2, nut_cnt);
      nut_cnt++;
   }
   nut2->EndModel();
}
```

```
double mult_contact(int num_ctacts, double R[][3], double T[], float
                    (*bolt)[13], float (*nut)[13], double R1[][3], double phi)
{  ////////////////////////////////////////////////////////////////////////
   // This is where we divide contact pairs into multiple contact points //
   ////////////////////////////////////////////////////////////////////////
   double  X1,X2,X3,Y1,Y2,Y3, fixB_thetaX, fixB_thetaY, fixB_thetaZ;
   double  BthetaX, BthetaY, BthetaZ;
   double  X_avg, Y_avg, ct_cos;
   double  ep=10.0*pi/180.0;//ep allows +/-5 deg. of error in angular location
   double  delta[1000];    // each delta is the average angle location of tri
   int     pts, cntr=0;    // cntr determines the # of non-base tri's
   int     f_row[500];     // f_row retains the row # in the facet store
   int     cycle=0;        // This keeps track of the # of contacts
   int     found=-1, k, jj, kk;
   int     FLUSH=0;        // Flag for case when BOLT is flush with NUT surface
   int     resort_req = 0; // Don't resort delta's unless you are required
   int     row_ct[6] = { 0,0,0,0,0,0 };// This guy tracks the rows of each
                           // contact location array
                           // Its value stores the most recent row number
   //bc_loc Allows for 4 contacts, 200 tri's each
   double  control[6], bc_loc[4][200][13];
   double  nc_loc[4][200][13];
   double  Xcontact[6], Ycontact[6], Zcontact[6];

   for (pts=0; pts<=num_ctacts-1; pts++) {
      //Convert dir cos of Normal vector from the C0 frame to the BOLT (b) frame
      //To determine if there are any Base tri's involved in the contact
      fixB_thetaX = *(*(bolt + pts) +1) * Q[0][0] + *(*(bolt + pts) +2) *
                    Q[1][0] + *(*(bolt + pts) +3) * Q[2][0];
      fixB_thetaY = *(*(bolt + pts) +1) * Q[0][1] + *(*(bolt + pts) +2) *
                    Q[1][1] + *(*(bolt + pts) +3) * Q[2][1];
      fixB_thetaZ = *(*(bolt + pts) +1) * Q[0][2] + *(*(bolt + pts) +2) *
                    Q[1][2] + *(*(bolt + pts) +3) * Q[2][2];
      BthetaX = fixB_thetaX * R1[0][0] + fixB_thetaY * R1[1][0] + fixB_thetaZ *
                R1[2][0];
      BthetaY = fixB_thetaX * R1[0][1] + fixB_thetaY * R1[1][1] + fixB_thetaZ *
                R1[2][1];
      BthetaZ = fixB_thetaX * R1[0][2] + fixB_thetaY * R1[1][2] + fixB_thetaZ *
                R1[2][2];
      if (BthetaZ < -1.0) {
         BthetaZ = -1.0;
      }
      ct_cos = acos(BthetaZ);

      if ((ct_cos <= pi - ep) || (ct_cos >= pi + ep)) {// Base tri check
         f_row[cntr] = pts;
         //Convert coordinates from the C0 (tang) to the FIXED (F) frame
         X1 = *(*(bolt + pts) +4) * Q[0][0] + *(*(bolt + pts) +5) * Q[1][0] +
                 *(*(bolt + pts) +6) * Q[2][0] + xc;
         Y1 = *(*(bolt + pts) +4) * Q[0][1] + *(*(bolt + pts) +5) * Q[1][1] +
                 *(*(bolt + pts) +6) * Q[2][1] + yc;
         X2 = *(*(bolt + pts) +7) * Q[0][0] + *(*(bolt + pts) +8) * Q[1][0] +
                 *(*(bolt + pts) +9) * Q[2][0] + xc;
         Y2 = *(*(bolt + pts) +7) * Q[0][1] + *(*(bolt + pts) +8) * Q[1][1] +
                 *(*(bolt + pts) +9) * Q[2][1] + yc;
         X3 = *(*(bolt + pts) +10) * Q[0][0] + *(*(bolt + pts) +11) * Q[1][0] +
                 *(*(bolt + pts) +12) * Q[2][0] + xc;
         Y3 = *(*(bolt + pts) +10) * Q[0][1] + *(*(bolt + pts) +11) * Q[1][1] +
                 *(*(bolt + pts) +12) * Q[2][1] + yc;
         X_avg = (X1 + X2 + X3)/3.0;
         Y_avg = (Y1 + Y2 + Y3)/3.0;
         //Compute the avg angular location of the contact in the FIXED frame
         if ((Y_avg >= 0.0) && (X_avg >= 0.0)) {      // Quadrant I
```

```
            delta[cntr] = atan(Y_avg/X_avg);
         } else if ((Y_avg >= 0.0) && (X_avg < 0.0)) {// Quadrant II
            delta[cntr] = pi + atan(Y_avg/X_avg);
         } else if ((Y_avg < 0.0) && (X_avg < 0.0)) { // Quadrant III
            delta[cntr] = pi + atan(Y_avg/X_avg);
         } else if ((Y_avg < 0.0) && (X_avg >= 0.0)) {// Quadrant IV
            delta[cntr] = 2.0*pi + atan(Y_avg/X_avg);
         }
         cntr++;
      }
   }
   //We need to sort the delta's first (from min to max)
   int     ptr, first, G;
   double  hold, holdB[13], holdN[13];
   for (k=0; k<=cntr-2; k++) {
      ptr = k;
      first = k+1;
      for (int l=first; l<=cntr-1; l++) {
         if (delta[l] < delta[ptr]) {
            ptr = l;
         }
      }
      hold = delta[k];
      //Must sort vertex info along with delta's
      for (G=0; G<=12; G++) {
         holdB[G] = *(*(bolt + f_row[k]) + G);
         holdN[G] = *(*(nut  + f_row[k]) + G);
      }
      delta[k] = delta[ptr];
      for (G=0; G<=12; G++) {
         *(*(bolt + f_row[k]) + G) = *(*(bolt + f_row[ptr]) + G);
         *(*(nut  + f_row[k]) + G) = *(*(nut  + f_row[ptr]) + G);
      }
      delta[ptr] = hold;
      for (G=0; G<=12; G++) {
         *(*(bolt + f_row[ptr]) + G) = (float)holdB[G];
         *(*(nut  + f_row[ptr]) + G) = (float)holdN[G];
      }
   }
//Must Alter Delta's when contact tri's occur on the border of Quad I and IV
   if ((delta[0] <= 3.0*pi/180.0) && (delta[cntr-1] >= 357.0*pi/180.0)) {
      delta[0] = delta[0] + 2.0*pi;
      for (k=1; k<=cntr-1; k++) {
         if (((delta[k] + 2.0*pi) - delta[k-1]) <= 3.0*pi/180.0) {
            delta[k] = delta[k] + 2.0*pi;
         }
      }
      resort_req = 1;
   }
   //Resort the Delta's if the delta angles were re-calculated
   if (resort_req == 1) {
      for (k=0; k<=cntr-2; k++) {
         ptr = k;
         first = k+1;
         for (int l=first; l<=cntr-1; l++) {
            if (delta[l] < delta[ptr]) {
               ptr = l;
            }
         }
         hold = delta[k];
         //Must sort vertex info along with delta's
         for (G=0; G<=12; G++) {
            holdB[G] = *(*(bolt + f_row[k]) + G);
```

```
         holdN[G] = *(*(nut  + f_row[k]) + G);
      }
      delta[k] = delta[ptr];
      for (G=0; G<=12; G++) {
         *(*(bolt + f_row[k]) + G) = *(*(bolt + f_row[ptr]) + G);
         *(*(nut  + f_row[k]) + G) = *(*(nut  + f_row[ptr]) + G);
      }
      delta[ptr] = hold;
      for (G=0; G<=12; G++) {
         *(*(bolt + f_row[ptr]) + G) = (float)holdB[G];
         *(*(nut  + f_row[ptr]) + G) = (float)holdN[G];
      }
   }
}
//Now we check for continuity
int  delta_flush=0;
for (k=0; k<=cntr-2; k++) {
   if (fabs(delta[k+1] - delta[k]) < ep) {
      delta_flush++;
   }
}

if ((delta_flush == cntr-1) && (fabs(delta[cntr-1]*180.0/pi –
                                    delta[0]*180.0/pi) >= 45.0)) {
   if (step == 0.0001) {
      cout << "Flush with surface" << endl;
      FLUSH = 1;
      flush_true = 1;
      contact_set = 2;
   } else if (step != 0.0001) {
      xroll = xroll + step*pi/180.0;
      step = step/10.0;
      xroll = xroll - step*pi/180.0;
   }
} else if ((delta_flush != cntr-1) && (step == 0.0001)) {
   for (int ii=0; ii<=cntr-1; ii++) {
      if (cycle == 0) {//Is this the first time in this function?
         for (jj=0; jj<=12; jj++) {
            bc_loc[0][row_ct[0]][jj] = *(*(bolt + f_row[ii]) +jj);
            nc_loc[0][row_ct[0]][jj] = *(*(nut + f_row[ii]) +jj);
         }
         control[ii] = delta[ii];
         cycle++;
         row_ct[0]++;
      } else if (cycle != 0) {
         for (kk=0; kk<=cycle-1; kk++) { // Check Against Known Delta's
            if (fabs(delta[ii] - control[kk]) <= ep) {
               found = kk; // Set the contact location #
            }
         }
         if (found != -1) {//Same contact point, store in contact loc # array
            for (jj=0; jj<=12; jj++) {
               bc_loc[found][row_ct[kk-1]][jj] = *(*(bolt + f_row[ii]) +jj);
               nc_loc[found][row_ct[kk-1]][jj] = *(*(nut + f_row[ii]) +jj);
            }
            row_ct[kk-1]++;
            found = -1;
            control[kk-1] = delta[ii];
         } else if (found == -1) {//New contact point, store in new loc array
            for (jj=0; jj<=12; jj++) {
               bc_loc[kk][row_ct[kk]][jj] = *(*(bolt + f_row[ii]) +jj);
               nc_loc[kk][row_ct[kk]][jj] = *(*(nut + f_row[ii]) +jj);
            }
```

```
            control[kk] = delta[ii];
            cycle++;
            contact_set++;  //Global variable to track # of contact points
            row_ct[kk]++;//kk was incremented by 1 at end of kk loop, so it is
         }            //equal to one greater than the ending value
      }               //i.e.  0 to 0 is actually 0 to 1
   }//end of FOR loop
} else if ((delta_flush == cntr-1) && (fabs(delta[cntr-1] - delta[0]) <
                                                    45.0*pi/180.0)) {
   xroll = xroll - step*pi/180.0;
} else if ((delta_flush != cntr-1) && (step != 0.0001)) {
   xroll = xroll + step*pi/180.0;
   step = step/10.0;
   xroll = xroll - step*pi/180.0;
}// end of FLUSH if loop

//NOW OUTPUT THE CONTACT POINT LOCATIONS
if ((contact_set > 1) && (FLUSH == 0)) {
   // Number of contacts
   for (jj=0; jj<=cycle-1; jj++) {
      float (*bolt_split)[13] = new float[500][13];
      float (*nut_split)[13] = new float[500][13];
      for (int ll=0; ll<=row_ct[jj]-1; ll++) {
         for (int pp=0; pp<=12; pp++) {
            *(*(bolt_split + ll) +pp) = (float)bc_loc[jj][ll][pp];
            *(*(nut_split + ll) +pp) = (float)nc_loc[jj][ll][pp];
         }
      }
      //Determine the current contact point
      //Contact points are given in the C0 (tang) frame
      contact_point((row_ct[jj]), R, T, bolt_split, nut_split);
      Xcontact[jj] = midpoint[0];
      Ycontact[jj] = midpoint[1];
      Zcontact[jj] = midpoint[2];
      //Use the C0 (tang) frame coordinates to define the PIVOT axis
      x_t[jj] = Xcontact[jj];
      y_t[jj] = Ycontact[jj];
      z_t[jj] = Zcontact[jj];
      //Convert to FIXED (f) frame
      tang_to_fixed(midpoint);
      //Convert to BOLT (b) frame
      tang_to_bolt(midpoint);
      //We are isolating which of the contact points are at the origin of the
      //C0 frame.  The value of 0.0048 is the allowable error to deviate from
      //the initial contact point
      if ((pow(x_t[jj],2)+pow(y_t[jj],2)+pow(z_t[jj],2)) <= 0.0048) {
         x[jj] = xc;
         y[jj] = yc;
         z[jj] = zc;
      } else if ((pow(x_t[jj],2)+pow(y_t[jj],2)+pow(y_t[jj],2)) > 0.0048) {
         x[jj] = C2_fixed[0];
         y[jj] = C2_fixed[1];
         z[jj] = C2_fixed[2];
      }
      delete [] bolt_split;
      delete [] nut_split;
   }
} else if ((contact_set > 1) && (FLUSH == 1)) {//Flush with NUT surface
   cycle = 2;
   pivot_axes_theo(phi);
}//end of contact point output loop
return 0;
}
```

