A FINAL REPORT FOR: EVALUATION OF THE SHUTTLE REMOTE MANIPULATOR

Otherwise known as:

DETERMINATION OF TARGET POSE WITHOUT SENSOR REFLECTION

Prepared for: Leo Monford, NASA-JSC

by
L. J. Everett
and
R. C. Redfield
Mechanical Engineering Department
Texas A&M University
College Station, Texas 77843-3123

September 23, 1993

INTRODUCTION

The objective initially proposed was to analyze RMS performance data collected during a Shuttle Flight. The data was to consist of video TRAC data collected via a video recorder. Unfortunately, the flight never collected the data due to higher priority experiments superseding it. As a result, the research team at Texas A&M was directed to work on several other pressing issues regarding the TRAC sensor. All but one of these issues have been reported to the contract monitor (Mr. Leo Monford) earlier in the form of periodic status reports.

In fulfilment of the grant conditions, the last issue investigated is being reported as the final report. Ordinarily, a TRAC sensor determines the orientation of an object by analyzing the image reflected from a mirror target. The concern addressed by this report is to develop a method for using the TRAC sensor when the target does not reflect a usable image.

Determination of Target Pose without Sensor Reflection

Objective

Given two objects or structures with relative pose between them, an object with an integral, specially configured target and an object with a vision sensor, determine the position and orientation of the target relative to the sensor. No mirror reflection of the sensor off the target and back to the sensor is available.

Hardware

The target is a configuration of four distinct objects (LED arrays, retro-reflectors, etc.) that can be independently recognized. The vision system sensor views the target and a computer algorithm determines the azimuth and elevation of each LED array relative to the sensor coordinate system.

Coordinates

In Figure 1 the vision system is at the origin, O, of the right-hand coordinate system fixed to the sensor object. The target is at P. The forward axis of the sensor, which is the optical axis of the vision system, is in the z direction. Upward is the y direction and to the left is the x direction. The position vector from O to P is $\mathbf{R}$ which has length r. Azimuth angle, θ, is defined about the y axis measured from positive z; elevation angle, α, is about x in the negative direction measured from the x-z plane.

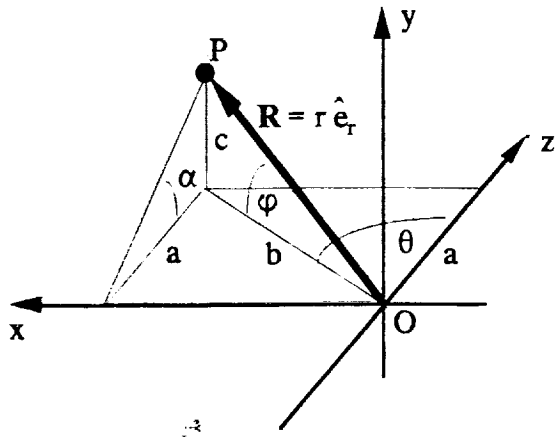

Figure 1 - Coordinate system

A working angle φ is defined as the angle between the position vector and the *x-z* plane.

Cartesian geometry shows that if

$$\mathbf{R} = p_x\hat{i} + p_y\hat{j} + p_z\hat{k}, \tag{1}$$

then

$$\begin{aligned} p_x &= r\cos\varphi\sin\theta \\ p_y &= r\sin\varphi \\ p_z &= r\cos\varphi\cos\theta \end{aligned} \tag{2}$$

where

$$\varphi = \tan^{-1}\left[\frac{c}{b}\right] = \tan^{-1}\left[\frac{c}{a}\bullet\frac{a}{b}\right\} = \tan^{-1}\left[\tan\alpha\cos\theta\right] \tag{3}$$

Full geometry

The entire geometry including the 4 target objects is shown in Figure 2.

