Package ‘FEA’

January 20, 2025

Type Package

Title Finite Element Modeling for R

Version 0.0.2

Author Henna D. Bhramdat

Maintainer Henna D. Bhramdat <bhramdath@ufl.edu>

Description Finite element modeling of beam structures and 2D geometries using constant strain triangles. Applies material properties and boundary conditions (load and constraint) to generate a finite element model. The model produces stress, strain, and nodal displacements; a heat map is available to demonstrate regions where output variables are high or low. Also provides options for creating a triangular mesh of 2D geometries. Package developed with reference to: Bathe, K. J. (1996). Finite Element Procedures.[ISBN 978-0-9790049-5-7] -- Seshu, P. (2012). Textbook of Finite Element Analysis. [ISBN-978-81-203-2315-5] -- Mustapha, K. B. (2018). Finite Element Computations in Mechanics with R. [ISBN 9781315144474].

License GPL-2 | GPL-3

Encoding UTF-8

RoxygenNote 7.2.1

Imports geometry, geosphere, ptinpoly, sp, MASS, graphics

Depends R (>= 3.5.0)

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2023-01-10 22:33:10 UTC

Contents

`ApplyBC.2d` *ApplyBC.2d*

Description

Boundary constraint for element centroids based on coordinate points. For the x & y direction per centroid create matrix with boundary 1(unfixed) or 0(fixed).

Usage

```
ApplyBC.2d(meshP, BoundConx, BoundCony)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh
`BoundConx`	Boundary constraint for nodes in the x-direction
`BoundCony`	Boundary constraint for nodes in the y-direction

Value

A data frame with constraint parameters applied to each node in the x and y directions. Formatted for use in reduced element matrix.

`NodeKnownL`	Constraint parameters

Examples

```
data(triMesh)

meshP = triMesh$MeshPts$p
BoundConx = BoundCony = numeric(NROW(meshP))
BoundConx[1:NROW(meshP)] = BoundCony[1:NROW(meshP)] = 1
BoundConx[c(10, 11, 12)] = BoundCony[c(10, 11, 12)] = 0

bound = ApplyBC.2d(meshP, BoundConx, BoundCony)
```

AutoAdjust.2d *AutoAdjust.2d*

Description

Allows for automatic refinement of the triangular mesh generated based on given parameters. Will remove elements that are outside the margin of the geometry.

Usage

```
AutoAdjust.2d(meshP, meshT, edge, centroid, AspectR, AR)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`meshT`	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
`edge`	Coordinate points of the initial geometry.
`centroid`	Matrix (2 x n) of triangle elements.
`AspectR`	Aspect ratio of each triangle element.
`AR`	maximum desired aspect ratio, numeric value.

Value

Generates new mesh and centroid tables

`Meshpts`	Includes both new mesh coordinate points and triangulation of points.
`Centroids`	Centroid positions for each triangle element.

Examples

```
data(triMesh)
data(polyshape)
data(dime)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
edge =  polyshape$Line
centroid = triMesh$Centroids
AspectR = dime$AspectRatio
AR = 10

auto = AutoAdjust.2d(meshP, meshT, edge, centroid, AspectR, AR)
```

`beamApplyBC` *beamApplyBC*

Description

Boundary constraint for element centroids based on coordinate points. For the x & y direction per centroid create matrix with boundary 1(unfixed) or 0(fixed).

Usage

```
beamApplyBC(beamP, BCtran, BCrot)
```

Arguments

`beamP`	Matrix (2 x n) of beam coordinates.
`BCtran`	Boundary constraint for nodes to translate in x or y directions. Input as a non-matrix column.
`BCrot`	Boundary constraint for nodes to rotate. Input as a non-matrix column.

Value

A data frame with constraint parameters applied to each node for directional translation and rotation. Formatted for use in reduced element matrix.

`NodeKnownL`	Matrix (1 x n) of constraint parameters

Examples

```
data(beamGeo)

beamBC = beamApplyBC(beamGeo$beamP, beamGeo$BCtran, beamGeo$BCrot)
```

`beamBC` *Boundary conditions applied to each node. Obtained from function: beamApplyBC*

Description

Boundary conditions applied to each node. Obtained from function: beamApplyBC

Usage

```
beamBC
```

Format

An object of class `matrix` (inherits from `array`) with 8 rows and 1 columns.

beamDime *Dimensional data for beam elements. Includes area, length, aspect ratio, angles and lengths of elements. Obtained from function: beamDimensions*

Description

Dimensional data for beam elements. Includes area, length, aspect ratio, angles and lengths of elements. Obtained from function: beamDimensions

Usage

```
beamDime
```

Format

An object of class `list` of length 7.

---

beamDimensions *beamDimensions*

Description

Calculates input dimensions needed for beam finite element.