```
double tang_to_fixed(double c_point[])//c_point[] is given in the C0 (tang) frame
{
   //Convert New Contact Point Into Fixed Reference Frame
   C2_fixed[0] = c_point[0]*Q[0][0] + c_point[1]*Q[1][0]+c_point[2]*Q[2][0]+xc;
   C2_fixed[1] = c_point[0]*Q[0][1] + c_point[1]*Q[1][1]+c_point[2]*Q[2][1]+yc;
   C2_fixed[2] = c_point[0]*Q[0][2] + c_point[1]*Q[1][2]+c_point[2]*Q[2][2]+zc;
   return 0;
}

double tang_to_bolt(double c_point[])//c_point[] is given in the C0 (tang) frame
{
   double  xb_mt,yb_mt,zb_mt;
   //Preserve the orientation of the BOLT vertices,
   //we are converting from the C1 to the C0 frame
   xb_mt = c_point[0]*M[0][0] + c_point[1]*M[0][1] + c_point[2]*M[0][2];
   yb_mt = c_point[0]*M[1][0] + c_point[1]*M[1][1] + c_point[2]*M[1][2];
   zb_mt = c_point[0]*M[2][0] + c_point[1]*M[2][1] + c_point[2]*M[2][2];
   //Transfer from the C0 (tang) to the BOLT (B) frame
   C2_bolt[0] = xb_mt*B1[0][0] + yb_mt*B1[1][0] + zb_mt*B1[2][0] + S[0];
   C2_bolt[1] = xb_mt*B1[0][1] + yb_mt*B1[1][1] + zb_mt*B1[2][1] + S[1];
   C2_bolt[2] = xb_mt*B1[0][2] + yb_mt*B1[1][2] + zb_mt*B1[2][2] + S[2];
   return 0;
}

double bolt_to_FIXED(double pointX, double pointY, double pointZ)
{
   double  pX_f,pY_f,pZ_f,pX_c1,pY_c1,pZ_c1,bX_c0,bY_c0,bZ_c0;
   //Convert from the BOLT (b) to the FIXED (f) frame
   pX_f = pointX*R1[0][0] + pointY*R1[0][1] + pointZ*R1[0][2];
   pY_f = pointX*R1[1][0] + pointY*R1[1][1] + pointZ*R1[1][2];
   pZ_f = pointX*R1[2][0] + pointY*R1[2][1] + pointZ*R1[2][2] + T1[2];
   //Convert from the FIXED (f) to the C1 frame
   pX_c1 = pX_f*Q[0][0] + pY_f*Q[0][1] + pZ_f*Q[0][2] - (xc*Q[0][0] + yc*Q[0][1]
                                                        + zc*Q[0][2]);
   pY_c1 = pX_f*Q[1][0] + pY_f*Q[1][1] + pZ_f*Q[1][2] - (xc*Q[1][0] + yc*Q[1][1]
                                                        + zc*Q[1][2]);
   pZ_c1 = pX_f*Q[2][0] + pY_f*Q[2][1] + pZ_f*Q[2][2] - (xc*Q[2][0] + yc*Q[2][1]
                                                        + zc*Q[2][2]);
   //Convert from the C1 to the C0 frame
   bX_c0 = pX_c1*M[0][0] + pY_c1*M[1][0] + pZ_c1*M[2][0];
   bY_c0 = pX_c1*M[0][1] + pY_c1*M[1][1] + pZ_c1*M[2][1];
   bZ_c0 = pX_c1*M[0][2] + pY_c1*M[1][2] + pZ_c1*M[2][2];
   //Convert from the C0 to the FIXED (f) frame
   B2_fixed[0] = bX_c0*Q[0][0] + bY_c0*Q[1][0] + bZ_c0*Q[2][0] + xc;
   B2_fixed[1] = bX_c0*Q[0][1] + bY_c0*Q[1][1] + bZ_c0*Q[2][1] + yc;
   B2_fixed[2] = bX_c0*Q[0][2] + bY_c0*Q[1][2] + bZ_c0*Q[2][2] + zc;
   return 0;
}
```

```
//This function computes the theoretical contact points given phi
//It is only used when the bolt is flush with the NUT surface
double pivot_axes_theo(double phi)
{
   double  phase = phi*180.0/pi, p=0.05;
   double  TiniB, TiniN, tc1, tc2, R, K1, K2, A, B, C, Q, RR, S, T, U, V;
   double  D, E, F, G, AA, BB, CC, KK1, KK2;
   double  K = sqrt(3.0) * p/(2.0*pi);
   double  L = p/(2.0*pi), a = 0.00055;
   double  b_max = 0.12445, n_max = 0.125, crrb = 0.0025;
   double  rrb = 0.005, crrn = 0.005, rrn = 0.0025;
   double  b_min = b_max - 5.0*sqrt(3.0)*p/16.0;
   double  n_min = n_max - 5.0*sqrt(3.0)*p/16.0;

   if ((phase >= 0.0) & (phase <= 4.78046)) {
      TiniB = 297.23047873*pi/180.0;
      TiniN = 323.36382538*pi/180.0;
      tc1 = 0.5*(1/K *(b_max-n_min-0.5*(crrb+crrn))+TiniB+TiniN+phi);
      R = (n_min + 0.5*crrn) + K*(tc1 - TiniN);
      x[0] = R*cos(tc1);
      y[0] = R*sin(tc1);
      TiniB = 144.92663116*pi/180.0;
      TiniN = 115.66767295*pi/180.0;
      tc2 = 0.5*(1/K *(n_max-b_min-0.5*(rrn+rrb))+TiniB+TiniN+phi);
      R = (n_max - 0.5*rrn) - K*(tc2 - TiniN);
      x[1] = R*cos(tc2);
      y[1] = R*sin(tc2);
   } else if ((phase > 4.78046) & (phase <= 65.3663)) {
      TiniB = 297.23047873*pi/180.0;
      TiniN = 36.63617462*pi/180.0;
      tc1 = 0.5*(1/K *(b_max-n_min-0.5*(crrb+crrn)) +phi-2.0*pi+(TiniB-TiniN));
      R = (n_min + 0.5*crrn) + K*(tc1 + TiniN);
      x[0] = R*cos(tc1);
      y[0] = R*sin(tc1);
      TiniB = 144.92663116*pi/180.0;
      TiniN = 115.66767295*pi/180.0;
      tc2 = 0.5*(1/K *(n_max-b_min-0.5*(rrn+rrb))+TiniB+TiniN+phi);
      R = (n_max - 0.5*rrn) - K*(tc2 - TiniN);
      x[1] = R*cos(tc2);
      y[1] = R*sin(tc2);
   } else if ((phase > 65.3663) & (phase <= 70.3030)) {
      TiniB = 297.23047873*pi/180.0;
      TiniN = 36.63617462*pi/180.0;
      tc1 = 0.5*(1/K *(b_max-n_min-0.5*(crrb+crrn)) +phi-2.0*pi+(TiniB-TiniN));
      R = (n_min + 0.5*crrn) + K*(tc1 + TiniN);
      x[0] = R*cos(tc1);
      y[0] = R*sin(tc1);
      TiniB = 144.92663116*pi/180.0;
      TiniN = 212.57921568*pi/180.0;
      A = pow(L,2) + pow(K,2);
      B = -2*(pow(L,2)*TiniN + pow(K,2)*(phi +TiniB) + K*(a + 0.5*crrn) +
                                                        sqrt(3.0)*crrn*L/2.0);
      C = pow(L,2)*pow(TiniN,2) + pow(K,2)*pow((phi +TiniB),2) + 2.0*K*(a +
              0.5*crrn)*(phi + TiniB) + sqrt(3.0)*crrn*L*TiniN + a*(a + crrn);
      tc2 = (-B - sqrt(pow(B,2) - 4*A*C))/(2*A);
      R = (n_min + crrn) - sqrt(pow(crrn,2) - pow((sqrt(3.0)*crrn/2.0 - L*(tc2 –
                                                              TiniN)),2));
      x[1] = R*cos(tc2);
      y[1] = R*sin(tc2);
   } else if ((phase > 70.3030) & (phase <= 113.5918)) {
      TiniB = 297.23047873*pi/180.0;
      TiniN = 36.63617462*pi/180.0;
      tc1 = 0.5*(1/K *(b_max-n_min-0.5*(crrb+crrn)) +phi-2.0*pi+(TiniB-TiniN));
```

```
   R = (n_min + 0.5*crrn) + K*(tc1 + TiniN);
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 113.74971662*pi/180.0;
   TiniN = 212.57921568*pi/180.0;
   K1 = 2*pow(L,2)*(TiniN - (phi+TiniB) + sqrt(3.0)*crrn/(2*L));
   K2 = pow(L,2)*(pow((phi+TiniB),2)-pow(TiniN,2)) - 0.75*pow(crrn,2) –
                                          sqrt(3.0)*crrn*L*TiniN -pow(a,2);
   A = pow(K1,2)/(4*pow(a,2)) + pow(L,2);
   B = K1*K2/(2*pow(a,2)) -2*pow(L,2)*(phi+TiniB);
   C = pow(K2,2)/(4*pow(a,2)) -pow(rrb,2) + pow(L,2)*pow((phi+TiniB),2);
   tc2 = (-B + sqrt(pow(B,2) - 4*A*C))/(2*A);
   R = (n_min + crrn) - sqrt(pow(crrn,2) - pow((sqrt(3.0)*crrn/2.0 - L*(tc2 –
                                                                TiniN)),2));
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 113.5918) & (phase <= 120.7590)) {
   TiniB = 297.23047873*pi/180.0;
   TiniN = 36.63617462*pi/180.0;
   tc1 = 0.5*(1/K *(b_max-n_min-0.5*(crrb+crrn)) +phi-2.0*pi+(TiniB-TiniN));
   R = (n_min + 0.5*crrn) + K*(tc1 + TiniN);
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 113.74971662*pi/180.0;
   TiniN = 243.75613022*pi/180.0;
   tc2 = (TiniB + phi) + 1/L *sqrt(pow(rrb,2) - pow((b_min + rrb –
                                                              n_min),2));
   R = n_min;
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 120.7590) & (phase <= 126.1418)) {
   TiniB = 281.64202146*pi/180.0;
   TiniN = 36.63617462*pi/180.0;
   Q = pow(K,2) + pow(L,2);
   RR = 2.0*K*(K*TiniN + n_min + 0.5*crrn - b_max +crrb)-
                                             2*pow(L,2)*(phi+TiniB-2*pi);
   S = pow((K*TiniN + n_min + 0.5*crrn - b_max +crrb),2) -pow(crrb,2) +
                                          pow(L,2)*pow((phi+TiniB-2*pi),2);
   tc1 = (-RR + sqrt(pow(RR,2)-4*Q*S))/(2*Q);
   R = (n_min + 0.5*crrn) + K*(tc1 + TiniN);
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 113.74971662*pi/180.0;
   TiniN = 243.75613022*pi/180.0;
   tc2 = (TiniB + phi) + 1/L *sqrt(pow(rrb,2) - pow((b_min + rrb –
                                                              n_min),2));
   R = n_min;
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 126.1418) & (phase <= 142.1194)) {
   TiniB = 281.64202146*pi/180.0;
   TiniN = 60.27536811*pi/180.0;
   K1 = 2.0*pow(L,2)*((phi+TiniB-2*pi)-TiniN-sqrt(3.0)*rrn/(2.0*L));
   K2 = pow(L,2)*(pow(TiniN,2)-pow((phi+TiniB-2*pi),2)+sqrt(3.0)*rrn*TiniN/L)
                                            + pow(a,2) + 0.75*pow(rrn,2);
   T = pow(K1,2)/(4.0*pow(a,2)) + pow(L,2);
   U = K1*K2/(2.0*pow(a,2)) - 2.0*(phi+TiniB-2.0*pi)*pow(L,2);
   V = pow(K2,2)/(4.0*pow(a,2)) -pow(crrb,2)+pow(L,2)*pow((phi+TiniB-
                                                              2*pi),2);
   tc1 = (-U - sqrt(pow(U,2) - 4.0*T*V))/(2.0*T);
   R = (n_max - rrn) + sqrt(pow(rrn,2) - pow((sqrt(3.0)*rrn/2.0 - L*(tc1 –
                                                          TiniN)),2));
   x[0] = R*cos(tc1);
```

```
   y[0] = R*sin(tc1);
   TiniB = 113.74971662*pi/180.0;
   TiniN = 243.75613022*pi/180.0;
   tc2 = (TiniB + phi) + 1/L *sqrt(pow(rrb,2) - pow((b_min + rrb –
                                                          n_min),2));
   R = n_min;
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 142.1194) & (phase <= 162.0266)) {
   TiniB = 257.42663116*pi/180.0;
   TiniN = 60.27536811*pi/180.0;
   tc1 = 1.0/L *(sqrt(3.0)*rrn/2.0 - sqrt(pow(rrn,2) - pow((rrn-a),2))) +
                                                            TiniN;
   R = (n_max - rrn) + sqrt(pow(rrn,2) - pow((sqrt(3.0)*rrn/2.0 - L*(tc1 –
                                                          TiniN)),2));
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 113.74971662*pi/180.0;
   TiniN = 243.75613022*pi/180.0;
   tc2 = (TiniB + phi) + 1.0/L *sqrt(pow(rrb,2) - pow((b_min + rrb –
                                                          n_min),2));
   R = n_min;
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 162.0266) & (phase <= 167.1732)) {
   TiniB = 257.42663116*pi/180.0;
   TiniN = 60.27536811*pi/180.0;
   tc1 = 1.0/L *(sqrt(3.0)*rrn/2.0 - sqrt(pow(rrn,2) - pow((rrn-a),2))) +
                                                            TiniN;
   R = (n_max - rrn) + sqrt(pow(rrn,2) - pow((sqrt(3.0)*rrn/2.0 - L*(tc1 –
                                                          TiniN)),2));
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 113.74971662*pi/180.0;
   TiniN = 292.18691084*pi/180.0;
   K1 = 2.0*pow(L,2)*(TiniN - (phi+TiniB));
   K2 = pow(L,2)*(pow((phi+TiniB),2)-pow(TiniN,2)) + pow(a,2);
   A = pow(K1,2)/(4.0*pow(a,2)) + pow(L,2);
   B = K1*K2/(2.0*pow(a,2)) -2.0*pow(L,2)*TiniN;
   C = pow(K2,2)/(4.0*pow(a,2)) -pow(crrn,2) + pow(L,2)*pow(TiniN,2);
   tc2 = (-B + sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
   R = (n_min + crrn) - sqrt(pow(crrn,2) - pow(L,2)*pow((tc2-TiniN),2));
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 167.1732) & (phase <= 175.3403)) {
   TiniB = 241.83817389*pi/180.0;
   TiniN = 60.27536811*pi/180.0;
   K1 = 2.0*pow(L,2)*((phi+TiniB-2.0*pi)-TiniN);
   K2 = pow(L,2)*(pow(TiniN,2)-pow((phi+TiniB-2*pi),2)) +
                  sqrt(3.0)*L*(rrn*(TiniN-(phi+TiniB-2.0*pi))) - pow(a,2);
   A = pow(K1,2)/(4.0*pow(a,2)) + pow(L,2);
   B = K1*K2/(2.0*pow(a,2)) -2.0*pow(L,2)*TiniN - sqrt(3.0)*rrn*L;
   C = pow(K2,2)/(4.0*pow(a,2)) -(pow(rrn,2))/4.0 + pow(L,2)*pow(TiniN,2) +
                                                     sqrt(3.0)*rrn*L*TiniN;
   tc1 = (-B - sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
   R = (n_max - rrn) + sqrt(pow(rrn,2) - pow((sqrt(3.0)*rrn/2.0 - L*(tc1-
                                                        TiniN)),2));
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 113.74971662*pi/180.0;
   TiniN = 292.18691084*pi/180.0;
   KK1 = 2.0*pow(L,2)*(TiniN - (phi+TiniB));
   KK2 = pow(L,2) *(pow((phi+TiniB),2)-pow(TiniN,2)) + pow(a,2);
```

