**D. Implementation**

|  |
| --- |
| From all the measurement values x1,y1,z1to  xn,yn,zn, build the design matrix D.  Using the example file mag.txt with n = 25599, this produces a huge matrix of 10 rows x 25599 columns.    [https://sites.google.com/site/sailboatinstruments1/_/rsrc/1315617270526/step-1/matrix_d.jpg?height=233&width=320](https://sites.google.com/site/sailboatinstruments1/step-1/matrix_d.jpg?attredirects=0)    Simplified implementation example:    CStdioFile readFile; CString strFilePath = "C:\\MagCal\\mag.txt"; int nlines = 0; char buf[120];  readFile.Open(strFilePath, CFile::modeRead);    // count number of lines while(readFile.ReadString(buf, 100) != NULL)    nlines++;    // come back to beginning of file readFile.SeekToBegin();    // allocate memory for design matrix D double\* D = new double[10\*nlines];  int i;  double x, y, z; for( i = 0; i < nlines; i++) {     readFile.ReadString(buf, 100);     sscanf(buf, "%lf\t%lf\t%lf", &x, &y, &z);     D[i\*10] = x \* x;     D[i\*10+1] = y \* y;     D[i\*10+2] = z \* z;     D[i\*10+3] = 2.0 \* y \* z;     D[i\*10+4] = 2.0 \* x \* z;     D[i\*10+5] = 2.0 \* x \* y;     D[i\*10+6] = 2.0 \* x;     D[i\*10+7] = 2.0 \* y;     D[i\*10+8] = 2.0 \* z;     D[i\*10+9] = 1.0; } readFile.Close();    Now create the 10x10 symmetric matrix S by multiplying D by its own transpose DT:              D\*DT= S        [10 x nlines] \* [nlines x 10] = [10 x 10]                                         [m x k]     \*       [k x n]      =   [m x n]                                            [A]        \*         [B]T        =     [C]    This is done with a single call to the function DGEMM.    // allocate memory for matrix S double\* S = new double[10\*10];    // configure DGEMM parameters  char transa = 'N';  char transb = 'T';  // use the transpose of the right-hand matrix B  int m = 10; int k = nlines; int n = 10; double alpha = 1.0; double beta = 0.0; int lda = 10;  // number of rows of left-hand matrix A  int ldb = 10;  // number of rows of right-hand matrix B (before transposition) int ldc = 10;  // number of rows of result matrix C    DGEMM(&transa, &transb, &m, &n, &k, &alpha, D, &lda, D, &ldb, &beta, S, &ldc);    // discard matrix D delete [] D;    For the example file mag.txt, we have S[10x10] = D[10x25599] \* DT[25599x10].  The resulting matrix S appears as attachment below.    Now split the 10x10 *S* matrix in 4 smaller matrices as follows:    [https://sites.google.com/site/sailboatinstruments1/_/rsrc/1315691127225/step-1/ssplit.jpg?height=54&width=200](https://sites.google.com/site/sailboatinstruments1/step-1/ssplit.jpg?attredirects=0)    Because *S* is a symmetric matrix, *S12T* is the transpose of *S12*.    // create S11   6x6 double\* S11 = new double[6\*6]; for(i = 0; i < 6; i++) {     for(j = 0; j < 6; j++)         S11[6\*j+i] = S[10\*j+i]; }    // create S12  6x4 double\* S12 = new double[6\*4]; for(i = 0; i < 6; i++) {      for(j = 0; j < 4; j++)         S12[6\*j+i] = S[10\*j+i+60]; }   // create S12t  4x6 double\* S12t = new double[4\*6]; for(i = 0; i < 4; i++) {     for(j = 0; j < 6; j++)         S12t[4\*j+i] = S[10\*j+i+6]; }   // create S22 4x4 double\* S22 = new double[4\*4]; for(i = 0; i < 4; i++) {     for(j = 0; j < 4; j++)         S22[4\*j+i] = S[10\*j+i+66]; }    Calculate *S22-1*, the Moore-Penrose pseudo-inverse of the square matrix *S22.