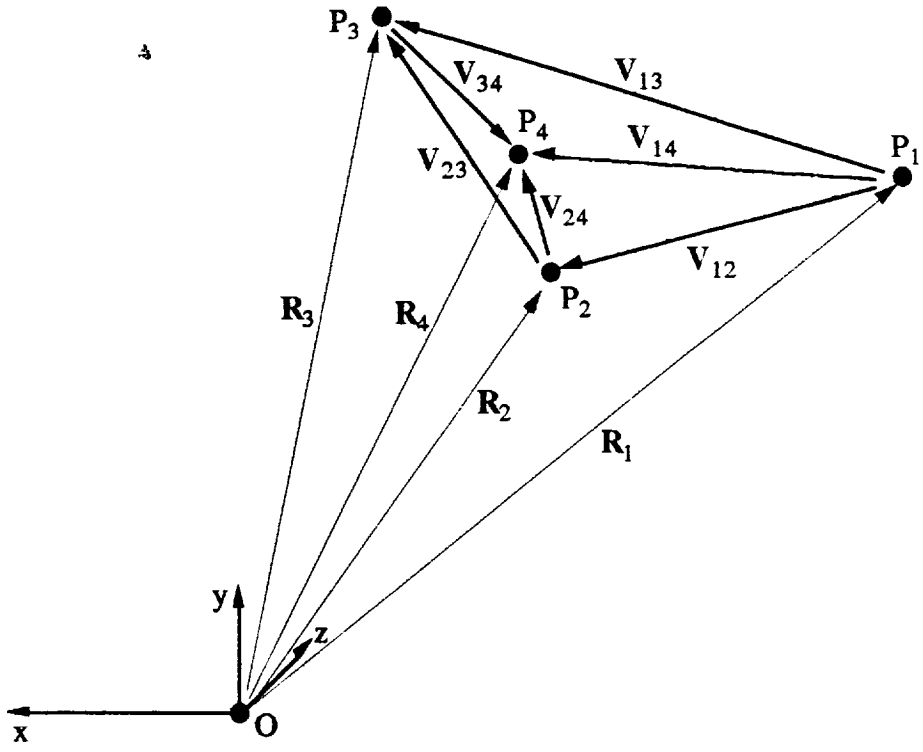

Figure 2 - Geometry of tracking scenario

The origin of the *x-y-z* system is the vision system, the 4 black dots are the target objects P_i, the $\mathbf{R}_i$

are the radius vectors between the sensor object and the individual target objects, and the $\mathbf{V}_{ij}$ are the vectors connecting the points in the target. The azimuth and elevation of each radius vector is known, but its length is not. The geometry of the target is known, that is the lengths of the $\mathbf{V}_{ij}$ and the angles between the $\mathbf{V}_{ij}$.

Determination of target pose

If the lengths of the $\mathbf{R}_i$ (r_i) can be found, the points P_i can be calculated and the target pose will be known. To the investigators' knowledge, any solution to this problem involves the simultaneous solution of a set of nonlinear algebraic equations. To solve this set of equations, an initial guess of the r_i converges to a solution by iteration.

Two approaches are taken to this problem, both of which are detailed in the next section of this paper. The "direct" approach takes a guess at one of the radius vector lengths, r_1, and calculates the remaining radius vectors based on geometry. If these vectors determine a target configuration geometry that agrees with the actual geometry to some tolerance, a solution is found, otherwise a new guess at r_1 is taken. The second approach guesses all four radius vectors, applies the geometric nonlinear relations, $f_i(r_1, r_2, r_3, r_4) = f_i(\underline{r}) = \varepsilon$, and calculates a residual that indicates error. The four vector lengths are simultaneously altered with a Newton-Raphson technique for sets of nonlinear equations.

Direct approach

An initial guess of one of the radius vector lengths, r_i, allows the calculation of a second radius vector length, r_j from the geometry of Figure 3.