Usage

```
beamDimensions(Y, G, Nu, beamP, beamT, thick,  fx, fy)
```

Arguments

`Y`	Elastic modulus value for material (Pa).
`G`	Shear modulus value for material (Pa). If using Euler-Bernoulli model, G = 0.
`Nu`	Poisson's ratio value for material.
`beamP`	Matrix (2 x n) of beam coordinates.
`beamT`	Matrix (2 x n) containing the number of the coordinate point as shown in beamP that connect to form a given beam (Discretization table).
`thick`	Thickness of the beam
`fx`	Load value (newtons) in the x direction.
`fy`	Load value (newtons) in the y direction.

Value

Calculates values needed for both Timoshenko-Ehrenfest and Euler-Bernoulli beam theories.

`k`	Timoshenko-Ehrenfest correction
`Length`	Beam length
`Angle`	Beam angle within the plane
`MomentofInertia`	Moment of Inertia for each beam
`Displacement`	Displacement under Timoshenko-Ehernfest beam theory
`RotationAngle`	Angle of rotation
`StiffnessAngle`	Stiffness angle

Examples

```
data(beamGeo)

DOF = 4
n = NROW(beamGeo$beamT)
thick = matrix(c(0.039149, 0.03, 0.0246625), ncol = 1, nrow = n) #height(thickness) of beam

beamDime = beamDimensions(beamGeo$Y, beamGeo$G, beamGeo$Nu, beamGeo$beamP, beamGeo$beamT,
                          beamGeo$thick, beamGeo$fx, beamGeo$fy)
```

`beamElementMat` *beamElementMat*

Description

Generates element stiffness matrix for beams.

Usage

```
beamElementMat(beamP, beamT, Y, Length, MoI)
```

Arguments

`beamP`	Matrix (2 x n) of beam coordinates.
`beamT`	Matrix (2 x n) containing the number of the coordinate point as shown in beamP that connect to form a given beam (Discretization table).
`Y`	Elastic modulus value for material.
`Length`	Length of beams.
`MoI`	Moment of inertia for each beam segment.

Value

Generates initial element matrix needed for the finite element model.

`beamEmat`	An element matrix of the beam

Examples

```
data(beamGeo)
data(beamDime)

Length = beamDime$Length
MoI = beamDime$MomentofInertia

beamEmat = beamElementMat(beamGeo$beamP, beamGeo$beamT, beamGeo$Y, Length, MoI)
```

`beamEmat` *List of element matrices for each element. Obtained from function: beamElementMat*

Description

List of element matrices for each element. Obtained from function: beamElementMat

Usage

```
beamEmat
```

Format

An object of class `list` of length 3.

`beamExMat` *List of element matrices for each element. Obtained from function: beamElementMat*

Description

List of element matrices for each element. Obtained from function: beamElementMat

Usage

```
beamExMat
```

Format

An object of class `list` of length 3.

beamExpandEM *beamExpandEM*

Description

Expanded element matrix for beam.

Usage

```
beamExpandEM(beamP, beamT, ElementMat)
```

Arguments

`beamP`	Matrix (2 x n) of beam coordinates.
`beamT`	Matrix (2 x n) containing the number of the coordinate point as shown in beamP that connect to form a given beam (Discretization table).
`ElementMat`	Element stiffness matrix list.

Value

produces large (n x n) element matrix from initial element matrix.

`beamExMat`	The expanded element matrix

Examples

```
data(beamGeo)
data(beamEmat)

ElementMat = beamEmat
beamExMat =  beamExpandEM(beamGeo$beamP, beamGeo$beamT, ElementMat)
```

beamForceVector *beamForceVector*

Description

Creates a matrix of loads for beams in the x & y direction for each load unconstrained node.

Usage

```
beamForceVector(beamP, fx, fy, NodeKnownL)
```

Arguments

`beamP`	Matrix (2 x n) of beam coordinates.
`fx`	Load vector (newtons) in the x-direction.
`fy`	Load vector (newtons) in the y-direction.
`NodeKnownL`	Data frame with constraint parameters applied to each node in the x and y directions. Formatted for use in reduced element matrix. Generated from ApplyBC function.

Value

Produces a matrix with loading parameters for each node.

`ReducedFV`	Reduced force vector matrix containing the model load parameters.

Examples

```
data(beamGeo)
data(beamUDL)

NodeKnownL = beamBC
FV = beamForceVector(beamGeo$beamP, beamGeo$fx, beamGeo$fy, NodeKnownL)
```

`beamFV` *Load vector produced from function function: beamForceVector*

Description

Load vector produced from function function: beamForceVector

Usage

```
beamFV
```

Format

An object of class `matrix` (inherits from `array`) with 5 rows and 1 columns.

`beamGeo` *Sample geometry for beam. Includes shape, discretization table, boundary conditions, thickness, and material details.*

Description

Sample geometry for beam. Includes shape, discretization table, boundary conditions, thickness, and material details.