*  Note : the pseudo-inverse of a square matrix is identical to its inverse, except if the matrix is singular, which should not occur unless you have a really bad or incomplete sample of measurements.   // allocate memory for the pseudo-inverse S22p double\* S22\_1 = new double[4\*4];    // initially set S22\_1 to identity matrix for(i = 0; i < 4; i++) {     for(j = 0; j < 4; j++)     {         if(i == j)             S22\_1[i\*4+j] = 1.0;         else             S22\_1[i\*4+j] = 0.0;     } }    // configure the parameters for querying the optimal DGELSS workspace int info;  int ldwork = -1;  double \*work = new double[1];  int nrhs = 4; int lda = 4;  int ldb = 4; double\* singz = new double[4]; double rcond = -1.0; int irank = -1;    // query the optimal workspace DGELSS(&m, &n, &nrhs, S22, &lda, S22\_1, &ldb, singz, &rcond, &irank, work, &ldwork, &info);    // allocate optimal workspace ldwork = work[0];  delete [] work;  work = new double[ldwork];   // working call to DGELSS DGELSS(&m, &n, &nrhs, S22, &lda, S22\_1, &ldb, singz, &rcond, &irank, work, &ldwork, &info);    delete [] work;  delete [] singz;    Calculate *SS* = *S11 - S12\* S22-1\* S12T*    First calculate *S22a* = *S22-1* \* *S12T*      ( 4x6 = 4x4 \* 4x6)   (mxn = mxk \* kxn)    double\* S22a = new double[4\*6];  transb = 'N'; m = 4; n = 6; k = 4; lda = m;   ldb = k; ldc = m;   DGEMM(&transa, &transb, &m, &n, &k, &alpha, S22\_1, &lda,  S12t, &ldb, &beta, S22a, &ldc);      Then calculate *S22b* = *S12*\* *S22a*      ( 6x6 = 6x4 \* 4x6)   (mxn = mxk \* kxn)    double\* S22b = new double[6\*6];  m = 6; n = 6; k = 4; lda = m;  ldb = k; ldc = m;  DGEMM(&transa, &transb, &m, &n, &k, &alpha, S12, &lda, S22a, &ldb, &beta, S22b, &ldc);      Finallly calculate *SS*= *S11* - *S22b*    double\* SS = new double[6\*6];  for(i = 0; i < 36; i++)      SS[i] = S11[i]- S22b[i];    Setup the constraint matrix C    double\* C = new double[6\*6];   C[0] = -1.0; C[6] = 1.0; C[12] = 1.0; C[18] = 0.0;  C[24] = 0.0;  C[30] = 0.0; C[1] = 1.0;  C[7] = -1.0; C[13] = 1.0; C[19] = 0.0;  C[25] = 0.0;  C[31] = 0.0; C[2] = 1.0;  C[8] = 1.0; C[14] = -1.0; C[20] = 0.0;  C[26] = 0.0;  C[32] = 0.0; C[3] = 0.0;  C[9] = 0.0;  C[15] = 0.0;  C[21] = -4.0; C[27] = 0.0;  C[33] = 0.0; C[4] = 0.0;  C[10] = 0.0; C[16] = 0.0;  C[22] = 0.0;  C[28] = -4.0; C[34] = 0.0; C[5] = 0.0;  C[11] = 0.0; C[17] = 0.0;  C[23] = 0.0;  C[29] = 0.0;  C[35] = -4.0;    Invert matrix C in place    double \*ipiv = new double[6]; work = new double[1]; lwork = -1;  m = 6;  DGETRF(&m, &m, C, &m, ipiv, &info);    // query optimal DGRTRI workspace  DGETRI(&m, C, &m, ipiv, work, &lwork, &info);    lwork = work[0];  delete [] work;  work = new double[lwork];    // working DGETRI call  DGETRI(&m, C, &m, ipiv, work, &lwork, &info);  delete [] work;  delete [] ipiv;    Calculate *E* = *C* \* *SS*      ( 6x6 = 6x6 \* 6x6) (mxn = mxk \* kxn)    double\* E = new double[6\*6];  m = 6; n = 6; k = 6; lda = m;  ldb = k; ldc = m;  DGEMM(&transa, &transb, &m, &n, &k, &alpha, C, &lda,  SS, &ldb, &beta, E, &ldc);    Calculate eigenvalues wr(6x1) and eigenvectors vr(6x6) of matrix *E*    char jobvl = 'N'; char jobvr = 'V'; n = 6; lda = 6; double\* wr = new double[6]; double\* wi = new double[6]; double\* vl = NULL; int ldvl = 6 ; double\* vr = new double[6\*6]; int ldvr = 6; work = new double[1]; lwork = -1; DGEEV(&jobvl, &jobvr, &n, E, &lda, wr, wi, vl, &ldvl, vr, &ldvr, work, &lwork, &info);  lwork = work[0];  delete [] work;  work = new double[lwork]; DGEEV(&jobvl, &jobvr, &n, E, &lda, wr, wi, vl, &ldvl, vr, &ldvr, work, &lwork, &info);  delete [] wi;    Find the zero-based position of the only positive eigenvalue. The associated eigenvector will be in the corresponding column of matrix vr(6x6).    