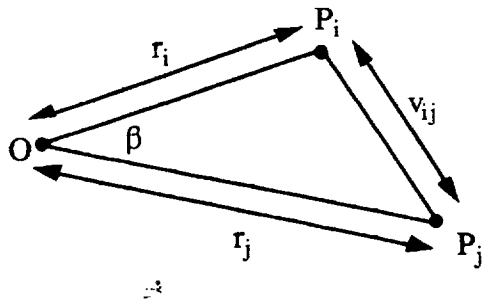

Figure 3 - Guess at radius vectors

The law of cosines is applied with a guess for r_i, and a known v_{ij} and β. β is the angle subtended by $\mathbf{R}_i$ and $\mathbf{R}_j$. The cosine of β is found as the dot product of the radius unit vectors, $\hat{e}_{ri} \cdot \hat{e}_{rj}$.

The geometry yields zero, one, or two real r_j for each given r_i as seen in the quadratic equation of equation 4.

$$r_j^2 - 2r_i \cos\beta\, r_j + r_i^2 - v_{ij}^2 = 0 \tag{4}$$

In Figure 4, four possible cases, A-D, are shown for the solution of equation 4. $\mathbf{R}_i$ and $\mathbf{R}_j$ are two radius vectors of known directions but unknown lengths and $\mathbf{V}_{ij}$ is the target vector between $\mathbf{R}_i$ and $\mathbf{R}_j$ with known length but unknown direction. The circles all have a radius of $v_{ij} = |\mathbf{V}_{ij}|$. a-d are distances along $\mathbf{R}_i$ from the origin, *O*.

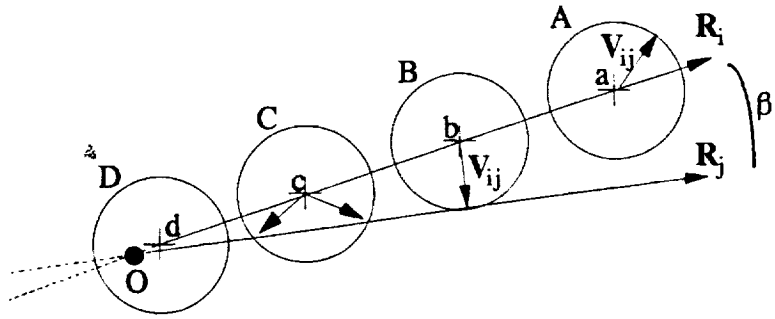

Figure 4 - Possible radius vectors

Case A shows a guessed length for $\mathbf{R}_i$ of *a*. No direction of $\mathbf{V}_{ij}$ allows contact with $\mathbf{R}_j$ so the guessed length is too long and the solution to equation 4 would yield complex numbers. Case B shows a guess length for $\mathbf{R}_i$ of *b* that allows one solution, vector $\mathbf{V}_{ij}$ is normal to $\mathbf{R}_j$. The quadratic equation would result in a repeated root. Case C shows a length for $\mathbf{R}_i$ of *c* that gives two solutions to the direction of $\mathbf{V}_{ij}$ and Case D shows that one of the two solutions implies a negative $\mathbf{R}_j$ which is not physically possible in the targeting scenario.

For the solution of all four radius vectors, the following steps are taken which will be detailed afterwards:

a) Determine valid range and increment for r_1.
b) Cycle through the r_1 range from maximum to minimum.
   1) For each r_1 calculate possible r_2, r_3, and r_4.
   2) For each combination of $r_1 \ldots r_4$ determine all v_{ij} from equation 4.
   3) Calculate error as difference between v_{ij} from step b2 and actual target v_{ij}.
   4) Record $\underline{r}_{min}$ that minimizes error
   5) Pick new r_1 and repeat b.
c) Determine new range near $r_{1\,min}$; repeat step b
d) Stop when r_1 increment is less than tolerance.

