Natural helical motion of a transcription factor predicts a nonspecific complex and suggests a DNA binding mechanism

HK

June 24, 2014

Abstract

test section Abstract

Introduction

test section Introduction

Methods

I. Project coordinates of a helix on a plane of a unit sphere

From a set of Cartesian coordinates $\vec{R} = X\hat{\imath} + Y\hat{\jmath} + Z\hat{k}$ project onto the plane $aX + bY + cZ = 0$. In this section and in Section II, Z is the normal vector to the plane onto which we project our coordinates (i.e. $z = 0$), simplifying the projection algebra and, more importantly, simplifying parameter fitting.

Find the projection of (X, Y, Z) coordinates onto a unit plane, (a, b, c). Consider $\hat{e_x}$, where $\hat{n} = (a, b, c)$ and $\mid \hat{n} \mid = 1$, i.e. $a^2 + b^2 + c^2 = 1$,

$\hat{e}_X \cdot \hat{n} = a$ recovers the a dimension,

$\hat{e}_{X\parallel} = a\hat{e}_z$ along the vector normal to the plane,

and $\hat{e}_{X\perp} = \hat{e}_X - a\hat{e}_z$ within the plane,

where $\hat{e}_z = a\hat{e}_X + b\hat{e}_Y + c\hat{e}_Z$ is the normal vector.[1]

Rearranging leads to the normalized relation between our original X and our new x projected onto (a, b, c),

[1]Our unit normal vector, $\hat{e}_z = (\hat{e}_X, \hat{e}_Y, \hat{e}_Z) \cdot (a, b, c)$, has special properties in this article. $\hat{e}_z$ is the unit normal to the plane onto which we wish to project a helix to find a circle. If the projection gives a perfect circle, (a, b, c) is the plane where $\hat{e}_X = \cos(t) = \hat{e}_x$ and $\hat{e}_Y = \sin(t) = \hat{e}_y$ if the radius of the helix is 1 and t is the parameter of the helix; $t \in [0, 2\pi)$. $\hat{e}_z$, then, is not only the unit normal to the plane that our helix projects a perfect circle, but $\hat{e}_z$ is also the helical axis. Thus $\hat{e}_Z = kt$ and the pitch of the helix is simply $2\pi k$

$$\hat{e}_x = (\sqrt{1-a^2})\hat{e}_X - (\frac{ab}{\sqrt{1-a^2}})\hat{e}_Y - (\frac{ac}{\sqrt{1-a^2}})\hat{e}_Z \qquad 1$$

We find $\hat{e}_y$ by taking the cross product of $\hat{e}_x$ and $\hat{e}_z$, both of which we know. taking $\hat{e}_z$ as the normal vector to (a, b, c) and $\hat{e}_x$ from equation (1),

$$\hat{e}_y = \hat{e}_z \times \hat{e}_x, \qquad 2$$

$$\hat{e}_y = (\frac{1}{\sqrt{1-a^2}}) \begin{vmatrix} \hat{e}_X & \hat{e}_Y & \hat{e}_Z \\ a & b & c \\ 1-a^2 & -ab & -ac \end{vmatrix} \qquad 3$$

Finally, we combine the above three normalized relations for $\hat{e}_x$, $\hat{e}_y$, and $\hat{e}_z$ to project our original coordinates (X, Y, Z) onto the (a, b, c) plane to calculate new coordinates (x, y, z),

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sqrt{1-a^2} & 0 & a \\ \frac{-ab}{r} & \frac{c}{r} & b \\ \frac{-ac}{r} & \frac{-b}{r} & c \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \qquad 4$$

which we use to fit (X, Y, Z) to a circle on the plane (a, b, c): the $\hat{e}_x$ and $\hat{e}_y$ components of an ideal cylindrical helix whose helical axis is the normal vector to (a, b, c) is just a circle.

As we do not a priori know the helical axis, we wish to find the plane (a, b, c) whose normal vector *is* helical axis. So we need to be able to transform (a, b, c) to find the helical axis. We find the helical axis by solving the orthogonal problem: onto which plane (a, b, c) does (X, Y, Z) project the best estimated circle? The helical axis is then the normal vector to that plane.