int index = 0; double maxval = wr[0]; for(i = 1; i < 6; i++) {     if(wr[i] > maxval)     {         maxval = wr[i];         index = i;     } }    Extract the associated eigenvector *v1*    v1 = new double[6];   v1[0] = vr[6\*index];   v1[1] = vr[6\*index+1]; v1[2] = vr[6\*index+2]; v1[3] = vr[6\*index+3]; v1[4] = vr[6\*index+4]; v1[5] = vr[6\*index+5];    delete [] wr;  delete [] vr;    Check sign of eigenvector *v1*    if(v1[0] < 0.0)  {   v1[0] = -v1[0];   v1[1] = -v1[1];   v1[2] = -v1[2];   v1[3] = -v1[3];   v1[4] = -v1[4];   v1[5] = -v1[5];  }    Calculate *v2* = *S22a* \* *v1*      ( 4x1 = 4x6 \* 6x1)   (mxn = mxk \* kxn)    v2 = new double[4];  m = 4;  n = 1; k = 6; lda = m;  ldb = k; ldc = m;  DGEMM(&transa, &transb, &m, &n, &k, &alpha, S22a, &lda,  v1, &ldb, &beta, v2, &ldc);    Setup vector *v*    v = new double[10];  v[0] = v1[0]; v[1] = v1[1]; v[2] = v1[2]; v[3] = v1[3]; v[4] = v1[4]; v[5] = v1[5]; v[6] = -v2[0]; v[7] = -v2[1]; v[8] = -v2[2]; v[9] = -v2[3];    delete [] v1;  delete [] v2;    At this point, we have found the general equation of the fitted ellipsoid:    [https://sites.google.com/site/sailboatinstruments1/_/rsrc/1320242405958/step-1/quadric1.jpg?height=82&width=400](https://sites.google.com/site/sailboatinstruments1/step-1/quadric1.jpg?attredirects=0)   Source : <http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/geometry/simple.html>    where:      A = v[0]        - term in x2      B = v[1]        - term in y2      C = v[2]        - term in z2      D = v[5]        - term in xy      E = v[4]        - term in xz      F = v[3]        - term in yz      G = v[6]       - term in x      H = v[7]        - term in y       I = v[8]        - term in z      J = v[9]        - constant term    Note the different order of terms in xy, xz and yz between this general equation and the design matrix D above, which explains why we find the order v[5], v[4], v[3] instead of v[3], v[4], v[5].    The general equation of the ellipsoid can also be put in matrix form:    [https://sites.google.com/site/sailboatinstruments1/_/rsrc/1320244905025/step-1/quadric2.png?height=151&width=400](https://sites.google.com/site/sailboatinstruments1/step-1/quadric2.png?attredirects=0)   Source : <http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/geometry/simple.html>    If we define:  [https://sites.google.com/site/sailboatinstruments1/_/rsrc/1320250199532/step-1/QU.jpg?height=80&width=320](https://sites.google.com/site/sailboatinstruments1/step-1/QU.jpg?attredirects=0)    then the center of the ellipsoid can be calculated as the vector    B = - Q-1 U  .  The center of the ellipsoid represents the combined bias.    