```
   AA = pow(K1,2)/(4.0*pow(a,2)) + pow(L,2);
   BB = KK1*KK2/(2.0*pow(a,2)) -2.0*pow(L,2)*TiniN;
   CC = pow(KK2,2)/(4.0*pow(a,2)) -pow(crrn,2) + pow(L,2)*pow(TiniN,2);
   tc2 = (-BB + sqrt(pow(BB,2) - 4.0*AA*CC))/(2.0*AA);
   R = (n_min + crrn) - sqrt(pow(crrn,2) - pow(L,2)*pow((tc2-TiniN),2));
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 175.3403) & (phase <= 175.7867)) {
   TiniB = 241.83817389*pi/180.0;
   TiniN = 36.63617462*pi/180.0;
   A = pow(K,2) + pow(L,2);
   B = 2.0*pow(K,2)*TiniN + 2.0*K*(n_min-b_max+crrn) –
                          2.0*pow(L,2)*(phi+TiniB-2*pi)-sqrt(3.0)*crrb*L;
   C = pow(K,2)*pow(TiniN,2) + 2.0*K*(n_min-b_max+crrn)*TiniN + pow((n_min-
        b_max+crrn),2) - (pow(crrb,2))/4.0 + pow(L,2)*(phi+TiniB-2.0*pi) +
        sqrt(3.0)*crrb*L*(phi+TiniB-2.0*pi);
   tc1 = (-B + sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
   R = (n_min + 0.5*crrn) + K*(tc1 + TiniN);
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 113.74971662*pi/180.0;
   TiniN = 292.18691084*pi/180.0;
   KK1 = 2.0*pow(L,2)*(TiniN - (phi+TiniB));
   KK2 = pow(L,2)*(pow((phi+TiniB),2)-pow(TiniN,2)) + pow(a,2);
   AA = pow(KK1,2)/(4.0*pow(a,2)) + pow(L,2);
   BB = KK1*KK2/(2.0*pow(a,2)) -2.0*pow(L,2)*TiniN;
   CC = pow(KK2,2)/(4.0*pow(a,2)) -pow(crrn,2) + pow(L,2)*pow(TiniN,2);
   tc2 = (-BB + sqrt(pow(BB,2) - 4.0*AA*CC))/(2.0*AA);
   R = (n_min + crrn) - sqrt(pow(crrn,2) - pow(L,2)*pow((tc2-TiniN),2));
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 175.7867) & (phase <= 176.1509)) {
   TiniB = 241.83817389*pi/180.0;
   TiniN = 36.63617462*pi/180.0;
   A = pow(K,2) + pow(L,2);
   B = 2.0*pow(K,2)*TiniN + 2.0*K*(n_min-b_max+crrn) –
                         2.0*pow(L,2)*(phi+TiniB-2.0*pi)-sqrt(3.0)*crrb*L;
   C = pow(K,2)*pow(TiniN,2) + 2.0*K*(n_min-b_max+crrn)*TiniN + pow((n_min-
        b_max+crrn),2) - (pow(crrb,2))/4.0 + pow(L,2)*(phi+TiniB-2.0*pi) +
        sqrt(3.0)*crrb*L*(phi+TiniB-2.0*pi);
   tc1 = (-B + sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
   R = (n_min + 0.5*crrn) + K*(tc1 + TiniN);
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 144.92663116*pi/180.0;
   TiniN = 292.18691084*pi/180.0;
   AA = pow(K,2) + pow(L,2);
   BB = -2.0*(pow(L,2)*TiniN + pow(K,2)*(phi+TiniB) + K*(a+0.5*crrn));
   CC = pow(L,2)*pow(TiniN,2) + pow(K,2)*pow((phi+TiniB),2) +
        2.0*K*(a+0.5*crrn)*(phi+TiniB) + pow((a+0.5*crrn),2) - pow(crrn,2);
   tc2 = (-BB - sqrt(pow(BB,2) - 4.0*AA*CC))/(2.0*AA);
   R = (n_min + crrn) - sqrt(pow(crrn,2) - pow(L,2)*pow((tc2-TiniN),2));
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 180.7235) & (phase <= 189.7012)) {
   TiniB = 281.64202146*pi/180.0;
   TiniN = 100.07921568*pi/180.0;
   K1 = 2.0*pow(L,2)*(phi + TiniB - 2.0*pi - TiniN);
   K2 = pow(L,2)*(pow(TiniN,2) - pow((phi + TiniB - 2*pi),2)) - pow(a,2);
   A = pow(K1,2)/(4.0*pow(a,2)) + pow(L,2);
   B = K1*K2/(2.0*pow(a,2)) -2.0*TiniN*pow(L,2);
   C = pow(K2,2)/(4.0*pow(a,2)) - pow(rrn,2) + pow(L,2)*pow(TiniN,2);
   tc1 = (-B + sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
```

```
   R = n_max - rrn + sqrt(pow(rrn,2) - pow(L,2)*pow((tc1 - TiniN),2));
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 34.14202146*pi/180.0;
   TiniN = 212.57921568*pi/180.0;
   KK1 = -2.0*pow(L,2)*(phi + TiniB - TiniN);
   KK2 = -1.0*pow(L,2)*(pow(TiniN,2) - pow((phi+TiniB),2)) –
                          sqrt(3.0)*L*rrb*(TiniN-(phi+TiniB)) - pow(a,2);
   E = pow(KK1,2)/(4.0*pow(a,2)) + pow(L,2);
   F = KK1*KK2/(2.0*pow(a,2)) - sqrt(3.0)*rrb*L - 2.0*pow(L,2)*(phi+TiniB);
   G = pow(KK2,2)/(4.0*pow(a,2)) - 0.25*pow(rrb,2) +
            sqrt(3.0)*rrb*L*(phi+TiniB) + pow(L,2)*pow((phi + TiniB),2);
   tc2 = (-F - sqrt(pow(F,2) - 4.0*E*G))/(2.0*E);
   R = n_min + crrn - sqrt(pow(crrn,2) - pow((sqrt(3.0)*crrn/2.0 - L*(tc2 –
                                                              TiniN)),2));
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 189.7012) & (phase <= 194.8518)) {
   TiniB = 257.42663116*pi/180.0;
   TiniN = 100.07921568*pi/180.0;
   A = 1.0;
   B = -2.0*TiniN;
   C = pow(TiniN,2) + (pow(a,2) - 2.0*a*rrn)/pow(L,2);
   tc1 = (-B + sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
   R = n_max - rrn + sqrt(pow(rrn,2) - pow(L,2)*pow((tc1 - TiniN),2));
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 34.14202146*pi/180.0;
   TiniN = 212.57921568*pi/180.0;
   KK1 = -2.0*pow(L,2)*(phi + TiniB - TiniN);
   KK2 = -1.0*pow(L,2)*(pow(TiniN,2) - pow((phi+TiniB),2)) –
                           sqrt(3.0)*L*rrb*(TiniN-(phi+TiniB)) - pow(a,2);
   E = pow(KK1,2)/(4.0*pow(a,2)) + pow(L,2);
   F = KK1*KK2/(2.0*pow(a,2)) - sqrt(3.0)*rrb*L - 2.0*pow(L,2)*(phi+TiniB);
   G = pow(KK2,2)/(4.0*pow(a,2)) - 0.25*pow(rrb,2) +
               sqrt(3.0)*rrb*L*(phi+TiniB) + pow(L,2)*pow((phi + TiniB),2);
   tc2 = (-F - sqrt(pow(F,2) - 4.0*E*G))/(2.0*E);
   R = n_min + crrn - sqrt(pow(crrn,2) - pow((sqrt(3.0)*crrn/2.0 - L*(tc2 –
                                                                TiniN)),2));
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 194.8518) & (phase <= 213.9166)) {
   TiniB = 257.42663116*pi/180.0;
   TiniN = 100.07921568*pi/180.0;
   A = 1.0;
   B = -2.0*TiniN;
   C = pow(TiniN,2) + (pow(a,2) - 2.0*a*rrn)/pow(L,2);
   tc1 = (-B + sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
   R = n_max - rrn + sqrt(pow(rrn,2) - pow(L,2)*pow((tc1 - TiniN),2));
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 34.14202146*pi/180.0;
   TiniN = 243.75613022*pi/180.0;
   D = pow(L,2);
   E = -(sqrt(3.0)*rrb*L + 2.0*pow(L,2)*(phi+TiniB));
   F = sqrt(3.0)*rrb*L*(phi+TiniB) + pow(L,2)*pow((phi+TiniB),2) + pow((rrb-
                                                    a),2) - 0.25*pow(rrb,2);
   tc2 = (-E - sqrt(pow(E,2) - 4.0*D*F))/(2.0*D);
   R = n_min;
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 213.9166) & (phase <= 230.7326)) {
   TiniB = 241.83817389*pi/180.0;
```

```
   TiniN = 100.07921568*pi/180.0;
   K1 = 2.0*pow(L,2)*(phi + TiniB - 2.0*pi) - 2.0*pow(L,2)*TiniN +
                                                        sqrt(3.0)*crrb*L;
   K2 = pow(L,2)*pow(TiniN,2) - pow(L,2)*pow((phi + TiniB - 2.0*pi),2) –
        sqrt(3.0)*crrb*L*(phi+TiniB-2.0*pi)-(0.75*pow(crrb,2)+pow(a,2));
   A = pow(K1,2)/(4.0*pow(a,2)) + pow(L,2);
   B = K1*K2/(2.0*pow(a,2)) - 2.0*pow(L,2)*TiniN;
   C = pow(K2,2)/(4.0*pow(a,2)) - pow(rrn,2) + pow(L,2)*pow(TiniN,2);
   tc1 = (-B + sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
   R = n_max - rrn + sqrt(pow(rrn,2) - pow(L,2)*pow((tc1 - TiniN),2));
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 34.14202146*pi/180.0;
   TiniN = 243.75613022*pi/180.0;
   D = pow(L,2);
   E = -(sqrt(3.0)*rrb*L + 2.0*pow(L,2)*(phi+TiniB));
   F = sqrt(3.0)*rrb*L*(phi+TiniB) + pow(L,2)*pow((phi+TiniB),2) + pow((rrb-
                                                  a),2) - 0.25*pow(rrb,2);
   tc2 = (-E - sqrt(pow(E,2) - 4.0*D*F))/(2.0*D);
   R = n_min;
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 230.7326) & (phase <= 236.1158)) {
   TiniB = 241.83817389*pi/180.0;
   TiniN = 115.66767295*pi/180.0;
   A = pow(K,2) + pow(L,2);
   B = -(2.0*pow(L,2)*(phi+TiniB-2*pi) +2.0*pow(K,2)*TiniN +
                                  2.0*(a+0.5*rrn)*K + sqrt(3.0)*crrb*L);
   C = pow(L,2)*pow((phi+TiniB-2.0*pi),2) + sqrt(3.0)*crrb*L*(phi+TiniB-
      2.0*pi) + pow(K,2)*pow(TiniN,2) + 2.0*(a+0.5*rrn)*K*TiniN + a*(a+rrn);
   tc1 = (-B - sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
   R = n_max - 0.5*rrn - K*(tc1 - TiniN);
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 34.14202146*pi/180.0;
   TiniN = 243.75613022*pi/180.0;
   D = pow(L,2);
   E = -(sqrt(3.0)*rrb*L + 2.0*pow(L,2)*(phi+TiniB));
   F = sqrt(3.0)*rrb*L*(phi+TiniB) + pow(L,2)*pow((phi+TiniB),2) + pow((rrb-
                                                  a),2) - 0.25*pow(rrb,2);
   tc2 = (-E - sqrt(pow(E,2) - 4.0*D*F))/(2.0*D);
   R = n_min;
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 236.1158) & (phase <= 243.2826)) {
   TiniB = 144.92663116*pi/180.0;
   TiniN = 115.66767295*pi/180.0;
   tc1 = (1.0/(2.0*K))*(n_max-b_min -0.5*(rrn+rrb)) + 0.5*(phi + TiniB +
                                                        TiniN -2.0*pi);
   R = n_max - 0.5*rrn - K*(tc1-TiniN);
   x[0] = R*cos(tc1);
   y[0] = R*sin(tc1);
   TiniB = 34.14202146*pi/180.0;
   TiniN = 243.75613022*pi/180.0;
   D = pow(L,2);
   E = -(sqrt(3.0)*rrb*L + 2.0*pow(L,2)*(phi+TiniB));
   F = sqrt(3.0)*rrb*L*(phi+TiniB) + pow(L,2)*pow((phi+TiniB),2) + pow((rrb-
                                                  a),2) - 0.25*pow(rrb,2);
   tc2 = (-E - sqrt(pow(E,2) - 4.0*D*F))/(2.0*D);
   R = n_min;
   x[1] = R*cos(tc2);
   y[1] = R*sin(tc2);
} else if ((phase > 243.2826) & (phase <= 286.5713)) {
```

