Vademecum: list of functions of the software StrainModeler developed in the paper:

StrainModeler: A MATHEMATICA™ –based program for 3D analysis of finite and progressive strain

Nilo C. Bobillo-Ares[a], Jesús Aller[b], Fernando Bastida[b], Omar Menéndez[a], Richard J. Lisle[c]

[a] *Departamento de Matemáticas, Universidad de Oviedo, 33007 Oviedo, Spain*

[b] *Departamento de Geología, Universidad de Oviedo, Jesús Arias de Velasco s/n, 33005 Oviedo, Spain*

[c] *School of Earth and Ocean Sciences, Cardiff University, Cardiff CF10 3AT, UK*

Basis Change (BC__)
BLOCK DESCRIPTION
This is a straightforward method to make a basis change.
The new basis is defined from the old one using 2 + 2 angles
{alpha_1, theta_1} {alpha_2, theta_2}
(see figure)

BLOCK FUNCTIONS

Code: BC01
Description: It computes the matrix of the basis change.
Syntax: **generalBasisChangeMatrix**[*axisIpAngles*,*directionUpAngles*]
Inputs:
axisIpAngles: they are the angles corresponding to the direction of the axis i' of the new basis.
directionUpAngles: they are the angles of a direction u' that is also contained in the plane i-k.
Output: 3x3 base change matrix.

Code: BC02
Description: It computes the matrix of the deformation gradient in the old basis.
Syntax: **computeGradientOldBasis**[*changeMatrix*, *gradientNewBasis*]
Inputs:
changeMatrix: matrix of basis change.
gradientNewBasis: matrix of the deformation gradient in the new basis
Output: 3x3 matrix of the deformation gradient in the old basis

Code: BC03
Description: It computes the matrix of a vector in the old basis.
Syntax: **computeVectorOldBasis**[*changeMatrix*, *vectorNewBasis*]
Inputs:
changeMatrix: matrix of basis change.
vectorNewBasis: column matrix of a vector in the new basis.

Output: column matrix of a vector in the old basis.

Code: BC04
Description: It computes the matrix of the deformation gradient in the new basis.
Syntax: **computeGradientNewBasis**[*changeMatrix, gradientOldBasis*]
Inputs:
changeMatrix: matrix of basis change.
gradientOldBasis: matrix of the deformation gradient in the old basis.
Output: 3x3 matrix of the deformation gradient in the new basis.

Code: BC05
Description: It computes the matrix of a vector in the new basis.
Syntax: **computeVectorNewBasis**[*changeMatrix, vectorOldBasis*]
Inputs:
changeMatrix: matrix of basis change.
vectorOldBasis: column matrix of a vector in the old basis.
Output: column matrix of a vector in the new basis.

Matrix n - roots (MR)

BLOCK DESCRIPTION
Given the accumulated shear/flattening/exponential matrix, it computes its n-root.

BLOCK FUNCTIONS

Code: MR01
Description: It computes the n-root of a 3x3 simple shear matrix.
Syntax: **shearMatrixRoot**[*Ac, n*]
Inputs:
Ac*:* Matrix of the total simple shear.
n: Number of steps in which the process of simple shear is divided (order of the root).
Output: Matrix of the incremental deformation gradient (the n-root of the input matrix)

Code: MR02
Description: It computes the n-root of a 3x3 flattening matrix.
Syntax: **flatteningMatrixRoot**[*Ac,n*]
Inputs:
Ac*:* Matrix of the total flattening.
n: Number of steps in which the process of flattening is divided (order of the root).
Número de pasos en que quiero descomponer el proceso de flattening (orden de la raíz).
Output: Matrix of the incremental deformation gradient (the n-root of the input matrix)

Code: MR03
Description: It computes the n-root of a 3x3 exponential matrix.
Syntax: **exponentialMatrixRoot**[*Ac,n*]
Inputs:
Ac*:* Matrix of the total exponential process.
n: Number of steps (order of the root)
Output: Incremental gradient (the n-root of the input matrix)

Gradient Sequence Computations (GSC)

BLOCK DESCRIPTION
Computations associated to an Incremental Gradient Sequence (IGS).

BLOCK FUNCTIONS

Code: GSC01
Description: Builds an IGS composed by n identical matrices M:
M, n -> {M, M, M, ..., M}
Syntax: **buildIGS**[*M, n*]
Inputs:
M*:* Repeated matrix.
n: Number of steps.
Output: An IGS list of matrices, {M, M, M, ..., M}.