The maximum r_1 value is determined such that given v_{ij}, real r_2, r_3, and r_4 exist. Case B in Figure 4 shows the maximum r_i that allows a real solution for r_j. If r_i>b, there is no solution. The result is that

$$r_{i\,max} = \frac{v_{ij}}{\sin\beta} \tag{5}$$

This maximum is determined for each of the 3 vectors r_2-r_4 and the minimum of these is retained.

The minimum r_1 is initially taken to be zero and its range is arbitrarily divided into *n* increments of length $r_{inc} = r_{i\,max}/n$. r_1 is cycled through this range from maximum to minimum and for each r_1 the r_2-r_4 are determined with equation 4. Since there are usually 2 r_j (j=2,3,4) for each r_1, there are 2^3 combinations of radius vectors for each r_1.

Each of these combinations of r_i, r_j is inserted into equation 4 for i,j=2,3; 2,4; 3,4 and the v_{ij} are calculated. The closer these calculated v_{ij} are to the actual v_{ij} the better the estimate of $\underline{r}$. The difference between the two v_{ij} is termed "error." For the r_1 that gives a minimum error in the current range (r_1^*), a new range for r_1 is chosen to be $r_1^* + r_{inc}$ to $r_1^* - r_{inc}$. The maximum value is kept no more than $R_{1\,max}$ and the minimum value is never less than zero. A new increment is chosen by dividing the new range into *n* divisions. When the r_{inc} is less than some tolerance and the minimum error is found in the current range, the solution is in hand.

The FORTRAN code that implements this strategy is in Appendix A.

Newton-Raphson approach

This method solves a set of 4 nonlinear equations where the β_{ij} and v_{ij} are known and the r_i are unknown. Equation (6) is the set of equations; with correct $\underline{r}$, the $f_i = 0$.

$$
\begin{aligned}
f_1(r_1, r_2, r_3, r_4) &= r_1^2 + r_2^2 - 2r_1 r_2 \cos\beta_{12} - v_{12}^2 \\
f_2(r_1, r_2, r_3, r_4) &= r_2^2 + r_3^2 - 2r_2 r_3 \cos\beta_{23} - v_{23}^2 \\
f_3(r_1, r_2, r_3, r_4) &= r_3^2 + r_4^2 - 2r_3 r_4 \cos\beta_{34} - v_{34}^2 \\
f_4(r_1, r_2, r_3, r_4) &= r_4^2 + r_1^2 - 2r_4 r_1 \cos\beta_{41} - v_{41}^2
\end{aligned} \tag{6}
$$

Press et al. [1986] outline the method where a truncated Taylor series expansion approximates the f_i with the second and higher order terms dropped.

$$
f_i(\underline{r} + \delta\underline{r}) = f_i(\underline{r}) + \sum_{j=1}^{4} \frac{\partial f_i}{\partial r_j} \delta r_j + \cancel{O(\delta\underline{r}^2)} \tag{7}
$$

For an initial guess of $\underline{r}$, the left had sides of equation (6) are not usually zero. From equation (7), the $\delta\underline{r}$ are required that make $f_i(\underline{r} + \delta\underline{r})$ zero given an initial $f_i(\underline{r})$. Equation (7) is a linear set of equations in $\delta\underline{r}$ that can be solved with *LU* decomposition or any other method. As long as the initial guess of $\underline{r}$ is close enough to the solution, the Newton-Raphson schemes converges nicely. Otherwise, the scheme may never converge or perhaps converge to the wrong solution.

In practice, the algorithm converges very well in a local neighborhood around the correct solution. If the guess is not in the neighborhood the scheme either does not converge or it converges to a wrong solution. The difficulty is in defining the neighborhood. In testing, sometimes an initial guess for $\underline{r}$ of 50% of the solution would converge in less than 50 iterations. Other times, an initial guess within 5% of the solution would never converge. This uncertainty leaves the Newton-Raphson lacking without further inquiry.

The FORTRAN code for this scheme is in Appendix B.