```
      TiniB = 144.92663116*pi/180.0;
      TiniN = 115.66767295*pi/180.0;
      tc1 = (1.0/(2.0*K))*(n_max-b_min -0.5*(rrn+rrb)) + 0.5*(phi + TiniB +
                                                              TiniN -2.0*pi);
      R = n_max - 0.5*rrn - K*(tc1-TiniN);
      x[0] = R*cos(tc1);
      y[0] = R*sin(tc1);
      TiniB = 34.14202146*pi/180.0;
      TiniN = 292.18691084*pi/180.0;
      K1 = 2.0*pow(L,2)*TiniN - 2.0*pow(L,2)*(phi+TiniB) - sqrt(3.0)*rrb*L;
      K2 = pow(L,2)*(pow((phi+TiniB),2)-pow(TiniN,2)) + 0.75*pow(rrb,2) +
                                    sqrt(3.0)*rrb*L*(phi+TiniB) + pow(a,2);
      A = pow(K1,2)/(4.0*pow(a,2)) + pow(L,2);
      B = K1*K2/(2.0*pow(a,2)) - 2.0*TiniN*pow(L,2);
      C = pow(K2,2)/(4.0*pow(a,2)) - pow(crrn,2) + pow(L,2)*pow(TiniN,2);
      tc2 = (-B - sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
      R = n_min + crrn - sqrt(pow(crrn,2) - pow(L,2)*pow((tc2-TiniN),2));
      x[1] = R*cos(tc2);
      y[1] = R*sin(tc2);
   } else if ((phase > 286.5713) & (phase <= 291.5081)) {
      TiniB = 144.92663116*pi/180.0;
      TiniN = 115.66767295*pi/180.0;
      tc1 = (1.0/(2.0*K))*(n_max-b_min -0.5*(rrn+rrb)) + 0.5*(phi + TiniB +
                                                              TiniN -2.0*pi);
      R = n_max - 0.5*rrn - K*(tc1-TiniN);
      x[0] = R*cos(tc1);
      y[0] = R*sin(tc1);
      TiniB = 62.76952127*pi/180.0;
      TiniN = 292.18691084*pi/180.0;
      Q = n_min - b_max + crrn + 0.5*crrb;
      A = pow(K,2) + pow(L,2);
      B = 2.0*(Q*K - pow(K,2)*(phi-TiniB) - pow(L,2)*TiniN);
      C = -2.0*Q*K*(phi-TiniB) + pow(K,2)*pow((phi-TiniB),2) +
                          pow(L,2)*pow(TiniN,2) - pow(crrn,2) + pow(Q,2);
      tc2 = (-B + sqrt(pow(B,2) - 4.0*A*C))/(2.0*A);
      R = n_min + crrn - sqrt(pow(crrn,2) - pow(L,2)*pow((tc2 - TiniN),2));
      x[1] = R*cos(tc2);
      y[1] = R*sin(tc2);
   } else if ((phase > 291.5081) & (phase <= 360.0)) {
      TiniB = 144.92663116*pi/180.0;
      TiniN = 115.66767295*pi/180.0;
      tc1 = (1.0/(2.0*K))*(n_max-b_min -0.5*(rrn+rrb)) + 0.5*(phi + TiniB +
                                                              TiniN -2.0*pi);
      R = n_max - 0.5*rrn - K*(tc1-TiniN);
      x[0] = R*cos(tc1);
      y[0] = R*sin(tc1);
      TiniB = 62.76952127*pi/180.0;
      TiniN = 323.36382538*pi/180.0;
      tc2 = (1.0/(2.0*K))*(b_max-n_min-0.5*(crrb+crrn)) + 0.5*(phi+TiniN-TiniB);
      R = n_min + 0.5*crrn + K*(tc2 - TiniN);
      x[1] = R*cos(tc2);
      y[1] = R*sin(tc2);
   }
   //Contacts occur at the surface of the NUT
   z[0] = 0.0500;
   z[1] = 0.0500;
   return 0;
}
```

```
void third_model_input(double R1[][3], double T1[], RAPID_model *bolt3, float
                       (*pb3_tri)[13], float (*pb_tri)[13], RAPID_model *nut3,
                      float (*pn3_tri)[13], float (*pn_tri)[13], double xroll)
{
   int     mm,nn,blt_cnt=0,nut_cnt=0;
   double  xf_b3,yf_b3,zf_b3,xf_n3,yf_n3,zf_n3;//FIXED frame declarations
   double  xt_b3,yt_b3,zt_b3,xt_n3,yt_n3,zt_n3;//CO (tang) frame declarations
   double  xmt_b3,ymt_b3,zmt_b3;        //C1 frame declarations
   //P0 (pivot) frame declarations
   double  xp_b3[3],yp_b3[3],zp_b3[3],xp_n3[3],yp_n3[3],zp_n3[3];

   //BOLT3 Facet Input Area
   double  theta_X, theta_Y, theta_Z;
   double  theta_X_t, theta_Y_t, theta_Z_t;
   double  theta_X_mt, theta_Y_mt, theta_Z_mt;
   float fixed_ID;
   bolt3->BeginModel();
   for (mm=0; mm<=bcount-1; mm++) {
      fixed_ID = *(*(pb_tri+blt_cnt)+(0));
      //Convert from the BOLT (b) to the FIXED (F) frame
      theta_X  = *(*(pb_tri+blt_cnt)+(1)) * R1[0][0] + *(*(pb_tri+blt_cnt)+(2))
                  * R1[0][1] + *(*(pb_tri+blt_cnt)+(3)) * R1[0][2];
      theta_Y  = *(*(pb_tri+blt_cnt)+(1)) * R1[1][0] + *(*(pb_tri+blt_cnt)+(2))
                  * R1[1][1] + *(*(pb_tri+blt_cnt)+(3)) * R1[1][2];
      theta_Z  = *(*(pb_tri+blt_cnt)+(1)) * R1[2][0] + *(*(pb_tri+blt_cnt)+(2))
                  * R1[2][1] + *(*(pb_tri+blt_cnt)+(3)) * R1[2][2];
      //Convert from the FIXED (F) to the C1 frame
      theta_X_mt = theta_X * Q[0][0] + theta_Y * Q[0][1] + theta_Z * Q[0][2];
      theta_Y_mt = theta_X * Q[1][0] + theta_Y * Q[1][1] + theta_Z * Q[1][2];
      theta_Z_mt = theta_X * Q[2][0] + theta_Y * Q[2][1] + theta_Z * Q[2][2];
      //Convert from the C1 to the C0 (tang) frame
      theta_X_t = theta_X_mt*M[0][0] + theta_Y_mt*M[1][0] + theta_Z_mt*M[2][0];
      theta_Y_t = theta_X_mt*M[0][1] + theta_Y_mt*M[1][1] + theta_Z_mt*M[2][1];
      theta_Z_t = theta_X_mt*M[0][2] + theta_Y_mt*M[1][2] + theta_Z_mt*M[2][2];
      //Convert from the C0 (tang) to the P0 (pivot) frame
      *(*(pb3_tri+blt_cnt)+(0)) = fixed_ID;
      *(*(pb3_tri+blt_cnt)+(1)) = (float)(theta_X_t * P[0][0] + theta_Y_t *
                                  P[0][1] + theta_Z_t * P[0][2]);
      *(*(pb3_tri+blt_cnt)+(2)) = (float)(theta_X_t * P[1][0] + theta_Y_t *
                                  P[1][1] + theta_Z_t * P[1][2]);
      *(*(pb3_tri+blt_cnt)+(3)) = (float)(theta_X_t * P[2][0] + theta_Y_t *
                                  P[2][1] + theta_Z_t * P[2][2]);

      int ee=3;
      for (nn=0; nn<=2; nn++) {
         //Convert from the BOLT (b) to the FIXED (F) frame
         xf_b3 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R1[0][0] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R1[0][1] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R1[0][2] + T1[0];
         yf_b3 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R1[1][0] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R1[1][1] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R1[1][2] + T1[1];
         zf_b3 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R1[2][0] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R1[2][1] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R1[2][2] + T1[2];
         //Convert from the FIXED (F) to the C1 frame
         xmt_b3 = xf_b3 * Q[0][0] + yf_b3 * Q[0][1] + zf_b3 * Q[0][2] –
                    (Q[0][0]*U[0] + Q[0][1]*U[1] + Q[0][2]*U[2]);
         ymt_b3 = xf_b3 * Q[1][0] + yf_b3 * Q[1][1] + zf_b3 * Q[1][2] –
                    (Q[1][0]*U[0] + Q[1][1]*U[1] + Q[1][2]*U[2]);
         zmt_b3 = xf_b3 * Q[2][0] + yf_b3 * Q[2][1] + zf_b3 * Q[2][2] –
                    (Q[2][0]*U[0] + Q[2][1]*U[1] + Q[2][2]*U[2]);
         //Convert from the C1 to the C0 (tang) frame
```

```
      xt_b3 = xmt_b3*M[0][0] + ymt_b3*M[1][0] + zmt_b3*M[2][0];
      yt_b3 = xmt_b3*M[0][1] + ymt_b3*M[1][1] + zmt_b3*M[2][1];
      zt_b3 = xmt_b3*M[0][2] + ymt_b3*M[1][2] + zmt_b3*M[2][2];
      //Convert from the C0 (tang) to the P0 (pivot) frame
      xp_b3[nn] = xt_b3 * P[0][0] + yt_b3 * P[0][1] + zt_b3 * P[0][2];
      yp_b3[nn] = xt_b3 * P[1][0] + yt_b3 * P[1][1] + zt_b3 * P[1][2];
      zp_b3[nn] = xt_b3 * P[2][0] + yt_b3 * P[2][1] + zt_b3 * P[2][2];
      *(*(pb3_tri+blt_cnt)+(1+ee)) = (float)(xp_b3[nn]);
      *(*(pb3_tri+blt_cnt)+(2+ee)) = (float)(yp_b3[nn]);
      *(*(pb3_tri+blt_cnt)+(3+ee)) = (float)(zp_b3[nn]);
      ee += 3;
   }
   double b3p0[3] = { xp_b3[0], yp_b3[0], zp_b3[0] };
   double b3p1[3] = { xp_b3[1], yp_b3[1], zp_b3[1] };
   double b3p2[3] = { xp_b3[2], yp_b3[2], zp_b3[2] };
   bolt3->AddTri(b3p0, b3p1, b3p2, blt_cnt);
   blt_cnt++;
}
bolt3->EndModel();

//NUT3 Facet Input Area
double  Ntheta_X, Ntheta_Y, Ntheta_Z;
double  Ntheta_X_t, Ntheta_Y_t, Ntheta_Z_t;
float Nfixed_ID;
nut3->BeginModel();
for (mm=0; mm<=ncount-1; mm++) {
   //Extract NUT direction cosines given in the FIXED (F) frame
   Nfixed_ID = *(*(pn_tri+nut_cnt)+(0));
   Ntheta_X  = *(*(pn_tri+nut_cnt)+(1));
   Ntheta_Y  = *(*(pn_tri+nut_cnt)+(2));
   Ntheta_Z  = *(*(pn_tri+nut_cnt)+(3));
   //Convert from the FIXED (F) to the C0 (tang) frame
   Ntheta_X_t = Ntheta_X * Q[0][0] + Ntheta_Y * Q[0][1] + Ntheta_Z * Q[0][2]
                        - (Q[0][0]*U[0] + Q[0][1]*U[1] + Q[0][2]*U[2]);
   Ntheta_Y_t = Ntheta_X * Q[1][0] + Ntheta_Y * Q[1][1] + Ntheta_Z * Q[1][2]
                        - (Q[1][0]*U[0] + Q[1][1]*U[1] + Q[1][2]*U[2]);
   Ntheta_Z_t = Ntheta_X * Q[2][0] + Ntheta_Y * Q[2][1] + Ntheta_Z * Q[2][2]
                        - (Q[2][0]*U[0] + Q[2][1]*U[1] + Q[2][2]*U[2]);
   //Convert from the C0 (tang) to the P0 (pivot) frame
   *(*(pn3_tri+nut_cnt)+(0)) = Nfixed_ID;
   *(*(pn3_tri+nut_cnt)+(1)) = (float)(Ntheta_X_t * P[0][0] + Ntheta_Y_t *
                        P[0][1] + Ntheta_Z_t * P[0][2]);
   *(*(pn3_tri+nut_cnt)+(2)) = (float)(Ntheta_X_t * P[1][0] + Ntheta_Y_t *
                        P[1][1] + Ntheta_Z_t * P[1][2]);
   *(*(pn3_tri+nut_cnt)+(3)) = (float)(Ntheta_X_t * P[2][0] + Ntheta_Y_t *
                        P[2][1] + Ntheta_Z_t * P[2][2]);

   int ef=3;
   for (nn=0; nn<=2; nn++) {
      //Extract NUT coordinates given in the FIXED (F) frame
      xf_n3 = *(*(pn_tri+nut_cnt)+(1+ef));
      yf_n3 = *(*(pn_tri+nut_cnt)+(2+ef));
      zf_n3 = *(*(pn_tri+nut_cnt)+(3+ef));
      //Convert from the FIXED (F) to the C0 (tang) frame
      xt_n3 = xf_n3 * Q[0][0] + yf_n3 * Q[0][1] + zf_n3 * Q[0][2] –
                              (Q[0][0]*U[0] + Q[0][1]*U[1] + Q[0][2]*U[2]);
      yt_n3 = xf_n3 * Q[1][0] + yf_n3 * Q[1][1] + zf_n3 * Q[1][2] –
                              (Q[1][0]*U[0] + Q[1][1]*U[1] + Q[1][2]*U[2]);
      zt_n3 = xf_n3 * Q[2][0] + yf_n3 * Q[2][1] + zf_n3 * Q[2][2] –
                              (Q[2][0]*U[0] + Q[2][1]*U[1] + Q[2][2]*U[2]);
      //Convert from the C0 (tang) to the P0 (pivot) frame
      xp_n3[nn] = xt_n3 * P[0][0] + yt_n3 * P[0][1] + zt_n3 * P[0][2];
      yp_n3[nn] = xt_n3 * P[1][0] + yt_n3 * P[1][1] + zt_n3 * P[1][2];
```

```
        zp_n3[nn] = xt_n3 * P[2][0] + yt_n3 * P[2][1] + zt_n3 * P[2][2];