We define the (a, b, c) plane over a unit sphere. This requires only two variables, θ and ϕ, to define (a, b, c). That is, we define (a, b, c) in spherical coordinates by using the relations to Cartesian coordinates, $x = \sin\theta\cos\phi$, $y = \sin\theta\sin\phi$, and $z = \cos\theta$ ($r = 1$ for a unit sphere). So, (a, b, c) is defined:

$$a = \sin\theta\cos\phi \qquad 5$$

$$b = \sin\theta\sin\phi \qquad 6$$

$$b = \cos\theta \qquad 7$$

II. MAIN PROGRAM kSuperhelix.F90

Algorithm informationa and function flow

1 Read control variables

Cartesian coordinates to fit. These are assumed to be indexed by frame number, atom number, and x, y, and z.

Resolution of rotations over the unit sphere ultimately controlling how many planes are searched within the ranges of ϕ and θ set below.

Number of atoms to fit, per frame if a trajectory is read.

Number of frames of a trajectory, set to one if just one structure.

Range of ϕ and θ over a unit sphere to rotate planes $f(\phi, \theta) = (a, b, c)$ onto which we will project our input coordinates in search of a optimal set of helical parameters.

2 Build three arrays for all X, all Y, and all Z input coordinates. Initialize the three arrays that will hold the rotated coordinates x, y, and z.

3 Begin the search loop over the unit sphere.

First, convert the input variables from above so that we can iterate rotations of the coordinates onto planes of the unit sphere per the ranges of ϕ and θ subject to the angle between rotations (fineness) set by the Resolution. These are the iteration variables, (ϕ_i, θ_i).

Second, take the iteration variables (ϕ_i, θ_i) to find the unit plane (a, b, c) via equations (6), (7), and (8). The rotation matrix in equation (5) rotates the input coordinates into new coordinates, $f(\phi_i, \theta_i) = (a_i, b_i, c_i)$=$(x_i, y_i, z_i)$.

Third, the new coordinates are then fit to a circle using SVD. The residual of the fit is stored in an array indexed by (ϕ_i, θ_i), the resulting radius in an array indexed by (ϕ_i, θ_i), the resulting circle center in an array indexed by (ϕ_i, θ_i), the sweep in an array indexed by (ϕ_i, θ_i), the pitch in an array indexed by (ϕ_i, θ_i).

Fourth, the loop ends after the full range of ϕ and θ have been searched.

4 The MINLOC intrinsic function is used to find the element index in the residuals matrix with the smallest value, corresponding to the rotation with the best fit and the best estimate of the normal plane to the helical axis.

5 With the index of the best helical axis known, we simply print the elements from the radius, pitch, etc. arrays with the index from the previous step in the algorithm.

III. Fitting helical parameters by fitting projected coordinates to a circle using SVD

If the helical axis of a perfect cylindrical helix is the normal to a plane (a, b, c), the projection of helical points onto the plane is just a circle. To find the plane that leads to a best fit circle for the projected coordinates, $(a_{fit}, b_{fit}, c_{fit})$, and noting the radius, R_{fit} and center, (x_0, y_0) after fitting, we assume, is equivalent to finding the radius and center of the original helix. The remaining unknown parameter helical pitch, κ, would then be the pitch of the coordinates along the normal to $(a_{fit}, b_{fit}, c_{fit})$.

To find the best helical parameters to our coordinates, we must define the residual to our fitting,

$$Res^2 = \sum_{i=1}^{N_R} (x_i - x_0)^2 + (y_i - y_0)^2 + R_{fit}^2 - 2R_{fit}\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} \quad 8$$

for the linear least-squares problem ($\mathbf{AX} = \mathbf{B}$) fitting x_i and y_i for our NR coordinates by calculating the area of a circle,

$$\begin{pmatrix} 1 & 2x_{i=1} & 2y_{i=1} \\ 1 & ... & ... \\ 1 & 2x_{i=N_R} & 2y_{i=N_R} \end{pmatrix} \begin{pmatrix} k \\ x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} x_{i=1}^2 + y_{i=1}^2 \\ ... \\ x_{i=N_R}^2 + y_{i=N_R}^2 \end{pmatrix} \quad 9$$

This linear least-squares problem is then solved using Lawson's Singular Value Decomposition (SVD) routine to return the best values for k_i, $x_{0,i}$, and $y_{0,i}$ in the plane (a_i, b_i, c_i) for (ϕ_i, θ_i). The SVD routine was modified to prune output (see below) so that only the fitted parameters were returned, which replace the contents of the input newXYZAMAT(3,3) array. Within that routine, we used the singular value analysis (SVA) subroutine specific for linear least squares problems. The parameters and the residuals are then indexed by ϕ_i and θ_i into separate arrays. In the array of residuals, the smallest element is located, the index obtained and then used to locate the associated parameters $k_{bestfit}$, $x_{0,bestfit}$, and $y_{0,bestfit}$ from their arrays.