Usage

```
beamGeo
```

Format

An object of class `list` of length 12.

`beamGLforce` *Global and Local loading force matrices obtained from function: beamGLForces*

Description

Global and Local loading force matrices obtained from function: beamGLForces

Usage

```
beamGLforce
```

Format

An object of class `list` of length 2.

beamGLForces *beamGLForces*

Description

Uses nodal displacements to determine global and local forces at each node

Usage

```
beamGLForces(beamP, beamT, Y, MoI, Length, GMat, BUDL, BND)
```

Arguments

beamP	Matrix (2 x n) of beam coordinates.
beamT	Matrix (2 x n) containing the number of the coordinate point as shown in beamP that connect to form a given beam (Discretization table).
Y	Elastic Modulus of material
MoI	Moment of Inertia
Length	Length of beam
GMat	Global stiffness matrix
BUDL	Column matrix for beam distributed load
BND	beam nodal displacement, output from function "beamNodeDis"

Value

Matrices of global and local forces. (Results in kN)

GForce	Large global force matrix.
Lforce	Large local force matrix.

Examples

```
data(beamGeo)
data(beamDime)
data(beamsUDL)
data(beamND)
data(beamGloMat)

Length = beamDime$Length
MoI = beamDime$MomentofInertia
BUDL = beamsUDL
BND = beamND
GMat = beamGloMat

GLforce = beamGLForces(beamGeo$beamP, beamGeo$beamT, beamGeo$Y, MoI, Length, GMat, BUDL, BND)
```

beamGlobalEM *beamGlobalEM*

Description

Generates global stiffness matrix for beams.

Usage

```
beamGlobalEM(beamExEM)
```

Arguments

beamExEM	Expanded Element Matrix

Value

Produces large (n x n) global matrix

GlobalMat	Global matrix

Examples

```
data(beamExMat)

beamExEM = beamExMat
GMat = beamGlobalEM(beamExEM)
```

beamGloMat *Global element matrix, obtained from function: beamGlobalEM*

Description

Global element matrix, obtained from function: beamGlobalEM

Usage

```
beamGloMat
```

Format

An object of class `matrix` (inherits from `array`) with 8 rows and 8 columns.

beamND *Global nodal displacement, obtained from function: beamNodeDis*

Description

Global nodal displacement, obtained from function: beamNodeDis

Usage

```
beamND
```

Format

An object of class `list` of length 3.

beamNodeDis *beamNodeDis*

Description

Calculates global nodal displacements of beam.

Usage

```
beamNodeDis(beamP, BCtran, BCrot, REM, NodeKnownL, ForceV)
```

Arguments

`beamP`	Matrix (2 x n) of beam coordinates.
`BCtran`	Boundary constraint for nodes to translate in x or y directions.
`BCrot`	Boundary constraint for nodes to rotate.
`REM`	Reduced element matrix, returned from function ReducedEM.
`NodeKnownL`	data frame with constraint parameters applied to each node in the x and y directions. Formatted for use in reduced element matrix. Generated from ApplyBC function.
`ForceV`	Reduced force vector matrix containing the model load parameters. Returned from function ForceVector.

Value

Produces tables with new node coordinates that are produced by the geometry under an applied load.

`NodeDis`	Nodal displacement
`GlobalND`	Nodal displacement in the global environment
`GlobalNDMatrix`	Nodal displacement in the global environment as a reduced matrix

Examples

```
data(beamGeo)
data(beamFV)
data(beamREM)
data(beamBC)

ForceV = beamFV
REM = beamREM
NodeKnownL = beamBC

beamND = beamNodeDis(beamGeo$beamP, beamGeo$BCtran, beamGeo$BCrot, REM, NodeKnownL, ForceV)
```

`beamPlotSystem` *beamPlotSystem*

Description

Generates heat map for given stress or strain on the beam geometry. Threshold values for the color must be assigned.

Usage

```
beamPlotSystem(beamP, beamT, PlotVal, a, b, c, d, e, f, g, h, i, j,
                    Oc, ac, bc, cc, dc, ec, fc, gc, hc, ic, jc, LWD)
```

Arguments

`beamP`	Matrix (2 x n) of beam coordinates.
`beamT`	Matrix (2 x n) containing the number of the coordinate point as shown in beamP that connect to form a given beam (Discretization table).
`PlotVal`	Value to be plotted, either stress or strain, return from function beamLocalStress function.
`a`	Threshold 1
`b`	Threshold 2
`c`	Threshold 3
`d`	Threshold 4
`e`	Threshold 5
`f`	Threshold 6
`g`	Threshold 7
`h`	Threshold 8
`i`	Threshold 9
`j`	Threshold 10