        *(*(pn3_tri+nut_cnt)+(1+ef)) = (float)(xp_n3[nn]);
        *(*(pn3_tri+nut_cnt)+(2+ef)) = (float)(yp_n3[nn]);
        *(*(pn3_tri+nut_cnt)+(3+ef)) = (float)(zp_n3[nn]);
        ef += 3;
      }
      double n3p0[3] = { xp_n3[0], yp_n3[0], zp_n3[0] };
      double n3p1[3] = { xp_n3[1], yp_n3[1], zp_n3[1] };
      double n3p2[3] = { xp_n3[2], yp_n3[2], zp_n3[2] };
      nut3->AddTri(n3p0, n3p1, n3p2, nut_cnt);
      nut_cnt++;
   }
   nut3->EndModel();
}
```

```
//This function is only needed when the BOLT is flush with the NUT surface
void special_third(double R1[][3], double T1[], RAPID_model *bolt3, float
                (*pb3_tri)[13], float (*pb_tri)[13], RAPID_model *nut3, float
               (*pn3_tri)[13], float (*pn_tri)[13], double xroll, double phi)
{
   int     mm,nn,blt_cnt=0,nut_cnt=0;
   double  xf_b3,yf_b3,zf_b3,xf_n3,yf_n3,zf_n3;//FIXED (F) frame declarations
   //P0 (pivot) frame declarations
   double  xp_b3[3],yp_b3[3],zp_b3[3],xp_n3[3],yp_n3[3],zp_n3[3];
   //double   R[3][3];   //New rotation matrix that orients BOLT in space

   //BOLT3 Facet Input Area
   double  theta_X_f, theta_Y_f, theta_Z_f;
   float fixed_ID;
   R[0][0] = cos(phi);  R[0][1] = sin(phi);  R[0][2] = 0.0;
   R[1][0] = -sin(phi); R[1][1] = cos(phi);  R[1][2] = 0.0;
   R[2][0] = 0.0;       R[2][1] = 0.0;     R[2][2] = 1.0;

   bolt3->BeginModel();
   for (mm=0; mm<=bcount-1; mm++) {
      fixed_ID = *(*(pb_tri+blt_cnt)+(0));
      //Convert from the BOLT (b) to the FIXED (F) frame
      theta_X_f  = *(*(pb_tri+blt_cnt)+(1)) * R[0][0] + *(*(pb_tri+blt_cnt)+(2))
                   * R[1][0] + *(*(pb_tri+blt_cnt)+(3)) * R[2][0];
      theta_Y_f  = *(*(pb_tri+blt_cnt)+(1)) * R[0][1] + *(*(pb_tri+blt_cnt)+(2))
                   * R[1][1] + *(*(pb_tri+blt_cnt)+(3)) * R[2][1];
      theta_Z_f  = *(*(pb_tri+blt_cnt)+(1)) * R[0][2] + *(*(pb_tri+blt_cnt)+(2))
                   * R[1][2] + *(*(pb_tri+blt_cnt)+(3)) * R[2][2];
      //Convert from the FIXED (F) to the P0 (pivot) frame
      *(*(pb3_tri+blt_cnt)+(0)) = fixed_ID;
      *(*(pb3_tri+blt_cnt)+(1)) = (float)(theta_X_f * P[0][0] + theta_Y_f *
                                  P[0][1] + theta_Z_f * P[0][2]);
      *(*(pb3_tri+blt_cnt)+(2)) = (float)(theta_X_f * P[1][0] + theta_Y_f *
                                  P[1][1] + theta_Z_f * P[1][2]);
      *(*(pb3_tri+blt_cnt)+(3)) = (float)(theta_X_f * P[2][0] + theta_Y_f *
                                  P[2][1] + theta_Z_f * P[2][2]);

      int ee=3;
      for (nn=0; nn<=2; nn++) {
         //Convert from the BOLT (b) to the FIXED (F) frame
         xf_b3 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R[0][0] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R[1][0] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R[2][0] + 0.000;
         yf_b3 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R[0][1] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R[1][1] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R[2][1] + 0.000;
         zf_b3 = (*(*(pb_tri+blt_cnt)+(1+ee))) * R[0][2] +
                 (*(*(pb_tri+blt_cnt)+(2+ee))) * R[1][2] +
                 (*(*(pb_tri+blt_cnt)+(3+ee))) * R[2][2] + 0.050;
         //Convert from the FIXED (F) to the P0 (pivot) frame
         xp_b3[nn] = xf_b3 * P[0][0] + yf_b3 * P[0][1] + zf_b3 * P[0][2] –
                              (P[0][0]*x[0] + P[0][1]*y[0] + P[0][2]*z[0]);
         yp_b3[nn] = xf_b3 * P[1][0] + yf_b3 * P[1][1] + zf_b3 * P[1][2] –
                              (P[1][0]*x[0] + P[1][1]*y[0] + P[1][2]*z[0]);
         zp_b3[nn] = xf_b3 * P[2][0] + yf_b3 * P[2][1] + zf_b3 * P[2][2] –
                              (P[2][0]*x[0] + P[2][1]*y[0] + P[2][2]*z[0]);
         *(*(pb3_tri+blt_cnt)+(1+ee)) = (float)(xp_b3[nn]);
         *(*(pb3_tri+blt_cnt)+(2+ee)) = (float)(yp_b3[nn]);
         *(*(pb3_tri+blt_cnt)+(3+ee)) = (float)(zp_b3[nn]);
         ee += 3;
      }
      double b3p0[3] = { xp_b3[0], yp_b3[0], zp_b3[0] };
      double b3p1[3] = { xp_b3[1], yp_b3[1], zp_b3[1] };
```

```
      double b3p2[3] = { xp_b3[2], yp_b3[2], zp_b3[2] };
      bolt3->AddTri(b3p0, b3p1, b3p2, blt_cnt);
      blt_cnt++;
   }
   bolt3->EndModel();

   //NUT3 Facet Input Area
   double  Ntheta_X, Ntheta_Y, Ntheta_Z;
   float Nfixed_ID;
   nut3->BeginModel();
   for (mm=0; mm<=ncount-1; mm++) {
      //These guys are already in the FIXED frame
      Nfixed_ID = *(*(pn_tri+nut_cnt)+(0));
      Ntheta_X  = *(*(pn_tri+nut_cnt)+(1));
      Ntheta_Y  = *(*(pn_tri+nut_cnt)+(2));
      Ntheta_Z  = *(*(pn_tri+nut_cnt)+(3));
      //Convert to the P0 (pivot) frame
      *(*(pn3_tri+nut_cnt)+(0)) = Nfixed_ID;
      *(*(pn3_tri+nut_cnt)+(1)) = (float)(Ntheta_X * P[0][0] + Ntheta_Y *
                                  P[0][1] + Ntheta_Z * P[0][2]);
      *(*(pn3_tri+nut_cnt)+(2)) = (float)(Ntheta_X * P[1][0] + Ntheta_Y *
                                  P[1][1] + Ntheta_Z * P[1][2]);
      *(*(pn3_tri+nut_cnt)+(3)) = (float)(Ntheta_X * P[2][0] + Ntheta_Y *
                                  P[2][1] + Ntheta_Z * P[2][2]);

      int ef=3;
      for (nn=0; nn<=2; nn++) {
         //These guys are already in the FIXED frame
         xf_n3 = *(*(pn_tri+nut_cnt)+(1+ef));
         yf_n3 = *(*(pn_tri+nut_cnt)+(2+ef));
         zf_n3 = *(*(pn_tri+nut_cnt)+(3+ef));
         //Convert to the P0 (pivot) frame, need only to go from FIXED to P0
         xp_n3[nn] = xf_n3 * P[0][0] + yf_n3 * P[0][1] + zf_n3 * P[0][2] –
                              (P[0][0]*x[0] + P[0][1]*y[0] + P[0][2]*z[0]);
         yp_n3[nn] = xf_n3 * P[1][0] + yf_n3 * P[1][1] + zf_n3 * P[1][2] –
                              (P[1][0]*x[0] + P[1][1]*y[0] + P[1][2]*z[0]);
         zp_n3[nn] = xf_n3 * P[2][0] + yf_n3 * P[2][1] + zf_n3 * P[2][2] –
                              (P[2][0]*x[0] + P[2][1]*y[0] + P[2][2]*z[0]);
         *(*(pn3_tri+nut_cnt)+(1+ef)) = (float)(xp_n3[nn]);
         *(*(pn3_tri+nut_cnt)+(2+ef)) = (float)(yp_n3[nn]);
         *(*(pn3_tri+nut_cnt)+(3+ef)) = (float)(zp_n3[nn]);
         ef += 3;
      }
      double n3p0[3] = { xp_n3[0], yp_n3[0], zp_n3[0] };
      double n3p1[3] = { xp_n3[1], yp_n3[1], zp_n3[1] };
      double n3p2[3] = { xp_n3[2], yp_n3[2], zp_n3[2] };
      nut3->AddTri(n3p0, n3p1, n3p2, nut_cnt);
      nut_cnt++;
   }
   nut3->EndModel();
}
```

```
double contact_group(int cpairs, float (*bolt3_contact_tri)[13], float
                                                    (*nut3_contact_tri)[13])
{
   double  Bx1[2],By1[2],Bz1[2],Bx2[2],By2[2],Bz2[2],Bx3[2],By3[2],Bz3[2];
   double  D=1.0e-8, group[12][400][26];
   double  Adist1,Adist2,Adist3,Bdist1,Bdist2,Bdist3,Cdist1,Cdist2,Cdist3;
   double  Amin,Bmin,Cmin,min_dist;
   int     BLT_id,NUT_id,blt_diff=0,nut_diff=0;
   int     BV1=0,BV2=0,BV3=0,NV1=0,NV2=0,NV3=0,CLOSE=0,ADJ=0;
   int     bb,qq,gr_ct=0,gr_num=0;
   int     row_ct[] = { 0,0,0,0,0,0,0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

   //We need to sort the contact tri's first (from min to max)
   int     k, ptr, first, G;
   double  holdB[13], holdN[13];
   for (k=0; k<=cpairs-2; k++) {
      ptr = k;
      first = k+1;
      for (int l=first; l<=cpairs-1; l++) {
         if (*(*(bolt3_contact_tri+l)+0) < *(*(bolt3_contact_tri + ptr) + 0)) {
            ptr = l;
         }
      }
      //Must hold entire row of bolt3 & nut3, baby
      for (G=0; G<=12; G++) {
         holdB[G] = *(*(bolt3_contact_tri + k) + G);
         holdN[G] = *(*(nut3_contact_tri + k) + G);
      }
      //Reset original "k" row of bolt3 and nut3
      for (G=0; G<=12; G++) {
         *(*(bolt3_contact_tri + k) + G) = *(*(bolt3_contact_tri + ptr) + G);
         *(*(nut3_contact_tri + k) + G) = *(*(nut3_contact_tri  + ptr) + G);
      }
      //Yank the old stuff out of the "hold"
      for (G=0; G<=12; G++) {
         *(*(bolt3_contact_tri + ptr) + G) = (float)holdB[G];
         *(*(nut3_contact_tri  + ptr) + G) = (float)holdN[G];
      }
   }

   //Store the 1st contact pair into GROUP 0
   for (bb=0; bb <= 12; bb++) {
      group[0][0][bb] = *(*(bolt3_contact_tri + 0) + bb);
      group[0][0][bb+13] = *(*(nut3_contact_tri + 0) + bb);
   }
   row_ct[0] = 1;  gr_ct = 1;
   //Begin iterating through all the tris involved in the contact(s)
   for (qq=1; qq < cpairs; qq++) {
      BLT_id = (int)(*(*(bolt3_contact_tri + qq) + 0));
      NUT_id = (int)(*(*(nut3_contact_tri + qq) + 0));
      if (BLT_id != (int)(*(*(bolt3_contact_tri + (qq-1)) + 0))) {
         //Set the BOLT difference flag
         blt_diff = 1;
      }
      if (NUT_id != (int)(*(*(nut3_contact_tri + (qq-1)) + 0))) {
         //Set the NUT difference flag
         nut_diff = 1;
      }
      if (blt_diff == 1) {//Current BOLT tri is different than the previous tri
         //Read in the BOLT vertices (from current & PREVIOUS contact pairs)
         Bx1[0] = *(*(bolt3_contact_tri + qq) + 4);
         Bx1[1] = *(*(bolt3_contact_tri + (qq-1)) + 4);
```