We compute the singular value decomposition of $\mathbf{A}$, and M by N matrix as

$$\mathbf{A} = \mathbf{USV}' \quad 10$$

where $\mathbf{U}$ is M by M orthogonal, $\mathbf{S}$ is M by N diagonal with the diagonal terms nonnegative and ordered from large to small, and $\mathbf{V}$ is N by N orthogonal. The following is also true,

$$\mathbf{S} = \mathbf{U}'\mathbf{AV} \quad 11$$

The matrix $\mathbf{V}$ is returned from the subroutine SVDRS (Right Side vector) to the MAIN PROGRAM via the $\mathbf{A}$ array - the rotated coordinates whose e_z components have been set to one and the e_x and e_y components have been doubled per the linear problem in equation (10).

The diagonal terms of the $\mathbf{S}$ matrix return the singular values to the MAIN PROGRAM.

The product matrix $\mathbf{G} = \mathbf{U}'\mathbf{B}$ is returned from the subroutine SVDRS to the MAIN PROGRAM via the $\mathbf{B}$ matrix.

To obtain the best estimate of our linear problem set forth in equation (10), the solution **X** is obtained in the MAIN PROGRAM by computing the sum for $i = 1, ..., R$, where R is the psuedorank of A based on the sizes of the singular values, 3, of the outer products,
(column i of V) * (1/S(i)) * (row i of G)

Modifications to the SVD routine

The following output from the subroutine are muted in the output, as we require only the above three sets of values:

- Warning messages from QRDB calls (when ipass == 2)

  "SVRDS - Warning of Subaccuracy! Full accuracy was not attained in the SVD of the bidiagonal matrix." (lines 3060 to 3065)

  Error flag IPASS; when ipass = 2, a convergence failure in the QRDB algorithm that uses the QR algorithm for the SVD bidiagonal matrix, **D** with diagonal elements $\mathbf{Q(1:N)}$ and superdiagonal elements $\mathbf{E(2:N)}$ is pre- and post-multiplied by elementary rotation matrices **R** and **P** so that $R_k * ...R2 * R1 * D * P1' * P2' * ... * Pk' = diag(S1, ..., S_n)$

- Print only the R-norms (lines 2616 to 2623)
- Formated output (lines 1276, 1278, 1285, 1288, 1293, 1295)
- Sequence of candidate solutions, $mathbfX$ (line 1249)
- Comparison of y-norm, r-norm and their logarithms (lines 2589, 2591)
- Levenberg-Marquardt parameters for xnorm and rnorm are not printed

IV. The F90 code

First we load a set of Cartesian coordinates with NR atoms (line 70 to 72),

```
CALL getarg(1,INFILE)
OPEN(51,FILE=INFILE,ACTION='READ',STATUS='OLD')
CLOSE(1)
```

into an array of size $3 \times$ NR (line 107 to 109),

```
ALLOCATE(x          (3*NR*NFRAMES),stat=ierr)
IF(IERR /= 0) WRITE(*,*) "Allocate error in x, dude..."
READ(51,*) x
```

where NR is the number of atoms in the system read in from prompt and NFRAMES is the number of separate snapshots of system coordinates (line 84 and 85).

```
CALL getarg(3,ARG_NATOMS)
READ(ARG_NATOMS,*) NR
```

For values θ_i and ϕ_i, the range is set from prompt by variables PHISTART, PHIEND, THETASTART, and THETAEND. PHISTART > 0, PHIEND $< \pi$, THETASTART > 0, and THETAEND $< 2\pi$; PHISTART $< \phi_i <$ PHIEND and THETASTART $< \theta_i <$ THETAEND (line 94 to 104).

```
CALL getarg(5,ARG_PHISTART)
READ(ARG_PHISTART,*) PHISTART

CALL getarg(6,ARG_PHIEND)
READ(ARG_PHIEND,*) PHIEND

CALL getarg(7,ARG_THESTART)
READ(ARG_THESTART,*) THESTART

CALL getarg(8,ARG_THEEND)
READ(ARG_THEEND,*) THEEND
```