References

Press, W. H., Flannery, B. P., Teukolsky, S A., and Vetterling, W. T., 1986, *Numerical Recipes: The Art of Scientific Computing,* Cambridge University Press, Cambridge, pp. 269.

Appendix A - FORTRAN code for direct approach

```
c*******************************************************************************
c                                                              target.for
      program target
c
c*******************************************************************************
c
c Routine find Ri for 4 vectors by guessing R1, calculating R2, R3, and R4
c and applying law of cosines to find residual errors.  Ri that minimizes
c errors is best guess.

c       1         2         3         4         5         6         7
c234567890123456789012345678901234567890123456789012345678901234567890123456789012

      real r2(2),r3(2), r4(2), err(2,2)
      real t(4,4), az(4), el(4), phi(4), c(4,4), s2(4,4)
      real merit, min1, min2
      open (unit=10, file='target.inp', status='old')

c read input data from file TARGET.INP

      read (10,*) toldr1, tolmin                          ! Converg tolerances
      tmax = 0                                            ! Initialize max value
      do 5, i=1, 3
        do 5, j=i+1,4
        read (10,*) t(i,j)
        t(j,i) = t(i,j)                                   ! Symmetric
    5 tmax = amax1(tmax, t(i,j))                          ! Max target vector
      read (10,*) (az(i), i=1, 4)                         ! Azimuth angles
      read (10,*) (el(i), i=1, 4)                         ! Elevation angles

c convert to radians from degrees

      pi = 3.14159                                        ! Calculate pi
      do 10 i=1, 4
        az(i) = az(i) * pi/180
        el(i) = el(i) * pi/180
   10 phi(i) = atan( tan(el(i)) * cos(az(i)) )

c calculate cos and sin of angles between radius vectors (cos = e1 dot e2)

      do 20, i=1, 3
        do 20, j=i+1, 4
          c(i,j) = sin(phi(i)) * sin(phi(j)) +
     *     cos(phi(i))*cos(az(i)) * cos(phi(j))*cos(az(j)) +
     *     cos(phi(i))*sin(az(i)) * cos(phi(j))*sin(az(j))
          c(j,i) = c(i,j)
          write(6,*) 'c,t ', i,j, c(i,j), t(i,j)          ! echo input
          s2(i,j) = 1 - c(i,j)**2                         ! sine squared of angle
   20 s2(j,i) = s2(i,j)                                   ! Symmetric

c limits on ri values

      rmax = 1.e20                                        ! initial rmax
      do 30, j=2, 4
        tmp = sqrt( s2(1,j) )                             ! sine of angle 1-i
        if (tmp .eq. 0) tmp=1.e-20                        ! no / by zero
        tmp = t(1,j)/tmp                                  ! max ri due to r(j)
```

```
          if (tmp .eq. 0) tmp=1.e-20                    ! no / by zero
          tmp = t(1,j)/tmp                              ! max r1 due to r(j)
          rmax = amin1(rmax, tmp)                       ! max r1 is min of 4
   30   continue