```
By1[0] = *(*(bolt3_contact_tri + qq) + 5);
By1[1] = *(*(bolt3_contact_tri + (qq-1)) + 5);
Bz1[0] = *(*(bolt3_contact_tri + qq) + 6);
Bz1[1] = *(*(bolt3_contact_tri + (qq-1)) + 6);
Bx2[0] = *(*(bolt3_contact_tri + qq) + 7);
Bx2[1] = *(*(bolt3_contact_tri + (qq-1)) + 7);
By2[0] = *(*(bolt3_contact_tri + qq) + 8);
By2[1] = *(*(bolt3_contact_tri + (qq-1)) + 8);
Bz2[0] = *(*(bolt3_contact_tri + qq) + 9);
Bz2[1] = *(*(bolt3_contact_tri + (qq-1)) + 9);
Bx3[0] = *(*(bolt3_contact_tri + qq) + 10);
Bx3[1] = *(*(bolt3_contact_tri + (qq-1)) + 10);
By3[0] = *(*(bolt3_contact_tri + qq) + 11);
By3[1] = *(*(bolt3_contact_tri + (qq-1)) + 11);
Bz3[0] = *(*(bolt3_contact_tri + qq) + 12);
Bz3[1] = *(*(bolt3_contact_tri + (qq-1)) + 12);
//Check to see if BOLT vertices are coincident
if ((fabs(Bx1[0]-Bx1[1]) < D) && (fabs(By1[0]-By1[1]) < D) &&
                                        (fabs(Bz1[0]-Bz1[1]) < D)) {
   BV1 = 1;
} else if ((fabs(Bx1[0]-Bx2[1]) < D) && (fabs(By1[0]-By2[1]) < D) &&
                                        (fabs(Bz1[0]-Bz2[1]) < D)) {
   BV1 = 1;
} else if ((fabs(Bx1[0]-Bx3[1]) < D) && (fabs(By1[0]-By3[1]) < D) &&
                                        (fabs(Bz1[0]-Bz3[1]) < D)) {
   BV1 = 1;
}
if ((fabs(Bx2[0]-Bx2[1]) < D) && (fabs(By2[0]-By2[1]) < D) &&
                                     (fabs(Bz2[0]-Bz2[1]) < D)) {
   BV2 = 1;
} else if ((fabs(Bx2[0]-Bx1[1]) < D) && (fabs(By2[0]-By1[1]) < D) &&
                                        (fabs(Bz2[0]-Bz1[1]) < D)) {
   BV2 = 1;
} else if ((fabs(Bx2[0]-Bx3[1]) < D) && (fabs(By2[0]-By3[1]) < D) &&
                                     (fabs(Bz2[0]-Bz3[1]) < D)) {
   BV2 = 1;
}
if ((fabs(Bx3[0]-Bx3[1]) < D) && (fabs(By3[0]-By3[1]) < D) &&
                                        (fabs(Bz3[0]-Bz3[1]) < D)) {
   BV3 = 1;
} else if ((fabs(Bx3[0]-Bx1[1]) < D) && (fabs(By3[0]-By1[1]) < D) &&
                                        (fabs(Bz3[0]-Bz1[1]) < D)) {
   BV3 = 1;
} else if ((fabs(Bx3[0]-Bx2[1]) < D) && (fabs(By3[0]-By2[1]) < D) &&
                                        (fabs(Bz3[0]-Bz2[1]) < D)) {
   BV3 = 1;
}
//Check to see if BOLT vertices are "close"
//To account for tri's that were present but not adjacent
Adist1 = sqrt(pow((Bx1[0]-Bx1[1]),2) + pow((By1[0]-By1[1]),2) +
                                           pow((Bz1[0]-Bz1[1]),2));
Adist2 = sqrt(pow((Bx1[0]-Bx2[1]),2) + pow((By1[0]-By2[1]),2) +
                                           pow((Bz1[0]-Bz2[1]),2));
Adist3 = sqrt(pow((Bx1[0]-Bx3[1]),2) + pow((By1[0]-By3[1]),2) +
                                           pow((Bz1[0]-Bz3[1]),2));
Bdist1 = sqrt(pow((Bx2[0]-Bx1[1]),2) + pow((By2[0]-By1[1]),2) +
                                           pow((Bz2[0]-Bz1[1]),2));
Bdist2 = sqrt(pow((Bx2[0]-Bx2[1]),2) + pow((By2[0]-By2[1]),2) +
                                           pow((Bz2[0]-Bz2[1]),2));
Bdist3 = sqrt(pow((Bx2[0]-Bx3[1]),2) + pow((By2[0]-By3[1]),2) +
                                           pow((Bz2[0]-Bz3[1]),2));
Cdist1 = sqrt(pow((Bx3[0]-Bx1[1]),2) + pow((By3[0]-By1[1]),2) +
                                           pow((Bz3[0]-Bz1[1]),2));
```

```
      Cdist2 = sqrt(pow((Bx3[0]-Bx2[1]),2) + pow((By3[0]-By2[1]),2) +
                                              pow((Bz3[0]-Bz2[1]),2));
      Cdist3 = sqrt(pow((Bx3[0]-Bx3[1]),2) + pow((By3[0]-By3[1]),2) +
                                              pow((Bz3[0]-Bz3[1]),2));
      //Find the minimum distance
      Amin = minimum(Adist1,Adist2,Adist3);
      Bmin = minimum(Bdist1,Bdist2,Bdist3);
      Cmin = minimum(Cdist1,Cdist2,Cdist3);
      min_dist = minimum(Amin,Bmin,Cmin);

      if (min_dist <= 0.02) {//2.9% of the circumference of the NUT ID
         //The triangles are "close"
         CLOSE = 1;
      }
      }
      if ((BV1 == 1) || (BV2 == 1) || (BV3 == 1)) {
         ADJ = 1;
      }
      if ((ADJ == 1) || (CLOSE == 1)) {
         //The tri's share at least 1 vertex & are adjacent
         if (gr_ct == 1) {
            gr_num = 0;
         } else if (gr_ct != 1) {
            gr_num = gr_ct - 1;
         }
         for (bb=0; bb <= 12; bb++) {
            group[gr_num][row_ct[gr_num]][bb] = *(*(bolt3_contact_tri+qq)+bb);
            group[gr_num][row_ct[gr_num]][bb+13]=*(*(nut3_contact_tri+qq)+bb);
         }
         row_ct[gr_num]++;
      } else if ((BV1 == 0) && (BV2 == 0) && (BV3 == 0)) {
         //The tri's are not adjacent
         gr_ct++;
         gr_num = gr_ct-1;
         for (bb=0; bb <= 12; bb++) {
            group[gr_num][row_ct[gr_num]][bb]=*(*(bolt3_contact_tri + qq)+bb);
            group[gr_num][row_ct[gr_num]][bb+13]=*(*(nut3_contact_tri+qq)+bb);
         }
         row_ct[gr_num]++;
      }
   } else if (blt_diff != 1) {//Current BOLT tri same as the previous tri
      //Store contact pair into the active group
      for (bb=0; bb <= 12; bb++) {
         group[gr_num][row_ct[gr_num]][bb] = *(*(bolt3_contact_tri+qq) + bb);
         group[gr_num][row_ct[gr_num]][bb+13] = *(*(nut3_contact_tri+qq)+bb);
      }
      row_ct[gr_num]++;
   }
   BV1=0;  BV2=0;  BV3=0;  NV1=0;  NV2=0;  NV3=0;  CLOSE=0;   ADJ=0;
   blt_diff=0;  nut_diff=0;
}//end of triangle iteration

//Now let's calculate the contact point locations
if (gr_num >= 2) {
   //This means you have at least 3 contact points, gr_num starts from zero
   //However, let's make sure we have minimized our rotation
   if (fabs(psi_step) == 0.0001) {
      third_pt = 1;
   } else if (fabs(psi_step) != 0.0001) {
      psi = psi - psi_step*pi/180.0;
      psi_step = psi_step/10.0;
   }
}
```

```
if (third_pt == 1) {
   ofstream CPfile("contact_point.txt",ios::out|ios::app);
   ofstream DATAfile("points.txt",ios::out|ios::app);
   CPfile << setprecision(4);
   CPfile << "ROLL =\t" << alpha*180.0/pi << "\tPITCH =\t" << beta*180.0/pi
          << "\tPHASE =\t" << phi*180.0/pi << endl;
   CPfile << "Number of contact locations:\t" << gr_num+1 << "\tXROLL = "
          << xroll*180.0/pi << "\tPSI = " << psi*180.0/pi << endl;
   CPfile << "Contact Point Locations:" << endl;
   CPfile << setprecision(6);
   DATAfile << setprecision(8);
   if (flush_true == 1) {
      CPfile << "We are flush with surface, so the original 1st two points
                 are:\n";
      CPfile << "X\t\t" << "Y\t\t" << "Z\t\t" << endl;
      CPfile << x[0] << "\t" << y[0] << "\t" << z[0] << endl;
      CPfile << x[1] << "\t" << y[1] << "\t" << z[1] << endl;
   } else if (flush_true != 1) {
      CPfile << "The original 1st two points are:\n";
      CPfile << "X0\t" << x[0] << "\tY0\t" << y[0] << "\tZ0\t" << z[0] <<
                endl;
      CPfile << "X1\t" << x[1] << "\tY1\t" << y[1] << "\tZ1\t" << z[1]<<
                endl;
   }
   CPfile << "The three computed contact points are:\n";
   CPfile << "X\t\t" << "Y\t\t" << "Z\t\t" << "# Contact Pairs" << endl;
   for (int jj=0; jj <= gr_num; jj++) {
      //Open up pointers so that we can store the tri's from each group
      float (*blt3_tri)[13] = new float[1000][13];
      float (*nut3_tri)[13] = new float[1000][13];
      float (*blt_lft)[13] = new float[100][13];
      float (*nut_lft)[13] = new float[100][13];
      float (*blt_rht)[13] = new float[100][13];
      float (*nut_rht)[13] = new float[100][13];
      for (int kk=0; kk <= row_ct[jj]-1; kk++) {
         //Extract each group first
         for (bb=0; bb <= 12; bb++) {
            *(*(blt3_tri + kk) + bb) = (float)(group[jj][kk][bb]);
            *(*(nut3_tri + kk) + bb) = (float)(group[jj][kk][bb+13]);
         }
      }
      //What is the spread at the second point of contact?
      //Very first tri
      for (bb=0; bb <= 12; bb++) {
         *(*(blt_lft + 0) + bb) = (float)(group[jj][0][bb]);
         *(*(nut_lft + 0) + bb) = (float)(group[jj][0][bb+13]);
      }
      //Last tri
      for (bb=0; bb <= 12; bb++) {
         *(*(blt_rht + 0) + bb) = (float)(group[jj][row_ct[jj]-1][bb]);
         *(*(nut_rht + 0) + bb) = (float)(group[jj][row_ct[jj]-1][bb+13]);
      }
      contact_point(1, XP, TP, blt_lft, nut_lft);
      pivot_to_fixed(midpoint[0], midpoint[1], midpoint[2]);
      CPfile << C3_fixed[0] << "\t" << C3_fixed[1] << "\t" << C3_fixed[2] <<
                endl;
      contact_point(1, XP, TP, blt_rht, nut_rht);
      pivot_to_fixed(midpoint[0], midpoint[1], midpoint[2]);
      CPfile << C3_fixed[0] << "\t" << C3_fixed[1] << "\t" << C3_fixed[2] <<
                endl;
      //Send extracted group to the contact location function
      contact_point(row_ct[jj], XP, TP, blt3_tri, nut3_tri);
      pointX[jj] = midpoint[0];
```

```
         pointY[jj] = midpoint[1];
         pointZ[jj] = midpoint[2];
         if ((sqrt(pow(pointX[jj],2) + pow(pointY[jj],2) + pow(pointZ[jj],2)))
                                                                  <= 0.012) {
            pointX[jj] = 0.0;  pointY[jj] = 0.0;  pointZ[jj] = 0.0;
         }
         //Convert coordinates from the P0 to the FIXED (F) frame
         pivot_to_fixed(pointX[jj], pointY[jj], pointZ[jj]);
         //Find tangent vector to parametric equation that defines the NUT edge
         nut_vector(C3_fixed[0], C3_fixed[1], C3_fixed[2]);
         //Convert coordinates from the P0 to the BOLT (b) frame
         pivot_to_bolt(pointX[jj], pointY[jj], pointZ[jj]);
         //Find tangent vector to parametric equation that defines the BOLT edge
         blt_vector(C3_bolt[0], C3_bolt[1], C3_bolt[2]);
         //Compute the common normal from the two tangent vectors
         common_normal(BFt_i,BFt_j,BFt_k,NWt_i,NWt_j,NWt_k);
         CPfile << setprecision(6);
         CPfile << setw(9) << C3_fixed[0] << "\t" << setw(9) << C3_fixed[1] <<
                   "\t" << setw(9) << C3_fixed[2] << "\t" << setw(9) <<
                     row_ct[jj] << "\t";
         DATAfile << C3_fixed[0] << "\t" << C3_fixed[1] << "\t" << C3_fixed[2]
                   << "\t" << Com_Norm_i << "\t" << Com_Norm_j << "\t"
                     << Com_Norm_k << endl;
         delete [] blt3_tri;
         delete [] nut3_tri;
      }
      if (flush_true == 1) {//Flush with the surface is a special case
         bolt_to_fixed(0.0, 0.0, 0.0);
         CPfile << "BOLT Origin\n";
         CPfile << pX_f <<"\t" << pY_f << "\t" << pZ_f << endl;
         bolt_to_fixed(0.0, 0.0, 0.05);
         CPfile << "BOLT Top Surface\n";
         CPfile << pX_f <<"\t" << pY_f << "\t" << pZ_f << endl;
      } else if (flush_true != 1) {//BOLT maintains first contact point
         bolt_to_pivot(0.0, 0.0, 0.0);
         pivot_to_fixed(pX_p0, pY_p0, pZ_p0);
         CPfile << "BOLT Origin\n";
         CPfile << C3_fixed[0] <<"\t" << C3_fixed[1] << "\t" << C3_fixed[2] <<
                                                                     endl;
         bolt_to_pivot(0.0, 0.0, 0.05);
         pivot_to_fixed(pX_p0, pY_p0, pZ_p0);
         CPfile << "BOLT Top Surface\n";
         CPfile << C3_fixed[0] <<"\t" << C3_fixed[1] << "\t" << C3_fixed[2] <<
                                                                     endl;
      }
      CPfile << "-----------------------------------------------" << endl;
      CPfile.close();
      DATAfile.close();
   }
   return 0;
}
```