We use ϕ_i and θ_i to drive the entire program, as we explore ϕ_i and θ_i over a unit sphere. The final variable set by prompt (line 78 and 79), MAXITR, controls the granularity of the unit sphere over which we project our original coordiantes (X, Y, Z) onto (a, b, c) by dividing the range of ϕ and θ by MAXITR, which also controls the number of DO LOOP interations (line 158 to 181),

```
   CALL getarg(2,ARG_MAXITR)
   READ(ARG_MAXITR,*) MAXITR
```

```
   PHIS       = PHISTART * PI / 180.0
   THES       = THESTART * PI / 180.0

   PHIN       = PHIEND   * PI / 180.0
   THEN       = THEEND   * PI / 180.0

   PHIRANGE = PHIN - PHIS
   THERANGE = THEN - THES

! Spherical coordinates: 0 < phi < PI ; 0 < theta < 2*PI
   dTHETA        = THERANGE / MAXITR
   dPHI          = PHIRANGE / MAXITR

 DO IPHI     = 2,MAXITR-1      ! Loop-PHI
  DO ITHETA = 2,MAXITR-1      ! Loop-THETA

! Get the Phi and Theta angles
      PHI_L    = ( IPHI   * dPHI )   + PHIS
      THETA_M = ( ITHETA * dTHETA ) + THES

! Convert PHI_L and THETA_M to degrees for writing:
      PHID = PHI_L   * 180.0 /PI
      THED = THETA_M * 180.0 /PI
```

To project our input coordinates we define (a, b, c) located at (PHIL,THETAM), i.e. (ϕ_i, θ_i) on the unit sphere by using equation 5. First we define (O1,O2,O3,R), i.e. (a,b,c,r) from equations 6 through 9 (line 183 to 187),

```
O1   = SIN(THETA_M)  * COS(PHI_L)
O2   = SIN(PHI_L)    * SIN(THETA_M)
O3   = COS(THETA_M)
R    = SQRT(1-O1**2)
```

Second we build the elements of the 3 × 3 matrix,

```
COEF1 =    SQRT(1.0-O1**2.0)
COEF2 =    0
COEF3 =    O1
COEF4 =   -O1*O2  /  R
COEF5 =    O3     /  R
COEF6 =    O2
COEF7 =   -O1*O3  /  R
COEF8 =   -O2     /  R
COEF9 =    O3
```

And third we multiply the above 3 × 3 matrix of equation (5) and the input coordinates to build the X (newX), Y (newY), and Z (newZ) components of the projected coordinates,

```
DO IATOM = 1,NR                ! Loop-Atoms
   newX(IATOM) = COEF1*oldX(IATOM) + COEF2*oldY(IATOM)&
   + COEF3*oldZ(IATOM)
   newY(IATOM) = COEF4*oldX(IATOM) + COEF5*oldY(IATOM)&
   + COEF6*oldZ(IATOM)
   newZ(IATOM) = COEF7*oldX(IATOM) + COEF8*oldY(IATOM)&
   + COEF9*oldZ(IATOM)
END DO
```

0.1 call SVA subroutine from Lawson's SVD routine

SVD is used to solve the linear least-squares problem, $\mathbf{AX} = \mathbf{B}$ posed in equation 12 by finding the best approximation to a solution $\mathbf{X}$, i.e. minimizing the L2 norm of the residual $\mathbf{R} = \mathbf{AX} - \mathbf{B}$ (line 219).

```
CALL SVA (newXYZ_AMAT,NR,NR,3 ,NR,newXYZ_BMAT,SING,&
KPVEC,NAMES,1 ,D,WORK)
```

The calling syntax is

```
SUBROUTINE SVA( A, MDA, M, N, MDATA, B, SING, KPVEC, &
NAMES, ISCALE, D, WORK)
```

The parameters are

- A = **newXYZAMAT**, i.e. the **A**, matrix.

  KIND=8, REAL

  Dimension(MDA,N)

  OUTPUT: **A**

  ```
  DOUBLE PRECISION,DIMENSION(:,:) ,ALLOCATABLE :: &
  newXYZ_AMAT
  ```

- MDA = NR, first dimensioning parameter of **A**

  KIND = 4, INTEGER

  ```
  INTEGER(KIND=4)                        :: NR
  ```

- N = 3, columns: X, Y, Z coordinates in **A**

  KIND = 4, INTEGER

- M = NR, rows: number of atoms in **A**

  KIND = 4, INTEGER

- MDATA = NR, rows for statistics reporting on least squares problem

  KIND = 4, INTEGER

- B = **newXYZBMAT**, i.e. the **B**, matrix.