Code: GSC02
Description: It computes an Accumulated Gradient Sequence (AGS) from an IGS:
F= {M, M, M, ..., M} -> {M^1, M^2, ..., M^n}
Note: The elements of F may be different.
Syntax: **computeAGS**[*F*]
Inputs:
F: The IGS list of matrices.
Output: The AGS list {M^1, M^2, ..., M^n}.

Code: GSC03
Description: It computes a sequence of items composed of the axis ratios of the incremental strain ellipsoid {R_{13}, R_{12}, R_{23}} from an incremental gradient sequence (IGS).
Syntax: **computeSeqR13R12R23**[*F*]
Inputs:
F: The IGS list of matrices (the elements of F may be different).
Output: The list { {R_{13}, R_{12}, R_{23}}_1, {R_{13}, R_{12}, R_{23}}_2,..., {R_{13}, R_{12}, R_{23}}_n}

Code: GSC04
Description: It computes a sequence of items composed of the axis ratios of the incremental strain ellipsoid {R_{13}, R_{12}, R_{23}, A} from an incremental gradient sequence (IGS).
A: The product eigenvalue1 $\times$ eigenvalue3.

Syntax: **computeSeqR13R12R23A**[*F*]
Inputs:
F: IGS list of matrices (the elements of F may be different).
Output: The list { {R_{13}, R_{12}, R_{23},A}_1, {R_{13}, R_{12}, R_{23},A}_2,..., {R_{13}, R_{12}, R_{23},A}_n}

Code: GSC05
Description: It obtains the sequence of the three principal directions of the strain corresponding to the accumulated gradients.
Syntax: **computeSeqPrincVectors**[*F*]
Inputs:
F: IGS list of matrices (the elements of F may be different).
Output: The list {{V_1,V_2,V_3}_1, {V_1,V_2,V_3}_2, ..., {V_1,V_2,V_3}_n }.

Progressive Transformations (PT)

BLOCK DESCRIPTION
A progressive transformation is defined by an IGS F. We provide functions to transform directions and planes.

BLOCK FUNCTIONS

Code: PT01
Description: Starting from the initial vector v_0, we compute the successive transformed vectors;
F, v_0 -> {v_0, v_1, ..., v_n}
F: IGS
v_0: Initial vector
Caveat: The resulting sequence has n+1 elements.
Syntax: **progressiveVectorTransform**[*F, v_0*]
Inputs:
F: IGS F={F_1, F_2, ..., F_n}
v_0: Initial direction vector
Output: The list {v_0, v_1, ..., v_n}, where v_i=F_i*v(i-1).

Code: PT02
Description: It calculates two vectors {V_1,V_2} that define the plane from the dip and the dip direction, see fig. 4.
V_1 is the trace of the plane with the horizontal plane; V_2 is the vector corresponding to the dip.
Syntax: **calculatePlaneBasis**[*alpha, theta*]
Inputs:
alpha, theta are the angles that characterize the dip direction.
Output: The list {V_1,V_2}.

Code: PT03
Description: The function transforms the plane by a deformation defined by its gradient.

Syntax: **transformsPlaneParameters**[*basis, grad*]
Inputs:
basis: Vectors that define the original plane
grad: Gradient of the deformation.
Output: The list
{singular,norm1,norm2,normsRatio,dp1,dp2,normalPlano,normalPlanoAxial, norm1*norm2}

a) "singular" is an indicator if the ellipse is actually a circle.
   singular=False means that it is not a circle.
   If singular=True, the remaining entries are meaningless.
b) "norm1": magnitude of the major semi-axis of the strain ellipse on the deformed plane.
c) "norm2": magnitude of the minor semi-axis of the strain ellipse on the deformed plane.
d) "normsRatio" : semi-axes ratio of the strain ellipse on the deformed plane.
e) "dp1": orientation (α and θ values) of the major axis of the strain ellipse on the deformed plane.
f) "dp2" :orientation (α and θ values) of the minor axis of the strain ellipse on the deformed plane.
g) "PlaneNormal" : α and θ values for the transformed plane.
h) "AxialPlaneNormal": orientations (α and θ values) of the axial planes of the folds formed on the deformed plane. It is assumed that these axial planes contain the major axis of the finite strain ellipse on the plane and are perpendicular to the deformed plane.
i) "norm1$\times$ norm2": area change on the deformed plane.

Code: PT04
Description: Starting from the initial plane defined by basis0, it computes the data of the successive transformed planes. The successive transformations are defined by the IGS F.
Syntax: **progressivePlaneTransform**[*F, basis0*]
Inputs:
F: IGS F={F_1, F_2, ..., F_n}
basis0: Initial plane defined by two vectors (see the function PT02).
Output: The list { D_1, D_2, ..., D_n}, where D_i are the data calculated by PT03.