`Oc`	Color for all zero values
`ac`	Color 1
`bc`	Color 2
`cc`	Color 3
`dc`	Color 4
`ec`	Color 5
`fc`	Color 6
`gc`	Color 7
`hc`	Color 8
`ic`	Color 9
`jc`	Color 10
`LWD`	Line (beam) width

Value

Plot of colored beam based on the plot value

Examples

```
data(beamGeo)
data(beamStressResult)

PlotVal = beamStressResult

Oc = "slateblue"; ac = "steelblue2"; bc = "cyan2"; cc = "palegreen2";
dc = "darkolivegreen1"; ec = "lemonchiffon"; fc = "lightgoldenrod1";
gc = "gold"; hc= "lightsalmon"; ic= "tomato"; jc= "firebrick3"

a = 1e5;  b = 5e5;  c = 1e6;  d = 5e6;  e = 1e7;  f = 5e7;  g = 1e8;  h = 5e8; i = 1e9; j =5e9
beamPlotSystem(beamGeo$beamP, beamGeo$beamT, PlotVal, a, b, c, d, e, f, g, h, i, j, Oc,
ac, bc, cc, dc, ec, fc, gc, hc, ic, jc, LWD = 4)
```

`beamReducedEM` *beamReducedEM*

Description

Reduced stiffness matrix - use boundary condition to reduce matrix to smaller form by removing systems that are bound.

Usage

```
beamReducedEM(GMat, NodeKnownL)
```

Arguments

GMat	Global stiffness matrix
NodeKnownL	data frame with constraint parameters applied to each node in the x and y directions. Formatted for use in reduced element matrix. Generated from ApplyBC function.

Value

Produces a large matrix.

ReducedEM	Reduced element matrix.

Examples

```
data(beamBC)
data(beamGloMat)

NodeKnownL = beamBC
GMat = beamGloMat
beamREM = beamReducedEM(GMat, NodeKnownL)
```

`beamREM` *Reduced element matrix calculated from the expanded element matrix. Obtained from function: beamReducedEM*

Description

Reduced element matrix calculated from the expanded element matrix. Obtained from function: beamReducedEM

Usage

```
beamREM
```

Format

An object of class `matrix` (inherits from `array`) with 5 rows and 5 columns.

beamStress *beamStress*

Description

Calculates local stress and strain for beam elements

Usage

```
beamStress(beamP, beamT, Y, Length, MoI, RotAng, BND)
```

Arguments

beamP	Matrix (2 x n) of beam coordinates.
beamT	Matrix (2 x n) containing the number of the coordinate point as shown in beamP that connect to form a given beam (Discretization table).
Y	Value of Young's (Elastic) modulus
Length	Length of beam
MoI	Moment of Inertia
RotAng	Angle of rotation
BND	Global nodal displacement matrix, return from function beamNodeDis

Value

Completes FEM by calculating values of stress and strain, produces three (3) [3 x n] matrix.

BendingStress	Bending Stress

Examples

```
data(beamGeo)
data(beamGLforce)

Length = beamDime$Length
MoI = beamDime$MomentofInertia
RotAng = beamDime$Angle
BND = beamND

beamBendStress = beamStress(beamGeo$beamP, beamGeo$beamT, beamGeo$Y, Length, MoI, RotAng, BND)
```

`beamStressResult` *FEA results for the beam model. Obtained from function: beamStress*

Description

FEA results for the beam model. Obtained from function: beamStress

Usage

```
beamStressResult
```

Format

An object of class `numeric` of length 3.

`beamsUDL` *Uniformly distributed load on beam surface*

Description

Uniformly distributed load on beam surface

Usage

```
beamsUDL
```

Format

An object of class `list` of length 3.

`beamUDL` *beamUDL*

Description

Uniformly distributes load over the length of the beam.

Usage

```
beamUDL(beamP, beamT, Length, fx, fy)
```

Arguments

`beamP`	Matrix (2 x n) of beam coordinates.
`beamT`	Matrix (2 x n) containing the number of the coordinate point as shown in beamP that connect to form a given beam (Discretization table).
`Length`	Length of beam.
`fx`	Load value (newtons) in the x direction.
`fy`	Load value (newtons) in the y direction.

Value

Produces matrix representing uniformly distributed load on beam

`DLMatrix`	Column matrix for beam distributed load
`ExpandedDLMatrix`	Expanded beam distribution load
`ReductedDLMatrix`	Reduced beam distribution load

Examples

```
data(beamGeo)
data(beamDime)

Length = beamDime$Length
beamUDL = beamUDL(beamGeo$beamP, beamGeo$beamT, Length, beamGeo$fx, beamGeo$fy)
```

`bound` *Boundary conditions applied to each node. Obtained from function: ApplyBC*

Description

Boundary conditions applied to each node. Obtained from function: ApplyBC

Usage

```
bound
```

Format

An object of class `matrix` (inherits from `array`) with 100 rows and 1 columns.