  KIND = 8, REAL

  INPUT: Right-side vector of least squares problem

  OUTPUT: vector $\mathbf{G} = \mathbf{U}' * \mathbf{B}$, where **U** comes from SVD of **A**

  ```
  DOUBLE PRECISION, DIMENSION(:) , ALLOCATABLE :: &
  newXYZ_BMAT
  ```

- SING(N) = Singular values of **A**, in descending order, in locations (1) through min(M,N).

  KIND = 8, REAL

```
DOUBLE PRECISION, DIMENSION(:) , ALLOCATABLE :: &
SING
```

- KPVEC(4) = Print controller

  KPVEC(1) = 1, contents of KPVEC modified; default=0

  KPVEC(2) = 000000, Print block set to mute formatted output: (1) header; (2) v-matrix; (3) singular values and related quantities; (4) ynorm and rnorm and their logarithms; (5) Levenberg-Marquart analysis; (6) candidate solutions

  KPVEC(3) = -1, Unit: output written to "*" output unit, i.e. standard output unit

  KPVEC(4)=69, print width, unused though

```
INTEGER,SAVE,DIMENSION(4)          :: &
KPVEC = (/ 1, 000000, −1,69 /)
```

- NAMES = Controls name of jth component of solution vector. Empty string: If NAMES(1) contains only blank characters, the subroutine will not access array beyond first element.

```
CHARACTER(LEN=8), DIMENSION(1)    :: NAMES = (/'␣'/)
```

- ISCALE = 1. Column spacing option: 1, use identity scaling and ignore **D** array. 2, scale nonzero columns fo have unit Euclidean length; store reciprocal lengths of original nonzero columns in **D**. 3, user supplies column scaling factors in **D**. This routine mutliplies column **J** by **D**(**J**) and remores the scaling from the solution at the end.

  KIND = 4, INTEGER

- D = **D**(**NR**), a vector whose use depends on the value of ISCALE

```
DOUBLE PRECISION,DIMENSION(MX)    :: D(MX)
INTEGER,PARAMETER                 :: MX = 3
```

- WORK = **WORK**($\mathbf{2} * \mathbf{NR}$)Workspace array for routine

  KIND = 8, REAL

0.2 ϕ and θ arrays

all ϕ_i and θ_i scanned in the algorithm are stored into variable dimension arrays (line 129 and 130),

```
ALLOCATE(PHIDMAT(MAXITR,MAXITR))
ALLOCATE(THEDMAT(MAXITR,MAXITR))
```

(line 319 and 320),

```
THEDMAT(IPHI:IPHI,ITHETA:ITHETA) = THED
PHIDMAT(IPHI:IPHI,ITHETA:ITHETA) = PHID
```

0.3 circle center

SVD returns the values of x_0 and y_0 per equation 12 that are twice the values of the actual center in that plane (line 223 to 226),

```
    X0   = newXYZAMAT(2,3) /2
    Y0   = newXYZAMAT(3,3) /2
    X02  = X0*X0
    Y02  = Y0*Y0
```

0.4 circle radius

The circle center and parameter k (see equation 12) is then used to calculate the radius of the circle in that plane (line 227),

```
  R_SVD = SQRT( X02 + Y02 + (newXYZ_AMAT(1,3)) )
```

0.5 circle radii array

And the array of radii is built (line 315),

```
  RSVDMAT(IPHI:IPHI,ITHETA:ITHETA) = R_SVD
```

0.6 helical pitch

The pitch of the helix is calculated without affecting our SVD fitting, it is simply calculated for each $/phi_i$ and θ_i and stored into an array for later. We calculate the pitch of the helix along the normal to each and every plane (see subsection **locate lowest residual**).