Code: PT05
Description: Given a sequence of vectors V = {v_0, v_1, ..., v_n}, the function gives a sequence constituted by the corresponding lengths of the vectors of V.
Syntax: **lengthVectorSeq**[*V*]
Inputs:
*V: a s*equence of vectors, {v_0, v_1, ..., v_n}.
Output: The list {L_0, L_1, ..., L_n}, where L_i = Norm[L_i].

Interface: Angles/Cartesian Coordinates (ACC)

BLOCK DESCRIPTION
Functions that compute the angular parameters (α, θ) of a vector from its Cartesian components and viceversa (see Fig. 2).

BLOCK FUNCTIONS

Code: ACC01
Description: Given a direction, defined by angles (α, θ), it computes the corresponding vector in Cartesian components.
Syntax: **fromAnglesToCartesian**[*alpha, theta*]
Inputs:
alpha, *theta*: The two angles that define a direction.
Output: The list {v_1, v_2, v_3}: Cartesian components of the unit vector associated to the direction.

Code: ACC02
Description: Given a direction, defined by a vector in Cartesian components, it computes the corresponding angles (vertical vectors have not well defined angles). This function is the inverse of ACC02.
Syntax: **evalDirAngles**[*v*]
Inputs:
v: The list {v_1, v_2, v_3}: Cartesian components of the unit vector associated to the direction.
Output : The list {*alpha*, *theta*}: the two angles that define a direction.

Code: ACC03
Description: A sequence of vectors in Cartesian coordinates is transformed in a sequence of directions in angle form (using ACC02).
Syntax: **fromVectorSeqToAngleSeq**[*seqv*]
Inputs:
seqv: The list {v_1, v_2, ..., v_n} of vectors in Cartesian components.
Output: The list {d_1, d_2, ..., d_n} of the corresponding directions in angle form, where d_i= {$alpha_i$, $theta_i$} are the two angles that define the direction di.

Code: ACC04
Description: Given a sequence of vectors normal to planes, it computes another sequence corresponding to the same planes defined by the two angles that define their dip direction.
Syntax: **fromNormalVectorSeqToAngleSeq**[*seqv*]
Inputs:
seqv: The list {v_1, v_2, ..., v_n} of normal vectors in Cartesian components.
Output: The list {dip_1, dip_2, ..., dip_n} of the corresponding directions in angle form, where dip_i= {$alpha_i$, $theta_i$} are the two angles that define the dip direction dip_i.

Code: ACC05
Description: Given a plane defined by a normal vector, it computes the angles of

the corresponding dip direction.
Syntax: **evalPlaneAngles**[*r*]
Inputs:
r: The list {r_1, r_2, r_3} of the normal vector in Cartesian components.
Output: The list {*alpha*, *theta*} with the dip direction angles.

Code: ACC06
Description: A sequence of angles in radians is transformed to degrees.
Syntax: **radianSeqToDegreeSeq**[*rs*]
Inputs:
rs: A list of angles in radians.
Output: The input list converted to degrees.

Plotting (P)
BLOCK DESCRIPTION
Graphical utilities.

BLOCK FUNCTIONS

Code: P01
Description: A sequence *seq* is paired with evenly spaced real numbers of a specified interval. The resulting sequence is adequate for the *Mathematica* ListPlot command.
Example:
{a,b,c}, {3,5} -> {{3,a}, {4,b}, {5,c}}
Syntax: **putOneSequence**[*seq, interval*]
Inputs:
seq: Sequence.
interval: {start, end} interval of real numbers.
Output : A sequence of pairs in which *seq* is paired with evenly spaced real numbers of the specified interval.

Code: P02
Description: This function generates a linear sequence of values.
Example:
the inputs
{3,5} ,5
generates the list of values
{3, 3.5, 4, 4.5, 5}
Example:
{a,b,c}, {3,5} -> {{3,a}, {4,b}, {5,c}}
Syntax: **generateLinParameterSeq** *[interval, n*]
Inputs:
interval: {start, end} interval of real numbers.
n: number of points.
Output : A sequence with evenly spaced real numbers of the specified interval.

Code: P03
Description: In order to plot seq2 vs seq1, this function constructs a seq of pairs; for example,
{2,7,8}, {6,3,1} -> {{2,6}, {7,3}, {8,1}}
If one of the sequences has an initial extra term, it is deleted:
{0,2,7,4}, {6,3,1} -> {{2,6}, {7,3}, {4,1}}.
The resulting sequence is adequate for the *Mathematica* ListPlot command.
Syntax: **putTwoSequences**[sx, sy]
Inputs:
sx: Sequence.
sy: Sequence.
Output : A sequence of pairs.