c loop to find radius vector lengths - set increment based on limits

        rmin = 0.                                       ! r1 minimum
        rup = rmax
        dr1 = rmax / 10                                 ! r1 increment
        min1 = 1.e20                                    ! initial minimums
        min2 = 1.e20
   35   write(6,*) ' '
        write(6,*) 'dr1-> ', dr1, ' rup-> ',rup,        ! counter
     *             ' rmin-> ', rmin
        do 80, r1 =  rup, rmin, -dr1                    ! loop r1s

c two values of r2,r3,and r4 for each r1

        tmp2 = t(1,2)**2 - r1**2 * s2(1,2)
        if (tmp2 .lt. 0.) tmp2 = 0.
        tmp3 = t(1,3)**2 - r1**2 * s2(1,3)
        if (tmp3 .lt. 0.) tmp3 = 0.
          r2(1) = r1 * c(1,2) + sqrt(tmp2)
          r2(2) = r1 * c(1,2) - sqrt(tmp2)
          r3(1) = r1 * c(1,3) + sqrt(tmp3)
          r3(2) = r1 * c(1,3) - sqrt(tmp3)

c check 2-3 vector, 4 combinations

          do 60, i=1,2
            do 60 j=1,2
              err(i,j) = r2(i)**2 + r3(j)**2 -
     *                   2*r2(i)*r3(j)*c(2,3) - t(2,3)**2

              tmp4 = t(1,4)**2 - r1**2 * s2(1,4)
              if (tmp4 .lt. 0.) tmp4 = 0.
              r4(1) = r1 * c(1,4) + sqrt(tmp4)
              r4(2) = r1 * c(1,4) - sqrt(tmp4)

              do 55 k=1,2
                err24 = r2(i)**2 + r4(k)**2 -           ! vector 2-4
     *                  2*r2(i)*r4(k)*c(2,4) - t(2,4)**2
                err34 = r3(j)**2 + r4(k)**2 -           ! vector 3-4
     *                  2*r3(j)*r4(k)*c(3,4) - t(3,4)**2

c merit is sum of absolute 2-3, 2-4, and 3-4 vector errors

                merit = abs(err24) + abs(err34) + abs(err(i,j))
                if (merit .lt. min1) then
                  min2 = min1                           ! save best
                  r1f2 = r1f1
                  r2f2 = r2f1
                  r3f2 = r3f1
                  r4f2 = r4f1
                  min1 = merit
                  r1f1 = r1
                  r2f1 = r2(i)
                  r3f1 = r3(j)
                  r4f1 = r4(k)
                else
```

```
              if (merit .lt. min2) then
                min2 = merit                          ! save second best
                r1f2 = r1
                r2f2 = r2(i)
                r3f2 = r3(j)
                r4f2 = r4(k)
              endif
            endif

55        continue
60      continue
80    continue

      write(6,*) '1-> ', r1f1, r2f1, r3f1, r4f1       ! output
      write(6,*) 'merit 1-> ',min1
      write(6,*) '2-> ', r1f2, r2f2, r3f2, r4f2
      write(6,*) 'merit 2-> ',min2

      if (dr1 .le. toldr1 .or. min1 .le. tolmin) stop ! within tol?

      rup = amax1(r1f1, r1f2) + dr1                   ! new max
      if (rup .gt. rmax) rup=rmax
      rmin = amin1(r1f1, r1f2) - dr1                  ! new min
      dr1 = (rup-rmin)/10.                            ! new increment
      go to 35                                        ! new range & increment

      end
```