```
double pivot_to_fixed(double pointX, double pointY, double pointZ)
{//pointX,Y,Z are measured in the P0 (pivot) frame
   double  pX_t,pY_t,pZ_t,pX_p1,pY_p1,pZ_p1;
   //Convert from the P0 (pivot) to the C0 (tang) frame
   pX_t = P[0][0]*pointX + P[1][0]*pointY + P[2][0]*pointZ;
   pY_t = P[0][1]*pointX + P[1][1]*pointY + P[2][1]*pointZ;
   pZ_t = P[0][2]*pointX + P[1][2]*pointY + P[2][2]*pointZ;
   //Convert from the C0 (tang) to the FIXED (F) frame
   C3_fixed[0] = Q[0][0]*pX_t + Q[1][0]*pY_t + Q[2][0]*pZ_t + U[0];
   C3_fixed[1] = Q[0][1]*pX_t + Q[1][1]*pY_t + Q[2][1]*pZ_t + U[1];
   C3_fixed[2] = Q[0][2]*pX_t + Q[1][2]*pY_t + Q[2][2]*pZ_t + U[2];
   if (flush_true == 1) {//Flush with the surface is a special case
      //Convert from the P1 to the P0 (pivot) frame
      pX_p1 = pointX*XP[0][0] + pointY*XP[0][1] +pointZ*XP[0][2];
      pY_p1 = pointX*XP[1][0] + pointY*XP[1][1] +pointZ*XP[1][2];
      pZ_p1 = pointX*XP[2][0] + pointY*XP[2][1] +pointZ*XP[2][2];
      //Convert from the P0 (pivot) to the FIXED (F) frame
      C3_fixed[0] = pX_p1*P[0][0] + pY_p1*P[1][0] + pZ_p1*P[2][0] + x[0];
      C3_fixed[1] = pX_p1*P[0][1] + pY_p1*P[1][1] + pZ_p1*P[2][1] + y[0];
      C3_fixed[2] = pX_p1*P[0][2] + pY_p1*P[1][2] + pZ_p1*P[2][2] + z[0];
   }
   return 0;
}

double pivot_to_bolt(double pointX, double pointY, double pointZ)
{//points X,Y,Z are measured in the P0 frame
   double  pX_mp,pY_mp,pZ_mp,pX_t,pY_t,pZ_t,pX_mt,pY_mt,pZ_mt;
   double  pX_f,pY_f,pZ_f;
   //Convert from the P0 (pivot) to the P1 frame
   pX_mp = XP[0][0]*pointX + XP[1][0]*pointY + XP[2][0]*pointZ;
   pY_mp = XP[0][1]*pointX + XP[1][1]*pointY + XP[2][1]*pointZ;
   pZ_mp = XP[0][2]*pointX + XP[1][2]*pointY + XP[2][2]*pointZ;
   //Convert from the P1 to the C0 (tang) frame
   pX_t = pX_mp*P[0][0] + pY_mp*P[1][0] + pZ_mp*P[2][0];
   pY_t = pX_mp*P[0][1] + pY_mp*P[1][1] + pZ_mp*P[2][1];
   pZ_t = pX_mp*P[0][2] + pY_mp*P[1][2] + pZ_mp*P[2][2];
   //Convert from the C0 (tang) to the C1 frame
   pX_mt = pX_t*M[0][0] + pY_t*M[0][1] + pZ_t*M[0][2];
   pY_mt = pX_t*M[1][0] + pY_t*M[1][1] + pZ_t*M[1][2];
   pZ_mt = pX_t*M[2][0] + pY_t*M[2][1] + pZ_t*M[2][2];
   //Convert from the C1 to the BOLT (b) frame
   C3_bolt[0] = pX_mt*B1[0][0] + pY_mt*B1[1][0] + pZ_mt*B1[2][0] + S[0];
   C3_bolt[1] = pX_mt*B1[0][1] + pY_mt*B1[1][1] + pZ_mt*B1[2][1] + S[1];
   C3_bolt[2] = pX_mt*B1[0][2] + pY_mt*B1[1][2] + pZ_mt*B1[2][2] + S[2];
   if (flush_true == 1) {//Flush with the surface is a special case
      //Convert from the P0 (pivot) to the P1 frame
      pX_mp = XP[0][0]*pointX + XP[1][0]*pointY + XP[2][0]*pointZ;
      pY_mp = XP[0][1]*pointX + XP[1][1]*pointY + XP[2][1]*pointZ;
      pZ_mp = XP[0][2]*pointX + XP[1][2]*pointY + XP[2][2]*pointZ;
      //Convert from the P1 to the FIXED (F) frame
      pX_f = pX_mp*P[0][0] + pY_mp*P[1][0] + pZ_mp*P[2][0];
      pY_f = pX_mp*P[0][1] + pY_mp*P[1][1] + pZ_mp*P[2][1];
      pZ_f = pX_mp*P[0][2] + pY_mp*P[1][2] + pZ_mp*P[2][2];
      //Convert from the FIXED (F) to the BOLT (b) frame
      C3_bolt[0] = pX_f * R[0][0] + pY_f * R[1][0] + pZ_f * R[2][0] + 0.00;
      C3_bolt[1] = pX_f * R[0][1] + pY_f * R[1][1] + pZ_f * R[2][1] + 0.00;
      C3_bolt[2] = pX_f * R[0][2] + pY_f * R[1][2] + pZ_f * R[2][2] + 0.05;
   }
   return 0;
}
```

```
double bolt_to_fixed(double pointX, double pointY, double pointZ)
{//points X,Y,Z are measured from the BOLT (b) frame
   double  pX_p1,pY_p1,pZ_p1,pX_p0,pY_p0,pZ_p0;
   //Convert from the BOLT (b) to the FIXED (F) frame
   pX_f = pointX*R[0][0] + pointY*R[1][0] + pointZ*R[2][0];
   pY_f = pointX*R[0][1] + pointY*R[1][1] + pointZ*R[2][1];
   pZ_f = pointX*R[0][2] + pointY*R[1][2] + pointZ*R[2][2] + 0.05;
   //Convert from the FIXED (F) to the P1 frame
   pX_p1= pX_f*P[0][0] + pY_f*P[0][1] + pZ_f*P[0][2] - (P[0][0]*x[0] +
                                                        P[0][1]*y[0] + P[0][2]*z[0]);
   pY_p1= pX_f*P[1][0] + pY_f*P[1][1] + pZ_f*P[1][2] - (P[1][0]*x[0] +
                                                        P[1][1]*y[0] + P[1][2]*z[0]);
   pZ_p1= pX_f*P[2][0] + pY_f*P[2][1] + pZ_f*P[2][2] - (P[2][0]*x[0] +
                                                        P[2][1]*y[0] + P[2][2]*z[0]);
   //Convert from the P1 to the P0 (pivot) frame
   pX_p0= pX_p1*XP[0][0] + pY_p1*XP[0][1] + pZ_p1*XP[0][2];
   pY_p0= pX_p1*XP[1][0] + pY_p1*XP[1][1] + pZ_p1*XP[1][2];
   pZ_p0= pX_p1*XP[2][0] + pY_p1*XP[2][1] + pZ_p1*XP[2][2];
   //Convert from the P0 (pivot) to the FIXED (F) frame
   pX_f = pX_p0*P[0][0] + pY_p0*P[1][0] + pZ_p0*P[2][0] + x[0];
   pY_f = pX_p0*P[0][1] + pY_p0*P[1][1] + pZ_p0*P[2][1] + y[0];
   pZ_f = pX_p0*P[0][2] + pY_p0*P[1][2] + pZ_p0*P[2][2] + z[0];
   return 0;
}

double bolt_to_pivot(double pointX, double pointY, double pointZ)
{//points X,Y,Z are measured in the BOLT (b) frame
   double  pX_c1,pY_c1,pZ_c1,pX_c0,pY_c0,pZ_c0,pX_p1,pY_p1,pZ_p1;
   //Convert from the BOLT (b) to the FIXED (f) frame
   pX_f  = pointX*R1[0][0] + pointY*R1[0][1] + pointZ*R1[0][2];
   pY_f  = pointX*R1[1][0] + pointY*R1[1][1] + pointZ*R1[1][2];
   pZ_f  = pointX*R1[2][0] + pointY*R1[2][1] + pointZ*R1[2][2] + T1[2];
   //Convert from the FIXED (f) to the C1 frame
   pX_c1 = pX_f*Q[0][0] + pY_f*Q[0][1] + pZ_f*Q[0][2] - (Q[0][0]*U[0] +
                                                         Q[0][1]*U[1] + Q[0][2]*U[2]);
   pY_c1 = pX_f*Q[1][0] + pY_f*Q[1][1] + pZ_f*Q[1][2] - (Q[1][0]*U[0] +
                                                         Q[1][1]*U[1] + Q[1][2]*U[2]);
   pZ_c1 = pX_f*Q[2][0] + pY_f*Q[2][1] + pZ_f*Q[2][2] - (Q[2][0]*U[0] +
                                                         Q[2][1]*U[1] + Q[2][2]*U[2]);
   //Convert from the C1 to the C0 (tang) frame
   pX_c0 = pX_c1*M[0][0] + pY_c1*M[1][0] + pZ_c1*M[2][0];
   pY_c0 = pX_c1*M[0][1] + pY_c1*M[1][1] + pZ_c1*M[2][1];
   pZ_c0 = pX_c1*M[0][2] + pY_c1*M[1][2] + pZ_c1*M[2][2];
   //Convert from C0 (tang) to the P1 frame
   pX_p1 = pX_c0*P[0][0] + pY_c0*P[0][1] + pZ_c0*P[0][2];
   pY_p1 = pX_c0*P[1][0] + pY_c0*P[1][1] + pZ_c0*P[1][2];
   pZ_p1 = pX_c0*P[2][0] + pY_c0*P[2][1] + pZ_c0*P[2][2];
   //Convert from the P1 to the P0 (pivot) frame
   pX_p0 = pX_p1*XP[0][0] + pY_p1*XP[0][1] + pZ_p1*XP[0][2];
   pY_p0 = pX_p1*XP[1][0] + pY_p1*XP[1][1] + pZ_p1*XP[1][2];
   pZ_p0 = pX_p1*XP[2][0] + pY_p1*XP[2][1] + pZ_p1*XP[2][2];
   return 0;
}
```

```
double blt_vector(double loc_xc, double loc_yc, double loc_zc)
{
   double  b_max = 0.12445, rr=0.005, crr=0.0025;
   double  b_min = 0.12445 - 5.0*(0.05)*sqrt(3.0)/16.0;
   double  K = sqrt(3.0)*0.05/(2.0*pi), L = 0.05/(2.0*pi);
   double  Tb_i, Tb_j, mag_Tb, BMt_i, BMt_j, BMt_k;
   double  theta, angle, ang_ini, theta_ini=0.0 ,Bi_mt, Bj_mt, Bk_mt;
   double  Bi_t,Bj_t,Bk_t,Bi_mp,Bj_mp,Bk_mp,Bi_p,Bj_p,Bk_p,Bi_c1,Bj_c1,Bk_c1;

   if ((loc_xc >= 0.0) && (loc_yc >= 0.0)) {       // Quadrant I
      angle = atan(loc_yc/loc_xc);
   } else if ((loc_xc < 0.0) && (loc_yc >= 0.0)) {    // Quadrant II
      angle = pi + atan(loc_yc/loc_xc);
   } else if ((loc_xc < 0.0) && (loc_yc < 0.0)) {     // Quadrant III
      angle = pi + atan(loc_yc/loc_xc);
   } else if ((loc_xc >= 0.0) && (loc_yc < 0.0)) {    // Quadrant IV
      angle = 2.0*pi + atan(loc_yc/loc_xc);
   }
   if (loc_zc <= 0.0) {//Contact occurs at the base edge
      loc_zc = 0.0;
   }
   if (loc_zc > 0.0) {//Contact occurs above the base edge
      angle = angle + loc_zc*2.0*pi/0.05;
      if (angle > 2.0*pi) {
         angle = angle - 2.0*pi;
      }
   }
   theta = angle*180.0/pi;
   if ((theta >= 0.0) && (theta <= 34.14202146)) {//+ Flank
      ang_ini = 62.76952127*pi/180.0;
      Tb_i = -K*cos(angle) - (b_max - 0.5*crr - K*(angle+ang_ini))*sin(angle);
      Tb_j = -K*sin(angle) + (b_max - 0.5*crr - K*(angle+ang_ini))*cos(angle);
   } else if ((theta > 34.14202146) && (theta <= 65.31893600)) {//+ Root Fillet
      ang_ini = 34.14202146*pi/180.0;
      Tb_i = -(L*(sqrt(3)*rr/2.0 -L*(angle-ang_ini))*cos(angle))/sqrt(pow(rr,2)-
              pow((sqrt(3)*rr/2.0 -L*(angle-ang_ini)),2)) - (b_min + rr –
              sqrt(pow(rr,2)-pow((sqrt(3)*rr/2.0 -L*(angle-
              ang_ini)),2)))*sin(angle);
      Tb_j = -(L*(sqrt(3)*rr/2.0 -L*(angle-ang_ini))*sin(angle))/sqrt(pow(rr,2)-
              pow((sqrt(3)*rr/2.0 -L*(angle-ang_ini)),2)) + (b_min + rr –
              sqrt(pow(rr,2)-pow((sqrt(3)*rr/2.0 -L*(angle-
              ang_ini)),2)))*cos(angle);
   } else if ((theta > 65.31893600) && (theta <= 113.74971662)) {// Root
      Tb_i = -b_min*sin(angle);
      Tb_j =  b_min*cos(angle);
   } else if ((theta > 113.74971662) && (theta <= 144.92663116)) {//- RF
      ang_ini = 113.74971662*pi/180.0;
      Tb_i = (pow(L,2)*(angle-ang_ini)*cos(angle))/sqrt(pow(rr,2)-
              pow(L,2)*pow((angle-ang_ini),2)) - (b_min + rr - sqrt(pow(rr,2) –
              pow(L,2)*pow((angle-ang_ini),2)))*sin(angle);
      Tb_j = (pow(L,2)*(angle-ang_ini)*sin(angle))/sqrt(pow(rr,2)-
              pow(L,2)*pow((angle-ang_ini),2)) + (b_min + rr - sqrt(pow(rr,2) –
              pow(L,2)*pow((angle-ang_ini),2)))*cos(angle);
   } else if ((theta > 144.92663116) && (theta <= 241.83817389)) {//- Flank
      ang_ini = 144.92663116*pi/180.0;
      Tb_i = K*cos(angle) - (b_min + 0.5*rr + K*(angle-ang_ini))*sin(angle);
      Tb_j = K*sin(angle) + (b_min + 0.5*rr + K*(angle-ang_ini))*cos(angle);
   } else if ((theta > 241.83817389) && (theta <= 257.42663116)) {//- CF
      ang_ini = 241.83817389*pi/180.0;
      Tb_i = (L*(sqrt(3)*crr/2.0 -L*(angle-
              ang_ini))*cos(angle))/sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 –
              L*(angle-ang_ini)),2)) - (b_max - crr + sqrt(pow(crr,2)-
              pow((sqrt(3)*crr/2.0 -L*(angle-ang_ini)),2)))*sin(angle);
```