After the original coordinates have been projected onto the plane, the incremental pitch between successive points along the helix are calculated by finding the angle between three points: $atom_i$, (x_0, y_0), and $atom_{i+1}$, i.e. (x_i, y_i), (x_0, y_0), (x_{i+1}, y_{i+1}). Summed over all atoms, this defines the angular sweep of the helix, κ. The displacement between the first and last atom along the z-component multiplied by 2π is the pitch (line 235 to 264),

```
THETAHELIX = 0.0D+00
DO IATOM = 1, NR-1
    VECIX     = newX(IATOM)    - X0
    VECIY     = newY(IATOM)    - Y0
    VECIp1X   = newX(IATOM+1)  - X0
    VECIp1Y   = newY(IATOM+1)  - Y0
     DOTIIp1  = VECIX*VECIp1X + VECIY*VECIp1Y
     MAGI     = SQRT(VECIX**2 + VECIY**2)
     MAGIp1   = SQRT(VECIp1X**2 + VECIp1Y**2)

     ! ALPHA = ARCCOS(A*B/|A||B|), A is ith vec, B is i+1th vec...
     DOTOVERMAGS = DOTIIp1/(MAGI*MAGIp1)
     THETASTEP   = ACOS(DOTOVERMAGS)
     THEDSTEP    = THETASTEP*RTD

     THETAHELIX  = THETAHELIX + THEDSTEP

END DO
     ZETAPITCH = (newZ(NR)) - (newZ(1))    !height, z-component, of helix
     KAPPA     = ZETAPITCH * RTD / THETAHELIX    !this needs to be in rad.
     PITCH     = ABS ( KAPPA * 2*PI )          ! abs of pitch is ok, right?
```

0.7 pitch array

The pitch is then stored into an array (line 305),

```
PITMAT(IPHI:IPHI,ITHETA:ITHETA)  =  PITCH
```

The helical sweep is then stored into an array (line 308),

```
THETAPITMAT(IPHI:IPHI,ITHETA:ITHETA)=  THETAHELIX
```

0.8 residuals

The residual is calculated (line 268 to 278),

```
!Calculate the residual as per Vageli >>>
 RES = 0.0D+00
 DO I = 1,NR
   RES = RES + ABS( &
                      & (newX(I)-X0)*(newX(I)-X0) + &
                      & (newY(I)-Y0)*(newY(I)-Y0) + &
                      & (R_SVD)*(R_SVD)           - &
                      & (2*R_SVD)*( SQRT (          &
                      & (newX(I)-X0)*(newX(I)-X0) + &
                      & (newY(I)-Y0)*(newY(I)-Y0)  )  )  )
 END DO
```

0.9 residuals array

And the array of residuals is built (line 300),

```
 RESMAT(IPHI:IPHI,ITHETA:ITHETA)   = RES
```

0.10 end loop over unit sphere

```
  END DO ! ITHETA_complete the loop of the phi-s    !
END DO ! IPHI_complete the loop of the theta-s     !
```

0.11 locate lowest residual

Once all the planes on the unit sphere surface have been fitted and the residuals stored into RESMAT, the FORTRAN intrinsic function MINLOC is used to find the element in the array with the smallest value.

This is the critical point in the program, where we assume the plane yielding the lowest residual from fitting is the normal plane for the helix (line 335 to 338),

```
LOWESTRES     = MINLOC ( RESMAT , MASK = RESMAT .GT. 0.1D-11 )

MIN_IPHI      = LOWESTRES(1)
MIN_ITHETA    = LOWESTRES(2)
```

And with the index for RESMAT used to find the associated radius for the plane $(a_{fit}, b_{fit}, c_{fit})$ that gave the best fit, i.e. the plane with the best residual fit to a circle for the helix's projected coordinates $(\phi_{fit}, \theta_{fit}, Res_{min})$. That is MINIPHI and MINITHETA from above code (line 337 and 338) are $(\phi_{fit}, \theta_{fit})$.

We now know the best plane and can find the best residual (line 300 and 344),

```
RESMAT(IPHI:IPHI,ITHETA:ITHETA)   = RES
BEST_RES = RESMAT (MIN_IPHI,MIN_ITHETA)
```

the best radius (line 345),

```
RSVDMAT(IPHI:IPHI,ITHETA:ITHETA) = R_SVD
BEST_RAD = RSVDMAT(MIN_IPHI,MIN_ITHETA)
```

the best ϕ, i.e. ϕ_{fit} (line 348),

```
BEST_PHI = PHIDMAT(MIN_IPHI,MIN_ITHETA)
```

the best θ, i.e. θ_{fit} (line 349),

```
BEST_THE = THEDMAT(MIN_IPHI,MIN_ITHETA)
```

V. kSuperhelix.F90 sample

!!HK!! Vageli, i'll get you the test cases with pretty pictures soon. !!HK!!