Simulator for Active Origami : Variables

This documentation gives a summary of all variables used in the simulation package. We will focus on the functionality & sizes of these variables.

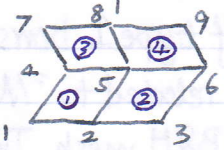
node 0: [oldNodeNum, 3]

the 3 nodal coordinates of original input geometry.

panel 0: {oldPanelNum}

the node numbers in each panel of original input geometry.

i.e. $\begin{Bmatrix} [1\ 2\ 5\ 4] \\ [2\ 3\ 6\ 5] \\ \vdots \end{Bmatrix}$



oldCreaseNum: int

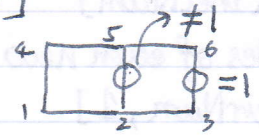
total number of creases in the original pattern.

oldCreaseConnect: [oldCreaseNum, 2]

the two nodes of each old crease.

oldCreaseType: [oldCreaseNum, 1]

$\begin{cases} =1: \text{Boundary Crease} \\ \neq 1: \text{Internal Crease} \end{cases}$

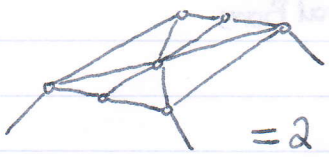


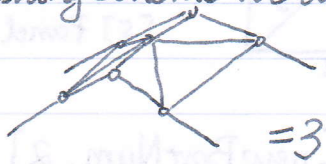
{modelGeometryConstant}:

this variable basically stores geometrical input of the pattern.

{1}: flag2D3D

this variable controls the meshing scheme we use.





{2}: compliantCreaseOpen

$\begin{cases} =1: \text{use compliant creases for analysis.} \\ =0: \text{use concentrated creases for analysis.} \end{cases}$

{3}: creaseWidthMat [oldCreaseNum, 1]

storing the width of each crease in the original pattern

i.e. creaseWidthMat(3) = 300 * 10^(-6);

{4}: panelInnerBarStart

the bar number of the first type 5 bar;

{5}: centerNodeStart

the node number of the first node at center of panel;

{6}: type1BarNum

the total number of type 1 bars.

Note: {4} {5} {6} are related to "Mesh_CompliantCreaseGeometry" used to generate the B&H mesh. They are also used latter for retrieving input data.

newNode: [newNodeNum, 3]

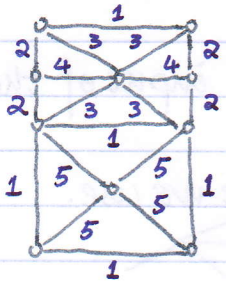
the nodal coordinates of new nodes;

newPanel: {newPanelNum}

the tree nodes of each new panel after meshing;

barType: [newBarNum, 1]

this vector stores type of bars



[1] Panel Boundary Bars

[2] Crease Vertical Bars

[3] Crease Diagonal Bars

[4] Crease Horizontal Bars

[5] Panel Diagonal Bars

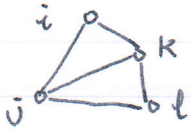
barConnect: [newBarNum, 2]

the number of 2 nodes at the end of each bars.

defined based on the new mesh after generating new pattern.

sprIJKL : [newBarNum, 4]

records the four adjacent node numbers to calculate the rotation angle of rotational springs.



[1] → i [3] → k
[2] → j [4] → l

newNode2OldNode : [newNodeNum, 1]

map the new node back to the old number, the value is 0 if there is no map (i.e. node at center of panel).

newCrease2OldCrease : [newBarNum, 1]

map the new bar number back to the old bar/crease.

newPanel2OldPanel : [newPanelNum, 1]

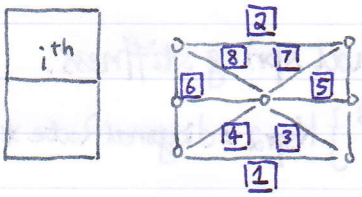
map the new panel number to the old panel.

newPanelNum : int

total number of new panels after meshing.

creaseRef : [oldCreaseNum, 8]

map the old crease to the 8 new bars in the crease region.



i^{th} row [1] [2] [3] [4] [5] [6] [7] [8]
the new crease number of bars.

barLength : [newBarNum, 1]

records the length of each new bars.

{modelMechanicalConstant}:

this cell stores the input parameters used to calculate the mechanical properties of the origami.

{1}: panelE

the Young's modulus of panel;

{2}: creaseE

the Young's modulus of creases;

{3}: panelPoisson

the Poisson's ratio of panels;

{4}: creasePoisson

the Poisson's ratio of creases;

{5}: panelThicknessMat [oldPanelNum, 1]

stores the thickness of each panel, The variable is defined based on old pattern before meshing.