Code: P04
Description: From the three lists
paramSeq, alphaSeqRad, thetaSeqRad
It generates a new list with the values of the parameter and the coordinates (x,y) in the net for each point corresponding to the specified angles.
Syntax: **genParametrizedCurveSeq**[*paramSeq,alphaSeqRad,thetaSeqRad*]
Inputs:
paramSeq: The list {p_1, p_2, ..., p_n}
alphaSeqRad: The list {alpha_1, alpha_2, ..., alpha_n}
thetaSeqRad: The list {theta_1, theta_2, ..., theta_n}
Output: The list {t_1, t_2, ..., t_n}, where t_i={p_i,x_i,y_i}.

Code: P05
Description: A list is sectioned into a sequence of lists overlapping one element.
Syntax: **listSectioning**[*longList, p*]
Inputs:
longList: a list
p: The length of the sublists.
Output: The list of the sublists.

Code: P06
Description: In order to plot a curve with 'tics' over the EPN(*), a list with graphic elements of subarcs is generated.
Syntax: **putCurve**[*parCurveSeq, p*]
Inputs:
parCurveSeq: A parameterized curve (generated by P04)
p: The length of the subarcs.
Output: A list with graphic elements of subarcs.
The successive values of the parameter at the 'tics' are printed.
(*) EPN is for Equiareal Projection Net.

Code: P07
Description: It generates a list with graphic elements of subarcs, including parameter values at the 'tics'.

Syntax: **putCurveWithLabels**[*parCurveSeq, p*]
Inputs:
parCurveSeq: A parameterized curve (generated by P04)
p: The length of the subarcs.
Output: A list with graphic elements of subarcs, including parameter values at the 'tics'.
The successive values of the parameter at the 'tics' are printed.

Code: P08
Description: It generates a graphic element for a labeled point.
Syntax: **myPointWithLabel**[*t, x, y*]
Inputs:
t : Numeric label.
x: Point abscissa.
y : Point ordinate.
Output: Graphic element for the point including label.

Code: P09
Description: It generates a graphic element for a point.
Syntax: **myPoint**[*x, y*]
Inputs:
x: Point abscissa.
y: Point ordinate.
Output: Graphic element for the point.

Code: P10
Description: It generates a list with graphic primitives for plotting an arc without 'tics'.
Syntax: **putSimpleArc**[*parCurveSeq*]
Inputs:
parCurveSeq: A parameterized curve (generated by P04), a list {{t,x,y}, ...}.
Output: A list with graphic elements of the arc (without 'tics').

Code: P11
Description: It generates a graphic primitive list to draw a EPN(*).
Syntax: **generatesTemplate**[*deltaTheta, smallDeltaTheta, deltaAlpha, smallDeltaAlpha, thin, thick, pale, dark*]
Inputs: Parameters defining the style of the template.
deltaTheta, smallDeltaTheta, deltaAlpha, smallDeltaAlpha, thin, thick, pale, dark.
Example:
```
degrees=Pi/180.;
deltaTheta=15 degrees;
smallDeltaTheta=5 degrees;
deltaAlpha=15 degrees;
smallDeltaAlpha=5 degrees;
(* Absolute thickness *)
thin=0.25;
```

thick=0.35;
(* Gray levels *)
pale=0.75;
dark=0.35;
Output: The list of graphic elements of the EPN.
(*) EPN is the Equiareal Projection Net.

Code: P12
Description: It generates a list with graphic primitives for plotting an arc without 'tics'.
Syntax: **plotCurveAndTemplate**[*curve, template*]
Inputs:
curve: List of graphic primitives of the curve.
template: List of graphic primitives of the EPN (generated by P11).
Output: Combined plot of curve and template.

Sequence Processing (SP)

BLOCK DESCRIPTION
Sequence manipulations.

BLOCK FUNCTIONS

Code: SP01
Description: From a list *s*, whose elements are also lists, this function constructs another list with the *na*_th elements of the elements of *s*.
Example:
With s={{3,2,9},{6,7,-4},{1,8,9},{5,3,4}} and na=2 the function gives the list {2,7,8,3}.
Syntax: **selectArgument**[*s, na*]
Inputs:
s: A list whose elements are also lists.
na: Ordinal number of the selected element.
Output: The list with the *na*_th elements of the elements of *s*.