Appendix B - FORTRAN code for Newton-Raphson approach

(Code in all CAPS is from Press [1986])

```
c ***************************************************************************
c                                                                      ROOT.FOR
      program root
c
c ***************************************************************************

c Program to find radius vector lengths for comet geometry

      real r(4), az(4), el(4), phi(4)
      include 'root.inc'                          ! Common for "usrfun"
c     common /dim/ t(4,4), c(4,4), s(4,4), betasum

c open input file

      open (unit=10, file='root.inp', status='old')

c read input data from file ROOT.DAT

      read(10,*) tolx, tolf, ntrial               ! Tolerance x & f, max. iters.
      do 5, i=1, 3
        do 5, j=i+1,4
          read (10,*) t(i,j)                      ! Target vector lengths
    5 t(j,i) = t(i,j)                             ! Symmetric
      read (10,*) (az(i), i=1, 4)                 ! Azimuth angles
      read (10,*) (el(i), i=1, 4)                 ! Elevation angles

c convert to radians from degrees

      pi = 3.14159                                ! PI
      do 10 i=1, 4
        az(i) = az(i) * pi/180
        el(i) = el(i) * pi/180
   10 phi(i) = atan( tan(el(i)) * cos(az(i)) )

c calculate cos and sin of angles between radius vectors (cos = e1 dot e2)

      do 20, i=1, 3
        do 20, j=i+1, 4
          c(i,j) = sin(phi(i)) * sin(phi(j)) +
     *     cos(phi(i))*cos(az(i)) * cos(phi(j))*cos(az(j)) +
     *     cos(phi(i))*sin(az(i)) * cos(phi(j))*sin(az(j))

          c(j,i) = c(i,j)                         ! Symmetric
          s(i,j) = sqrt( 1 - c(i,j)**2 )          ! Sine
   20   s(j,i) = s(i,j)                           ! Symmetric

c maximum radius vectors and initial r

      fac = 1.0
   22 do 30 i=1, 4
        rmax = 1.e20                              ! large to start
        do 25 j=1, 4
          if (j .eq. i) go to 25                  ! no angle here
```

```
            if (s(i,j) .eq. 0.) go to 25          ! no limit here
            test = t(i,j) / s(i,j)
            rmax = amin1 (rmax, test)
   25     continue
          r(i) = fac * rmax                       ! start at 80% max
   30   continue
        write (6,*) 'Initial r(i)'
        write (6,*) (r(i), i=1, 4)

c call solver

        nt = ntrial
        call mnewt(nt, r, 4, tolx, tolf)
        if (nt .eq. 0) then
          fac=fac - 0.1
          if (fac .eq. 0) stop                    ! plum run out
          go to 22
        endif

c check results

        do 50, i=1, 3
          do 50, j=i+1, 4
            tc = sqrt( r(i)**2 + r(j)**2 - 2*r(i)*r(j)*c(i,j) )
  50        write(6,*) 'in ',t(i,j),  'out ', tc

        write(6,*) 'Rs ', (r(i), i=1, 4)

        stop
        end
```

```
      SUBROUTINE MNEWT(NTRIAL,X,N,TOLX,TOLF)
      PARAMETER (NP=4)
      DIMENSION X(NP),ALPHA(NP,NP),BETA(NP),INDX(NP)
      DO 13  K=1,NTRIAL
        CALL USRFUN(X,ALPHA,BETA)
        ERRF=0.
        DO 11 I=1,N
          ERRF=ERRF+ABS(BETA(I))
11      CONTINUE
        IF(ERRF.LE.TOLF)RETURN
        CALL LUDCMP(ALPHA,N,NP,INDX,D)
        CALL LUBKSB(ALPHA,N,NP,INDX,BETA)
        ERRX=0.
        DO 12 I=1,N
          ERRX=ERRX+ABS(BETA(I))
          X(I)=X(I)+BETA(I)
12      CONTINUE
        IF(ERRX.LE.TOLX)RETURN
13    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LUDCMP(A,N,NP,INDX,D)
      PARAMETER (NMAX=100,TINY=1.0E-20)
      DIMENSION A(NP,NP),INDX(N),VV(NMAX)
      D=1.