Setup symmetric matrix *Q* (3x3)    Q = new double[3\*3];   Q[0] = v[0]; Q[1] = v[5]; Q[2] = v[4]; Q[3] = v[5]; Q[4] = v[1]; Q[5] = v[3]; Q[6] = v[4]; Q[7] = v[3]; Q[8] = v[2];    Setup vector *U*    U = new double[3];  U[0] = v[6]; U[1] = v[7]; U[2] = v[8];    Calculate matrix *Q-1*, the inverse of matrix *Q*    double\* Q\_1 = new double[3\*3];    for(i = 0; i < 9; i++)      Q\_1[i] = Q[i];    ipiv = new double[3]; work = new double[1]; lwork = -1;  m = 3; DGETRF(&m, &m, Q\_1, &m, ipiv, &info);  DGETRI(&m, Q\_1, &m, ipiv, work, &lwork, &info);  lwork = work[0];  delete [] work;  work = new double[lwork];  DGETRI(&m, Q\_1, &m, ipiv, work, &lwork, &info);  delete [] work;  delete [] ipiv;    Calculate *B* = *Q-1* \* *U*( 3x1 = 3x3 \* 3x1)   (mxn = mxk \* kxn)    double\* B = new double[3];  m = 3;  n = 1; k = 3; lda = m;  ldb = k; ldc = m;  DGEMM(&transa, &transb, &m, &n, &k, &alpha, Q\_1, &lda, U, &ldb, &beta, B, &ldc);    Calculate combined bias    B[0] = -B[0];     // x-axis combined bias B[1] = -B[1];     // y-axis combined bias B[2] = -B[2];     // z-axis combined bias        It can be shown that :   (see the page 'Geometric interpretation')    [https://sites.google.com/site/sailboatinstruments1/_/rsrc/1320368645647/step-1/itcanbe.jpg](https://sites.google.com/site/sailboatinstruments1/step-1/itcanbe.jpg?attredirects=0)  where HM is the norm of the field provided by the user.      Calculate *btqb* =  *BT \* Q* \* *B*    First calculate Q*B* = *Q* \* *B*( 3x1 = 3x3 \* 3x1) (mxn = mxk \* kxn)    double\* QB = new double[3];  m = 3;  n = 1; k = 3; lda = m;  ldb = k; ldc = m;  DGEMM(&transa, &transb, &m, &n, &k, &alpha, Q, &lda, B, &ldb, &beta, QB, &ldc);    Then calculate *btqb = BT \* QB*    ( 1x1 = 1x3 \* 3x1) (mxn = mxk \* kxn)    double btqb;  m = 1;  n = 1; k = 3; lda = m;  ldb = k; ldc = m;  DGEMM(&transa, &transb, &m, &n, &k, &alpha, B, &lda, QB, &ldb, &beta, &btqb, &ldc);    Calculate *hmb = sqrt(btqb - J).*    double J = v[9];  double hmb = sqrt(btqb - J);    Calculate *SQ*, the square root of matrix *Q*    jobvl = 'N'; jobvr = 'V'; n = 3; lda = 3; wr = new double[3]; wi = new double[3]; vl = NULL; ldvl = 3; vr = new double[9]; ldvr = 3; work = new double[1]; lwork = -1; DGEEV(&jobvl, &jobvr, &n, Q, &lda, wr, wi, vl, &ldvl, vr, &ldvr, work, &lwork, &info); lwork = work[0];  delete [] work;  work = new double[lwork]; DGEEV(&jobvl, &jobvr, &n, Q, &lda, wr, wi, vl, &ldvl, vr, &ldvr, work, &lwork, &info);    double\* Dz = new double[3\*3];  for(i = 0; i < 9; i++)      Dz[i] = 0.0;  Dz[0] = sqrt(wr[0]);  Dz[4] = sqrt(wr[1]);  Dz[8] = sqrt(wr[2]);    double\* vdz = new double[3\*3];  m = 3;  n = 3; k = 3; lda = m;  ldb = k; ldc = m; DGEMM(&transa, &transb, &m, &n, &k, &alpha, vr, &lda, Dz, &ldb, &beta, vdz, &ldc);          // invert matrix vr  ipiv = new double[3]; work = new double[1]; lwork = -1;  m = 3; DGETRF(&m, &m, vr, &m, ipiv, &info);  DGETRI(&m, vr, &m, ipiv, work, &lwork, &info);  lwork = work[0];  delete [] work;  work = new double[lwork];  DGETRI(&m, vr, &m, ipiv, work, &lwork, &info);  delete [] work;  delete [] ipiv;    double\* SQ = new double[3\*3];    m = 3;  n = 3; k = 3; lda = m;  ldb = k; ldc = m; DGEMM(&transa, &transb, &m, &n, &k, &alpha, vdz, &lda, vr, &ldb, &beta, SQ, &ldc);    Calculate *A-1*    double hm;  double\* A\_1 = new double[3\*3];    for(i = 0; i < 9; i++)      A\_1[i] = SQ[i] \* hm / hmb;    Calculate *A* to permit comparison with MagCal    double\* A = new double[3\*3];    for(i = 0; i < 9; i++)      A[i] = A\_1[i]    ipiv = new double[3]; work = new double[1]; lwork = -1;  m = 3; DGETRF(&m, &m, A, &m, ipiv, &info);  DGETRI(&m, A, &m, ipiv, work, &lwork, &info);  lwork = work[0];  delete [] work;  work = new double[lwork];  DGETRI(&m, A, &m, ipiv, work, &lwork, &info);  delete [] work;  delete [] ipiv; |