`Cart` *Sample geometry. Matrix with x and y coordinates for initial shape.*

Description

Sample geometry. Matrix with x and y coordinates for initial shape.

Usage

```
Cart
```

Format

An object of class `matrix` (inherits from `array`) with 11 rows and 2 columns.

`cleanpoly` *Cleaned nodal distribution in and on polygon. Obtained from function: Threshpts*

Description

Cleaned nodal distribution in and on polygon. Obtained from function: Threshpts

Usage

```
cleanpoly
```

Format

An object of class `list` of length 2.

`dime` *Dimensional data for mesh elements. Includes area, length, aspect ratio, angles and lengths of elements. Obtained from function: Dimensions*

Description

Dimensional data for mesh elements. Includes area, length, aspect ratio, angles and lengths of elements. Obtained from function: Dimensions

Usage

```
dime
```

Format

An object of class `list` of length 6.

Dimensions.2d *Dimensions.2d*

Description

Calculates dimensional values for each triangular element, including truss length & angles, distance from nodal point to centroid, aspect ratio of each triangle element, and area of the triangle.

Usage

```
Dimensions.2d(meshP, meshT, centroid)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`meshT`	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
`centroid`	Matrix (2 x n) containing coordinate points of the centroid of each triangular element.

Value

Evaluation of triangle elements truss, angle, and area.

`Truss`	Nodal pairs that form each truss.
`TrussLength`	Distance between each paired nodes forming a truss, its length.
`Dist2Cent`	Shortest distance from truss to triangle centroid.
`Truss angle`	Angles of the triangle created from truss meeting.
`AspectRatio`	Aspect ratio of triangle elements.
`Area`	Area within triangle elements.

Examples

```
data(triMesh)
data(polyshape)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
centroid = triMesh$Centroids

dime = Dimensions.2d(meshP, meshT, centroid)
```

`displacN` *Global nodal displacement, obtained from function: NodeDis*

Description

Global nodal displacement, obtained from function: NodeDis

Usage

```
displacN
```

Format

An object of class `list` of length 2.

`ElementMat.2d` *ElementMat.2d*

Description

Generates an element stiffness matrix

Usage

```
ElementMat.2d(meshP, meshT, Nu, Y, Thick)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`meshT`	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
`Nu`	Value of Poisson's ratio for each element
`Y`	Value of Young's (Elastic) modulus for each element
`Thick`	Value of the thickness of the mesh, a positive value must be given.

Value

Generates initial element matrix needed for the finite element model.

`EMPStress`	An element matrix of the geometry under stress.
`EMPStrain`	An element matrix of the geometry under strain.

Examples

```
data(triMesh)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
Y = matrix(20e9, nrow = NROW(meshT))
Nu = matrix(0.45, nrow = NROW(meshT))
Thick = 0.001
DOF = 6

fea_EM = ElementMat.2d(meshP, meshT, Nu, Y, Thick)
```

`EulerBeamFEA` *EulerBeamFEA*

Description

Calculates stress in beam structures using the Euler-Bernoulli beam theory.

Usage

```
EulerBeamFEA(Y, beamP, beamT, fx, fy, BCtran, BCrot, Length, MoI, RotAng)
```

Arguments

`Y`	Elastic modulus value for material (Pa).
`beamP`	Matrix (2 x n) of beam coordinates.
`beamT`	Matrix (2 x n) containing the number of the coordinate point as shown in beamP that connect to form a given beam (Discretization table).
`fx`	Load value (newtons) in the x direction.
`fy`	Load value (newtons) in the y direction.
`BCtran`	Boundary constraint for nodes to translate in x or y directions. Input as a non-matrix column.
`BCrot`	Boundary constraint for nodes to rotate. Input as a non-matrix column.
`Length`	Length of beam.
`MoI`	Moment of inertia for each beam segment.
`RotAng`	Angle of rotation

Value

Calculates local forces and stresses, as well as bending stress for beams following the Euler-Bernoulli beam theory.

`Stress`	Local stress at node
`LocalLoad`	Local load at node
`BendingStress`	Bending Stress

Examples

```
data(beamGeo)
data(beamDime)

Length = beamDime$Length
MoI = beamDime$MomentofInertia
RotAng = beamDime$Angle

beamFEA = EulerBeamFEA(beamGeo$Y, beamGeo$beamP, beamGeo$beamT, beamGeo$fx, beamGeo$fy,
                        beamGeo$BCtran, beamGeo$BCrot, Length, MoI, RotAng)
```

`ExpandEM.2d` *ExpandEM.2d*

Description

Generates the expanded element matrix, which represents the contribution of individual finite elements towards the global structural matrix