{6}: creaseThicknessMat [oldCreaseNum, 1]

stores the thickness of each crease. The variable is defined based on old pattern before meshing.

{7}: diagonalRate:

the factor used to scale rotational spring stiffness.

Kspr1: for bar type 1 & 4 springs
Kspr2: for bar type 3 springs
} Kspr2 = diagonalRate * Kspr1

{8}: panelW

this variable is used to calculate spring stiffness for panels. It should be set to have a value that is comparable crease width.

{9}: contactOpen

controls if we will check the panel contact.

=1: check panel contact. =0: do not check

{10}: ke

potential scaling factor. Scale size of potential for contact.

{11}: d0edge

{12}: d0center

the distance for contact initiation at the edge of panel or at the center of the panel.

{13}: rotationZeroStrain. [oldCreaseNum, 1]

stores the zero strain position of the creases. Defined based on the old pattern before meshing.

{14}: totalFoldingNum:

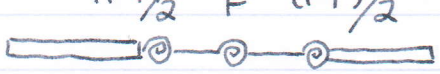
the total number of folding sequence.

{15}: foldingSequence: [oldCreaseNum, 1]

stores the self folding sequence of the creases. Defined based on the old pattern before meshing.

{16}: zeroStrainDistributionFactor

indicates how the zero strain angle is distributed in the compliant crease model.



we need this because we split a single crease to 3 lines of creases

barArea: [newBarNum , 1]

stores the areces for formulating the bar element.

sprK: [newBarNum, 1]

rotational stiffness of each spring element.

sprTargetZeroStrain: [newBarNum, 1]

the stress free fold angle of each rotational spring.

sprFoldingSequence: [newBarNum, 1]

the folding sequence of each rotational spring.

Here we introduce loading & support related variables:

supportInfo:

{1} supp: [suppNodeNum, 4]

first index gives the nodal number

following 3 indeces tell which Dof is supported.

1 means supported; 0 means not.

{2} elasticSuppOpen:

1 means non-rigid support is active.

0 means non-rigid support is not active.

{3} suppElastic [elasticSuppNum, 3]

first index: nodal number;

second index: direction of elastic support;

third index: support stiffness.

load: [loadNodeNum, 4]

first index: nodal number;

second-forth indeces: loading force in each direction;

assembleConstant: [3, 1]

(1) increStep: int

total step took to achieve the self assemble

(2) tor: double

tolerance of the iteration solver.

(3) itermax: int (well, matlab does not care about int & double)

the maximum number of iteration allowed.

loadConstant: [4,1]

(1) increStep:
total number of increment

(2) tor:
tolerance for each iteration to stop.

(3) iterMax:
maximum number of iteration allowed.

(4) lambdaBar
an input for the MGDCM. starting step length.

Next a brief intro of plotting control variables:

viewControl: [4,1]

(1) viewAngle1: e.g. 45

(2) viewAngle2: e.g. 45

(3) displayRange: e.g. 300×10^{-3}

(4) displayRate: e.g. 0.5

Here we give a brief summary of all variables involved in the calculation of mechanical self-folding & loading.

U: [newNodeNum, 3]
Deformation matrix that stores the current deformation field of origami. This variable is continuously updated throughout the entire analysis.

Ex: [newBarNum, 1].
the strain of bar elements.

Sx: [newBarNum, 1]
the second Piola-Kirchhoff stress of bar elements

C: [newBarNum, 1]
the tangent modulus of bar elements.

Tbar: [newNodeNum x 3, 1]
the inner force vector from bar elements

Kbar: [newNodeNum x 3, newNodeNum x 3]
the stiffness matrix of bar elements.

theta: [newBarNum, 1]
the rotation angle of each rotational spring element.

M: [newBarNum, 1]
the bending/rotational moment from each rotational spring elements.

sprKadj: [newBarNum, 1]
the adjusted tangent stiffness for rotational spring.
Note: the [sprK] is adjusted to [sprKadj] such that when a crease/rot spring is folded to 180°, it will stop!

Tspr: [newNodeNum x3, 1]
the inner force vector from rotational spring element.

Kspr: [newNodeNum x3, newNodeNum x3]
the global stiffness matrix from rotational springs.

The following variables are related to self-contact models implemented by the package:

Point, T1, T2, T3: [3,1]
the nodal coordinates of the Point, and the three nodes of the triangle (for Contact).

Dd2x2, Ddx: size depends on zones
the Hessian matrix and the gradient vector of the Point to Triangle distance. The sizes of them depend on which zones the points fall in.

Tcontact: [newNodeNum x3, 1]
the inner force vector from panel contact.