      DO 12 I=1,N
        AAMAX=0.
        DO 11 J=1,N
          IF (ABS(A(I,J)).GT.AAMAX) AAMAX=ABS(A(I,J))
11      CONTINUE
        IF (AAMAX.EQ.0.) PAUSE 'Singular matrix.'
        VV(I)=1./AAMAX
12    CONTINUE
      DO 19 J=1,N
        IF (J.GT.1) THEN
          DO 14 I=1,J-1
            SUM=A(I,J)
            IF (I.GT.1)THEN
              DO 13 K=1,I-1
                SUM=SUM-A(I,K)*A(K,J)
13            CONTINUE
              A(I,J)=SUM
            ENDIF
14        CONTINUE
        ENDIF
        AAMAX=0.
        DO 16 I=J,N
          SUM=A(I,J)
          IF (J.GT.1)THEN
            DO 15 K=1,J-1
              SUM=SUM-A(I,K)*A(K,J)
15          CONTINUE
            A(I,J)=SUM
          ENDIF
          DUM=VV(I)*ABS(SUM)
          IF (DUM.GE.AAMAX) THEN
            IMAX=I
            AAMAX=DUM
          ENDIF
16      CONTINUE
        IF (J.NE.IMAX)THEN
          DO 17 K=1,N
            DUM=A(IMAX,K)
            A(IMAX,K)=A(J,K)
            A(J,K)=DUM
17        CONTINUE
          D=-D
          VV(IMAX)=VV(J)
        ENDIF
        INDX(J)=IMAX
        IF(J.NE.N)THEN
          IF(A(J,J).EQ.0.)A(J,J)=TINY
          DUM=1./A(J,J)
          DO 18 I=J+1,N
            A(I,J)=A(I,J)*DUM
18        CONTINUE
        ENDIF
19    CONTINUE
      IF(A(N,N).EQ.0.)A(N,N)=TINY
      RETURN
      END
```

```
      SUBROUTINE LUBKSB(A,N,NP,INDX,B)
      DIMENSION A(NP,NP),INDX(N),B(N)
      II=0
      DO 12 I=1,N
        LL=INDX(I)
        SUM=B(LL)
        B(LL)=B(I)
        IF (II.NE.0)THEN
          DO 11 J=II,I-1
            SUM=SUM-A(I,J)*B(J)
11        CONTINUE
        ELSE IF (SUM.NE.0.) THEN
          II=I
        ENDIF
        B(I)=SUM
12    CONTINUE
      DO 14 I=N,1,-1
        SUM=B(I)
        IF(I.LT.N)THEN
          DO 13 J=I+1,N
            SUM=SUM-A(I,J)*B(J)
13        CONTINUE
        ENDIF
        B(I)=SUM/A(I,I)
14    CONTINUE
      RETURN
      END
```

```
c ***************************************************************************
c                                                              USRFUN.FOR
      subroutine usrfun (r, alpha, beta)
c
c ***************************************************************************

      include 'root.inc'
      real alpha(4,4), beta(4), r(4)

c calculate functions and derivatives from law of cosines

      beta(1) = -( r(1)*r(1) + r(2)*r(2) -                          ! 1-2
     *             2*r(1)*r(2)*c(1,2) - t(1,2)*t(1,2) )
      alpha(1,1) = 2*r(1) - 2*c(1,2)*r(2)
      alpha(1,2) = 2*r(2) - 2*c(1,2)*r(1)
      alpha(1,3) = 0
      alpha(1,4) = 0

      beta(2) = -( r(1)*r(1) + r(3)*r(3) -                          ! 1-3
     *             2*r(1)*r(3)*c(1,3) - t(1,3)*t(1,3) )
      alpha(2,2) = 0
      alpha(2,3) = 2*r(3) - 2*c(1,3)*r(1)
      alpha(2,1) = 2*r(1) - 2*c(1,3)*r3
      alpha(2,4) = 0

      beta(3) = -( r(3)*r(3) + r(4)*r(4) -                          ! 3-4
     *             2*r(3)*r(4)*c(3,4) - t(3,4)*t(3,4) )
      alpha(3,3) = 2*r(3) -2*c(3,4)*r(4)
      alpha(3,4) = 2*r(4) -2*c(3,4)*r(3)
      alpha(3,1) = 0
      alpha(3,2) = 0

      beta(4) = -( r(4)*r(4) + r(2)*r(2) -                          ! 2-4
     *             2*r(4)*r(1)*c(2,4) - t(2,4)*t(2,4) )
      alpha(4,2) = 2*r(2) -2*c(2,4)*r(4)
      alpha(4,4) = 2*r(4) -2*c(2,4)*r(2)
      alpha(4,1) = 0
      alpha(4,3) = 0

      betasum = 0                                                   ! Convergence
      write(6,*) 'r->    ', (r(i), i=1, 4)
      write(6,*) 'beta-> ', (beta(i), i=1, 4)

      return
      end
```