Usage

```
ExpandEM.2d(meshP, meshT, centroid, EMatrixlist)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`meshT`	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
`centroid`	Matrix (2 x n) containing coordinate points of the centroid of each triangular element.
`EMatrixlist`	EMPStress or EMPStrain generated from ElementMat function. List of element matrices.

Value

Produces large (n x n) matrix.

`ExpandedMat`	The expanded element matrix

Examples

```
data(triMesh)
data(fea_EM)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
centroid = triMesh$Centroids
```

```
EMatrixlist = fea_EM$EMPStress

fea_ExEM = ExpandEM.2d(meshP, meshT, centroid, EMatrixlist)
```

ExpandSFT.2d *ExpandSFT.2d*

Description

Generates expanded surface force element matrix from SurfaceTraction function

Usage

```
ExpandSFT.2d(meshP, meshT, SurfTrac)
```

Arguments

meshP	Matrix (2 x n) containing coordinate points of the mesh nodes.
meshT	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
SurfTrac	List of surface forces.

Value

Produces a large (n x n) element matrix of surface forces.

ExpandedSurf	Expanded surface force element matrix.

Examples

```
data(triMesh)
data(SurfTrac)

meshT = triMesh$MeshPts$T
meshP = triMesh$MeshPts$p

expSurf = ExpandSFT.2d(meshP, meshT, SurfTrac)
```

expSurf — *Expanded element matrix for surface forces. Obtained from function: ExpandSFT*

Description

Expanded element matrix for surface forces. Obtained from function: ExpandSFT

Usage

```
expSurf
```

Format

An object of class `list` of length 50.

fea_EM — *List of element matrices for each element. Obtained from function: ElementMat*

Description

List of element matrices for each element. Obtained from function: ElementMat

Usage

```
fea_EM
```

Format

An object of class `list` of length 2.

fea_ExEM — *List of large expanded element matrices calculated from the element matrix. Obtained from function: ExpandEM*

Description

List of large expanded element matrices calculated from the element matrix. Obtained from function: ExpandEM

Usage

```
fea_ExEM
```

Format

An object of class `list` of length 78.

`fea_result` *FEA results. Produces list with results from local stresses including Stress, Strain, and Stress from Strain. Obtained from function: LocalStress*

Description

FEA results. Produces list with results from local stresses including Stress, Strain, and Stress from Strain. Obtained from function: LocalStress

Usage

```
fea_result
```

Format

An object of class `list` of length 3.

`FEMStrain.2d` *FEMStrain.2d*

Description

Creates a complete finite element model using strain for a given 2D mesh under specified boundary conditions (constrain and load).

Usage

```
FEMStrain.2d(meshP, meshT, centroid, BoundConx, BoundCony, SFShear,
SFTensile, Length, area, Fx, Fy, Y, Nu, Thick)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`meshT`	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
`centroid`	Matrix (2 x n) containing coordinate points of the centroid of each triangular element.
`BoundConx`	Boundary constraint for nodes in the x-direction
`BoundCony`	Boundary constraint for nodes in the y-direction

`SFShear` Magnitude of positive shear traction; if there is no surface traction then SFShear = 0

`SFTensile` Magnitude of tensile surface traction; if there is no surface traction then SFTensile = 0

`Length` Truss length

`area` Triangle element area

`Fx` Load vector for the x-direction

`Fy` Load vector for the y-direction

`Y` Value of Young's (Elastic) modulus

`Nu` Value of Poisson's ratio

`Thick` Value of the thickness of the mesh, a value must be given.

Value

Completes the FEM to generate values of stress and strain and nodal displacement.

`NodeDisplacement` Node displacement on each axis

`LocalStress` Stress as calucated from stress, strain, and stress from strain. Three (3) [3 x n] matrices where [x, y, tau]

Examples

```
data(triMesh)
data(dime)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
centroid = triMesh$Centroids
Y = matrix(20e9, nrow = NROW(meshT))
Nu = matrix(0.45, nrow = NROW(meshT))
Thick = 0.001
DOF = 6
BoundConx = BoundCony = numeric(NROW(meshP))
BoundConx[1:NROW(meshP)] = BoundCony[1:NROW(meshP)] = 1
BoundConx[c(10, 11, 12)] = BoundCony[c(10, 11, 12)] = 0
SFShear = 0
SFTensile = 0
Length = dime$TrussLength
area = dime$Area
Fx = 10
Fy = 10

fea_strain = FEMStrain.2d(meshP, meshT, centroid, BoundConx, BoundCony, SFShear, SFTensile,
             Length, area, Fx, Fy, Y, Nu, Thick)
```

FEMStress.2d *FEMStress.2d*

Description

Creates a complete finite element model using stress for a given 2D mesh under specified boundary conditions (constrain and load).