Code: SP02
Description: It composes two sequences.
Syntax: **composeSequences**[*G1, G2*]
Inputs:
G1: A sequence.
G2: A sequence.
Output: A sequence made by the elements of *G1* first and *G2* last.

Change of reference (CHR)

BLOCK DESCRIPTION
IJK are the kinematic axes for the description of the deformation and NET the

geographical axes. The function defined in this block can be used to obtain IJK coordinates from NET coordinates.

BLOCK FUNCTIONS

Code: CHR01
Description: This function computes the change matrix from NET axis to IJK axis.
Syntax: **computeBasisChangeMatrix**[*axis1Angles, alternateDirection13Angles*]
Inputs:
  axis1Angles: Angles corresponding to the direction of the kinematic axis 1.
  alternateDirection13Angles: Angles corresponding to a direction u contained in the plane 13.
Output: 3x3 change matrix.

Code: CHR03
Description: A plane can be defined by its dip direction or by its normal vector. This function calculates the dip direction angles from the normal vector angles. The same function is used in the opposite conversion.
Syntax: **LMPvsNanglesChange**[*inputAngles*]
Inputs:
  inputAngles: Angles of the normal vector (respectively dip direction).
Output: Dip direction angles (respectively normal vector).

Code: CHR04
Description: Using the NET coordinate system, this function obtains the Cartesian coordinates of a direction vector specified by its angles {alpha, theta}.
Syntax: **computeNETcoordinates**[*angles*]
Inputs:
  a*ngles*: The list {alpha, theta} of the angles that define a direction u.
Output: The list {u_1, u_2, u_3} of the Cartesian components of the direction vector u.

Code: CHR05
Description: This function obtains the IJK coordinates of a direction vector from the NET coordinates.
Syntax: **fromNETcoordinatesToIJKcordinates**[*NETcoordinates, changeMatrix*]
Inputs:
  NETcoordinates: The list of the NET coordinates of a direction u.
  changeMatrix: The change matrix (obtained by CHR01).
Output: The list of IJK coordinates of the same direction u.

Auxiliary functions (AF)

BLOCK DESCRIPTION
Several auxiliary functions.

BLOCK FUNCTIONS

Code: AF01
Description: Given two directions by the corresponding angles, it calculates the acute angle between them.
Syntax: **angleBetweenDirections**[*par1, par2*]
Inputs:
par1: The list {alpha1, theta1} of angles of the first direction.
par2: The list {alpha2, theta2} of angles of the second direction.
Output: The angle between the directions (degrees).

Code: AF02
Description: This function computes the principal directions of the strain corresponding to a given gradient matrix.
Syntax: **principalDirections**[*mC*]
Inputs:
mC: The 3x3 gradient matrix.
Output: It prints several geometrical data of the principal directions.

Code: AF03
Description: A principal direction is represented by the angles {*alpha*, *theta*}. This function calculates the *alpha* angle of the principal direction number *pdN*.
Syntax: **alphaPrincipalDirection**[*mC, pdN*]
Inputs:
mC: The 3x3 gradient matrix.
pdN : The number of the principal direction for which the angle is calculated.
Output: The alpha angle of the principal direction number *pdN*.

Code: AF04
Description: A principal direction is represented by the angles {*alpha*, *theta*}. This function calculates the *theta* angle of the principal direction number *pdN*.
Syntax: **thetaPrincipalDirection**[*mC, pdN*]
Inputs:
mC: The 3x3 gradient matrix.
pdN : The number of the principal direction for which the angle is calculated.
Output: The *theta* angle of the principal direction number *pdN*.

Code: AF05
Description: Given a gradient matrix mC, this function calculates the ratio R_{13}=Sqrt[lambda1]/Sqrt[lambda3] between eigenvalues.
Syntax: **ratioR13**[*mC*]
Inputs:
mC: The 3x3 gradient matrix.
Output: The ratio R_{13}=Sqrt[lambda1]/Sqrt[lambda3].

Code: AF06
Description: Given a gradient matrix mC, this function calculates the ratio R_{12}=Sqrt[lambda1]/Sqrt[lambda2] between eigenvalues.
Syntax: **ratioR12**[*mC*]

Inputs:
    mC: The 3x3 gradient matrix.
Output: The ratio R_{12}=Sqrt[lambda1]/Sqrt[lambda2].

Code: AF07
Description: Given a gradient matrix mC, this function calculates the ratio R_{23}=Sqrt[lambda2]/Sqrt[lambda3] between eigenvalues.
Syntax: **ratioR23**[*mC*]
Inputs:
    mC: The 3x3 gradient matrix.
Output: The ratio R_{23}=Sqrt[lambda2]/Sqrt[lambda3].