Kcontact: [newNodeNum x3, newNodeNum x3]
the stiffness matrix from panel contact.

Next, we introduce variables related to the solution of origami heat transfer problem.

{modelThermalConstant}:
This cell stores the input variables for setting up the heat transfer problem within the active origami.

{1}: panelConductivityMat: [oldPanelNum, 1]
stores the thermal conductivity of panels; based on the numbering before meshing of origami.

{2}: creaseConductivity:
thermal conductivity of creases;

{3}: evironmentConductivity:
thermal conductivity of submerged environment;

{4}: thickEnvMat: [oldPanelNum, 1]
the thermal model assumes that at t_{env} away from panels, the submerged enviroment is at room temperature.

{5}: thickEnvCrease:
same as {4}, but this is for creases;

{6}: envLayer:
number of 1-D thermal elements used to discretize the structure-environment heat transfer;

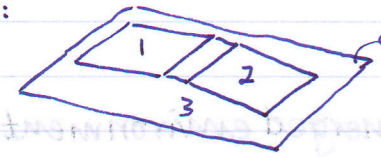
{7} thermalDissipationAngle:
the α angle used to construct the 1-D thermal conduction problem between structure & environment;

{8} envTemp:
temperature of submerged enviroments;

{9}: thermalBoundaryPanels: [ ] *
numbers of panels used as thermal boundary;

{10}: roomTempNode: [ ]
numbers of nodes at room temperature;

Note:



→ thermal B.C. panels [3] as input.

Next, we introduce the input used to set up the Timoshenko's bi-material morph systems.

{modelTimoshenkoConstant}:

{1}: deltaAlpha:
the differential thermal expansion between the two material of bimorph actuator;

{2}: mat1E:
Young's modulus of first material;

{3}: mat2E:
Young's modulus of second material;

{4}: t1:
thickness of the first layer;

{5}: t2:
thickness of the second layer;

Note: The Timoshenko's model gives the curvature estimation of bimaterial morph when subjected to elevating temperature. (Timoshenko. Analysis of Bi-metal Thermalstats)

Next are the intermidiate variables generated during the analysis that are significant.

thermalMat: [thermalNodeNum × (envLayer+1), same]
this is the thermal conductivity matrix, including all contributions from within structure & structure-enviroment heat transfer.

thermalNodeNum: int
total number of node (dofs) used for the within structure heat transfer.

qin: [thermalNodeNum × (envLayer+1), 1]
vector that stores the input heating energy.

T: [size(indexArray), 1]
vector that stores the temperature profile.
this vector only contains free degrees of freedom.

indexArray: [ ] (size depends on BC).
this vector returns the indeeies of the free degrees of freedom.

Tcrease: double.
Averaged Temperature of creases.

Next we introduce variables used for plotting the origami loading process after the analysis.

UhisAssemble: [assembleStep, newNodeNum, 3]
stores the history of deformation field during the entire self-folding assemble process.

UhisLoading: [loadingStep, newNodeNum, 3]
stores the history of deformation field during the mechanical loading process of origami.

strainEnergyAssemble: [assembleStep, 4]
stores the strain energy history of the self-assemble process.
[1]: Crease Bending Energy.
[2]: Panel Bending Energy.
[3]: Panel Stretching Energy.
[4]: Crease Stretching Energy.

strainEnergyLoading: [loadingStep, 4]
stores the strain energy history of the mechanical loading process of origami.

loadHis: [loadingStep, 1]
stores the history of load.
Note: the load can be sett up to store different quantities depending on what is needed. (There are a lot of ways to represent "load"). DOUBLE CHECK Solver Code!

nodeForce: [newNodeNum, 3]
stores the nodal reaction at end step of loading.

loadForce: [newNodeNum, 3]
stores the loading force at end step of mechanical loading.

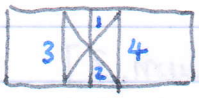
contautForce: [newNodeNum, 3]
Storce contact force at end step of mechanical loading.

angleHis: [assembleStep + loadingStep, 1]

records the folding angle of a given crease during the entire analysis period.

angleNum: [4, 1].

Stores the four number of creases to calculate angleHis



the 4 location. Based on the meshed configuration.

Note: the two variables are used in the function: Plot_DeformedHisAndCalcTheta.

temperatureHistory: [thermalNodeNum, thermalStep]

stores the time history of temperature profiles of the origami structure.

Note: actually thermalNodeNum is just the total number of node in origami.

UhisThermal: [thermalStep, thermalNodeNum, 3]

stores the deformation history during the thermal loading of the origami.

energyHisThermal: [thermalSteps, 4]

stores the strain energy history of origami during the thermal loading.