Usage

```
FEMStress.2d(meshP, meshT, centroid, BoundConx, BoundCony, SFShear,
SFTensile, Length, area, Fx, Fy,  Y, Nu, Thick)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`meshT`	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
`centroid`	Matrix (2 x n) containing coordinate points of the centroid of each triangular element.
`BoundConx`	Boundary constraint for nodes in the x-direction
`BoundCony`	Boundary constraint for nodes in the y-direction
`SFShear`	Magnitude of positive shear traction; if there is no surface traction then SFShear = 0
`SFTensile`	Magnitude of tensile surface traction; if there is no surface traction then SFTensile = 0
`Length`	Truss length
`area`	Triangle element area
`Fx`	Load vector for the x-direction
`Fy`	Load vector for the y-direction
`Y`	Value of Young's (Elastic) modulus
`Nu`	Value of Poisson's ratio
`Thick`	Value of the thickness of the mesh, a value must be given.

Value

Completes the FEM to generate values of stress and strain and nodal displacement.

`NodeDisplacement`	Node displacement on each axis
`LocalStress`	Stress as calucated from stress, strain, and stress from strain. Three (3) [3 x n] matrices where [x, y, tau]

Examples

```
data(triMesh)
data(dime)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
centroid = triMesh$Centroids
Y = matrix(20e9, nrow = NROW(meshT))
Nu = matrix(0.45, nrow = NROW(meshT))
Thick = 0.001
DOF = 6
BoundConx = BoundCony = numeric(NROW(meshP))
BoundConx[1:NROW(meshP)] = BoundCony[1:NROW(meshP)] = 1
BoundConx[c(10, 11, 12)] = BoundCony[c(10, 11, 12)] = 0
SFShear = 0
SFTensile = 0
Length = dime$TrussLength
area = dime$Area
Fx = 10
Fy = 10

fea_stress = FEMStress.2d(meshP, meshT, centroid, BoundConx, BoundCony, SFShear, SFTensile,
             Length, area, Fx, Fy, Y, Nu, Thick)
```

ForceVector.2d   *ForceVector.2d*

Description

Creates a matrix of loads in the x & y direction for each load unconstrained node.

Usage

```
ForceVector.2d(Fx, Fy, RSF, meshP, NodeKnownL)
```

Arguments

Fx	Load vector for the x-direction
Fy	Load vector for the y-direction
RSF	If surface traction is present assign value as the ReducedSF matrix; if there is no surface traction set RSF = 0
meshP	Matrix (2 x n) containing coordinate points of the mesh nodes.
NodeKnownL	data frame with constraint parameters applied to each node in the x and y directions. Formatted for use in reduced element matrix. Generated from ApplyBC function.

Value

Produces a matrix with loading parameters for each node.

ReducedFV Reduced force vector matrix containing the model load parameters.

Examples

```
data(triMesh)
data(reduc_SF)
data(bound)

meshP = triMesh$MeshPts$p
RSF = reduc_SF
Fx = 10
Fy = 10
NodeKnownL = bound

load = ForceVector.2d(Fx, Fy, RSF, meshP, NodeKnownL)
```

---

glfor *Global and Local loading force matrices obtained from function: GLForces*

Description

Global and Local loading force matrices obtained from function: GLForces

Usage

```
glfor
```

Format

An object of class `list` of length 2.

GLForces.2d *GLForces.2d*

Description

Uses nodal displacements to determine global and local forces at each node

Usage

```
GLForces.2d(meshP, meshT, GMat, GlobalND, EMatrixlist)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`meshT`	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
`GMat`	Global matrix
`GlobalND`	Global nodal displacement
`EMatrixlist`	Element matrix list

Value

Matrices of global and local forces

`GForce`	Large global force matrix.
`Lforce`	Large local force matrix.

Examples

```
data(triMesh)
data(gloMat)
data(displacN)
data(fea_EM)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
GMat = gloMat
GlobalND = displacN$GlobalND
EMatrixlist = fea_EM$EMPStress

glfor = GLForces.2d(meshP, meshT, GMat, GlobalND, EMatrixlist)
```

`GlobalMat.2d` *GlobalMat.2d*

Description

Generates global stiffness matrix - once established, the expanded element matrix must be combined to create the global structural stiffness matrix by adding the expanded matrices.