```
      Tb_j = (L*(sqrt(3)*crr/2.0 -L*(angle-
              ang_ini))*sin(angle))/sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 –
              L*(angle-ang_ini)),2)) + (b_max - crr + sqrt(pow(crr,2)-
              pow((sqrt(3)*crr/2.0 -L*(angle-ang_ini)),2)))*cos(angle);
   } else if ((theta > 257.42663116) && (theta <= 281.64202146)) {// Crest
      Tb_i = -b_max*sin(angle);
      Tb_j =  b_max*cos(angle);
   } else if ((theta > 281.64202146) && (theta <= 297.23047873)) {//+ CF
      ang_ini = 281.64202146*pi/180.0;
      Tb_i = -(pow(L,2)*(angle-ang_ini)*cos(angle))/sqrt(pow(crr,2)
              -pow(L,2)*pow((angle-ang_ini),2)) - (b_max - crr + sqrt(pow(crr,2)
              - pow(L,2)*pow((angle-ang_ini),2)))*sin(angle);
      Tb_j = -(pow(L,2)*(angle-ang_ini)*sin(angle))/sqrt(pow(crr,2)-
              pow(L,2)*pow((angle-ang_ini),2)) + (b_max - crr + sqrt(pow(crr,2)
              - pow(L,2)*pow((angle-ang_ini),2)))*cos(angle);
   } else if ((theta > 297.23047873) && (theta <= 360.0)) {//+ Flank
      ang_ini = 297.23047873*pi/180.0;
      Tb_i = -K*cos(angle) - (b_max - 0.5*crr - K*(angle-ang_ini))*sin(angle);
      Tb_j = -K*sin(angle) + (b_max - 0.5*crr - K*(angle-ang_ini))*cos(angle);
   }
   mag_Tb = sqrt(pow(Tb_i,2) + pow(Tb_j,2));
   //Dir. cosines of TANG-TO-EDGE vector as indicated in the BOLT (b) frame
   BMt_i = Tb_i/mag_Tb;
   BMt_j = Tb_j/mag_Tb;
   BMt_k = 0.0;

   //Convert from the BOLT (b) to the C1 frame
   Bi_mt = BMt_i*B1[0][0] + BMt_j*B1[0][1] + BMt_k*B1[0][2];
   Bj_mt = BMt_i*B1[1][0] + BMt_j*B1[1][1] + BMt_k*B1[1][2];
   Bk_mt = BMt_i*B1[2][0] + BMt_j*B1[2][1] + BMt_k*B1[2][2];
   //Convert from the C1 to the C0 (tang) frame
   Bi_t = Bi_mt*M[0][0] + Bj_mt*M[1][0] + Bk_mt*M[2][0];
   Bj_t = Bi_mt*M[0][1] + Bj_mt*M[1][1] + Bk_mt*M[2][1];
   Bk_t = Bi_mt*M[0][2] + Bj_mt*M[1][2] + Bk_mt*M[2][2];
   //Convert from the C0 (tang) to the P1 frame
   Bi_mp = Bi_t*P[0][0] + Bj_t*P[0][1] + Bk_t*P[0][2];
   Bj_mp = Bi_t*P[1][0] + Bj_t*P[1][1] + Bk_t*P[1][2];
   Bk_mp = Bi_t*P[2][0] + Bj_t*P[2][1] + Bk_t*P[2][2];
   //Convert from the P1 to the P0 (pivot) frame
   Bi_p = Bi_mp*XP[0][0] + Bj_mp*XP[0][1] + Bk_mp*XP[0][2];
   Bj_p = Bi_mp*XP[1][0] + Bj_mp*XP[1][1] + Bk_mp*XP[1][2];
   Bk_p = Bi_mp*XP[2][0] + Bj_mp*XP[2][1] + Bk_mp*XP[2][2];
   //Convert from the P0 (pivot) to the C0 (tang) frame
   Bi_c1 = Bi_p*P[0][0] + Bj_p*P[1][0] + Bk_p*P[2][0];
   Bj_c1 = Bi_p*P[0][1] + Bj_p*P[1][1] + Bk_p*P[2][1];
   Bk_c1 = Bi_p*P[0][2] + Bj_p*P[1][2] + Bk_p*P[2][2];
   //Convert from the C0 (tang) to the FIXED (F) frame
   BFt_i = Bi_c1*Q[0][0] + Bj_c1*Q[1][0] + Bk_c1*Q[2][0];
   BFt_j = Bi_c1*Q[0][1] + Bj_c1*Q[1][1] + Bk_c1*Q[2][1];
   BFt_k = Bi_c1*Q[0][2] + Bj_c1*Q[1][2] + Bk_c1*Q[2][2];
   return 0;
}
```

```
double nut_vector(double xc, double yc, double zc)
{
   double  n_max = 0.125, rr=0.0025, crr=0.005;
   double  n_min = 0.125 - 5.0*(0.05)*sqrt(3.0)/16.0;
   double  K = sqrt(3.0)*0.05/(2.0*pi), L = 0.05/(2.0*pi);
   double  Tn_i, Tn_j, mag_Tn;
   double  theta, nang, ang_ini;

   if ((xc >= 0.0) && (yc >= 0.0)) {       // Quadrant I
      nang = atan(yc/xc);
   } else if ((xc < 0.0) && (yc >= 0.0)) {    // Quadrant II
      nang = pi + atan(yc/xc);
   } else if ((xc < 0.0) && (yc < 0.0)) {     // Quadrant III
      nang = pi + atan(yc/xc);
   } else if ((xc >= 0.0) && (yc < 0.0)) {    // Quadrant IV
      nang = 2.0*pi + atan(yc/xc);
   }
   if ((zc-0.50) < 0.0) {
      nang = nang - (0.05-zc)*2.0*pi/0.05;
      if (nang < 0.0) {
         nang = nang + 2.0*pi;
      }
   }
   theta = nang*180.0/pi;
   if ((theta >= 0.0) && (theta <= 60.27536811)) {//- Flank
      ang_ini = 36.63617462*pi/180.0;
      Tn_i = K*cos(nang) - (n_min + 0.5*crr + K*(nang+ang_ini))*sin(nang);
      Tn_j = K*sin(nang) + (n_min + 0.5*crr + K*(nang+ang_ini))*cos(nang);
   } else if ((theta > 60.27536811) && (theta <= 75.86382538)) {//- Root Fillet
      ang_ini = 60.27536811*pi/180.0;
      Tn_i = (L*(sqrt(3)*rr/2.0 - L*(nang-ang_ini))*cos(nang))/sqrt(pow(rr,2)-
              pow((sqrt(3)*rr/2.0 -L*(nang-ang_ini)),2)) - (n_max - rr +
              sqrt(pow(rr,2)-pow((sqrt(3)*rr/2.0 -L*(nang-
              ang_ini)),2)))*sin(nang);
      Tn_j = (L*(sqrt(3)*rr/2.0 - L*(nang-ang_ini))*sin(nang))/sqrt(pow(rr,2)-
              pow((sqrt(3)*rr/2.0 -L*(nang-ang_ini)),2)) + (n_max - rr +
              sqrt(pow(rr,2)-pow((sqrt(3)*rr/2.0 -L*(nang-
              ang_ini)),2)))*cos(nang);
   } else if ((theta > 75.86382538) && (theta <= 100.07921568)) {// Root
      Tn_i = -n_max*sin(nang);
      Tn_j =  n_max*cos(nang);
   } else if ((theta > 100.07921568) && (theta <= 115.66767295)) {//+ RF
      ang_ini = 100.07921568*pi/180.0;
      Tn_i = -(pow(L,2)*(nang-ang_ini)*cos(nang))/sqrt(pow(rr,2)
              -pow(L,2)*pow((nang-ang_ini),2)) - (n_max - rr + sqrt(pow(rr,2)-
              pow(L,2)*pow((nang-ang_ini),2)))*sin(nang);
      Tn_j = -(pow(L,2)*(nang-ang_ini)*sin(nang))/sqrt(pow(rr,2)-
              pow(L,2)*pow((nang-ang_ini),2)) + (n_max - rr + sqrt(pow(rr,2)-
              pow(L,2)*pow((nang-ang_ini),2)))*cos(nang);
   } else if ((theta > 115.66767295) && (theta <= 212.57921568)) {//+ Flank
      ang_ini = 115.66767295*pi/180.0;
      Tn_i = -K*cos(nang) - (n_max - 0.5*rr - K*(nang-ang_ini))*sin(nang);
      Tn_j = -K*sin(nang) + (n_max - 0.5*rr - K*(nang-ang_ini))*cos(nang);
   } else if ((theta > 212.57921568) && (theta <= 243.75613022)) {//+ CF
      ang_ini = 212.57921568*pi/180.0;
      Tn_i = -(L*(sqrt(3)*crr/2.0 -L*(nang-ang_ini))*cos(nang))/sqrt(pow(crr,2)-
              pow((sqrt(3)*crr/2.0 -L*(nang-ang_ini)),2)) - (n_min + crr –
              sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 -L*(nang-
              ang_ini)),2)))*sin(nang);
      Tn_j = -(L*(sqrt(3)*crr/2.0 -L*(nang-ang_ini))*sin(nang))/sqrt(pow(crr,2)-
              pow((sqrt(3)*crr/2.0 -L*(nang-ang_ini)),2)) + (n_min + crr –
              sqrt(pow(crr,2)-pow((sqrt(3)*crr/2.0 -L*(nang-
              ang_ini)),2)))*cos(nang);
```

```
   } else if ((theta > 243.75613022) && (theta <= 292.18691084)) {// Crest
      Tn_i = -n_min*sin(nang);
      Tn_j =  n_min*cos(nang);
   } else if ((theta > 292.18691084) && (theta <= 323.36382538)) {//- CF
      ang_ini = 292.18691084*pi/180.0;
      Tn_i = (pow(L,2)*(nang-ang_ini)*cos(nang))/sqrt(pow(crr,2)-
              pow(L,2)*pow((nang-ang_ini),2)) - (n_min + crr - sqrt(pow(crr,2) –
              pow(L,2)*pow((nang-ang_ini),2)))*sin(nang);
      Tn_j = (pow(L,2)*(nang-ang_ini)*sin(nang))/sqrt(pow(crr,2)-
              pow(L,2)*pow((nang-ang_ini),2)) + (n_min + crr - sqrt(pow(crr,2) –
              pow(L,2)*pow((nang-ang_ini),2)))*cos(nang);
   } else if ((theta > 323.36382538) && (theta <= 360.0)) {//- Flank
      ang_ini = 323.36382538*pi/180.0;
      Tn_i = K*cos(nang) - (n_min + 0.5*crr + K*(nang-ang_ini))*sin(nang);
      Tn_j = K*sin(nang) + (n_min + 0.5*crr + K*(nang-ang_ini))*cos(nang);
   }
   mag_Tn = sqrt(pow(Tn_i,2) + pow(Tn_j,2));
   NWt_i = Tn_i/mag_Tn;
   NWt_j = Tn_j/mag_Tn;
   NWt_k = 0.0;
   return 0;
}
```

```
double common_normal(double BFt_i, double BFt_j, double BFt_k, double NWt_i,
                                                    double NWt_j, double NWt_k)
{
   double  BX_cos, NX_cos, CN_i, CN_j, CN_k, com_norm_mag;
   //Get the initial theta_X angles for the BOLT and NUT tangent vectors
   BX_cos = acos(BFt_i);
   NX_cos = acos(NWt_i);
   //Adjust BX_cos & NX_cos to account for counter-clockwise measurement
   //since we want the normal to point away from the NUT (positive direction)
   //The order of the x-product must be set before the X-product can be computed
   if ((acos(BFt_i) > pi/2.0) && (acos(BFt_j) > pi/2.0)) {//3rd Quadrant
      BX_cos = 2.0*pi - acos(BFt_i);
   } else if ((acos(BFt_i) <= pi/2.0) && (acos(BFt_j) > pi/2.0)) {//4th Quadrant
      BX_cos = 2.0*pi - acos(BFt_i);
   }
   if ((acos(NWt_i) > pi/2.0) && (acos(NWt_j) > pi/2.0)) {//3rd Quadrant
      NX_cos = 2.0*pi - acos(NWt_i);
   } else if ((acos(NWt_i) <= pi/2.0) && (acos(NWt_j) > pi/2.0)) {//4th Quadrant
      NX_cos = 2.0*pi - acos(NWt_i);
   }
   //BX_cos & NX_cos are measured CCW.  The order in the X-products is defined
   //for the first 180 degrees (pi) of separation between BX_cos & NX_cos
   if ((BX_cos > NX_cos) && ((BX_cos - NX_cos) > pi)) {//BFt X NWt
      CN_i = BFt_j*NWt_k - BFt_k*NWt_j;
      CN_j = BFt_k*NWt_i - BFt_i*NWt_k;
      CN_k = BFt_i*NWt_j - BFt_j*NWt_i;
   } else if ((BX_cos > NX_cos) && ((BX_cos - NX_cos) < pi)) {//NWt X BFt
      CN_i = NWt_j*BFt_k - NWt_k*BFt_j;
      CN_j = NWt_k*BFt_i - NWt_i*BFt_k;
      CN_k = NWt_i*BFt_j - NWt_j*BFt_i;
   }
   if ((NX_cos > BX_cos) && ((NX_cos - BX_cos) > pi)) {//NWt X BFt
      CN_i = NWt_j*BFt_k - NWt_k*BFt_j;
      CN_j = NWt_k*BFt_i - NWt_i*BFt_k;
      CN_k = NWt_i*BFt_j - NWt_j*BFt_i;
   } else if ((NX_cos > BX_cos) && ((NX_cos - BX_cos) < pi)) {//BFt X NWt
      CN_i = BFt_j*NWt_k - BFt_k*NWt_j;
      CN_j = BFt_k*NWt_i - BFt_i*NWt_k;
      CN_k = BFt_i*NWt_j - BFt_j*NWt_i;
   }
   //Normalize the I, J, & K values of the common normal vector
   com_norm_mag = sqrt(pow(CN_i,2) + pow(CN_j,2) + pow(CN_k,2));
   Com_Norm_i = CN_i/com_norm_mag;
   Com_Norm_j = CN_j/com_norm_mag;
   Com_Norm_k = CN_k/com_norm_mag;
   return 0;
}
```