Usage

```
GlobalMat.2d(meshP, meshT, ExEM)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`meshT`	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
`ExEM`	Expanded element matrix

Value

Produces large (n x n) global matrix

`GlobalMat` Global matrix

Examples

```
data(triMesh)
data(fea_ExEM)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
ExEM = fea_ExEM

gloMat = GlobalMat.2d(meshP, meshT, ExEM)
```

---

gloMat *Global element matrix, obtained from function: GlobalMat*

Description

Global element matrix, obtained from function: GlobalMat

Usage

```
gloMat
```

Format

An object of class `matrix` (inherits from `array`) with 100 rows and 100 columns.

---

load *Load vector produced from function function: ForceVector*

Description

Load vector produced from function function: ForceVector

Usage

```
load
```

Format

An object of class `matrix` (inherits from `array`) with 94 rows and 1 columns.

LocalStress.2d *LocalStress.2d*

Description

Calculates local stress and strain for triangular elements of the mesh

Usage

```
LocalStress.2d(meshP, meshT, Y, Nu, GlobalND)
```

Arguments

meshP	Matrix (2 x n) containing coordinate points of the mesh nodes.
meshT	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
Y	Value of Young's (Elastic) modulus
Nu	Value of Poisson's ratio
GlobalND	Global nodal displacement, return from function NodeDis

Value

Completes FEM by calculating values of stress and strain, produces three (3) [3 x n] matrix.

Strain	Calculated strain. [x, y, tau]
Stress	Calculated stress in pascals. [x, y, tau]
StressStrain	Stress as calucated from strain. [x, y, tau]

Examples

```
data(triMesh)
data(displacN)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
Y = matrix(20e9, nrow = NROW(meshT))
Nu = matrix(0.45, nrow = NROW(meshT))
GlobalND = displacN$GlobalND

fea_result = LocalStress.2d(meshP, meshT, Y, Nu, GlobalND)
```

ManualAdjust.2d *ManualAdjust.2d*

Description

Allows for manual refinement of the triangular mesh generated based on given parameters. Will remove triangle elements that are identified in the input (loc).

Usage

```
ManualAdjust.2d(meshP, meshT, edge, centroid, loc)
```

Arguments

meshP	Matrix (2 x n) containing coordinate points of the mesh nodes.
meshT	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
edge	Coordinate points of the initial geometry.
centroid	Matrix (2 x n) of triangle elements.
loc	String containing the number of the meshT matrix row of the triangle chosen to be removed.

Value

Generates new mesh and centroid tables

Meshpts	Includes both new mesh coordinate points and triangulation of points.
Centroids	Centroid positions for each triangle element.

Examples

```
data(triMesh)
data(polyshape)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
edge =  polyshape$Line
centroid = triMesh$Centroids
loc = c(7, 35, 17)

ManualAdjust.2d(meshP, meshT, edge, centroid, loc)
```

`NodeDis.2d` *NodeDis.2d*

Description

Calculates global nodal displacements

Usage

```
NodeDis.2d(meshP, REM, ForceV, NodeKnownL)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`REM`	Reduced element matrix, returned from function ReducedEM.
`ForceV`	Reduced force vector matrix containing the model load parameters. Returned from function ForceVector.
`NodeKnownL`	data frame with constraint parameters applied to each node in the x and y directions. Formatted for use in reduced element matrix. Generated from ApplyBC function.

Value

Produces tables with new node coordinates that are produced by the geometry under an applied load.

`NodeDis`	Nodal displacement
`GlobalND`	Nodal displacement in the global environment

Examples

```
data(triMesh)
data(load)
data(reduc_EM)
data(bound)

meshP = triMesh$MeshPts$p
REM = reduc_EM
ForceV = load
NodeKnownL = bound

displacN = NodeDis.2d(meshP, REM, ForceV, NodeKnownL)
```

PlotSystem.2d *PlotSystem.2d*

Description

Generates heat map for given stress or strain on the geometry. Threshold values for the color must be assigned.

Usage

```
PlotSystem.2d(meshP, meshT, PlotVal, a, b, c, d, e, f, g, h, i, j,
                  Oc, ac, bc, cc, dc, ec, fc, gc, hc, ic, jc)
```

Arguments

meshP	Matrix (2 x n) containing coordinate points of the mesh nodes.
meshT	Matrix (3 x n) containing the number of the coordinate point that forms a given triangle within the mesh.
PlotVal	Value to be plotted, either stress or strain, return from function LocalStress function.
a	Threshold 1
b	Threshold 2
c	Threshold 3
d	Threshold 4
e	Threshold 5
f	Threshold 6
g	Threshold 7
h	Threshold 8
i	Threshold 9
j	Threshold 10
Oc	Color 0
ac	Color 1
bc	Color 2
cc	Color 3
dc	Color 4
ec	Color 5
fc	Color 6
gc	Color 7
hc	Color 8
ic	Color 9
jc	Color 10

Value

Plot of colored polygon with mesh colored based on the plot value

Examples

```
data(triMesh)
data(fea_result)

meshP = triMesh$MeshPts$p
meshT = triMesh$MeshPts$T
PlotVal = abs(fea_result$Stress[,1])
Oc = "slateblue"; ac = "steelblue2"; bc = "cyan2"; cc = "palegreen2";
dc = "darkolivegreen1"; ec = "lemonchiffon"; fc = "lightgoldenrod1"; gc = "gold";
hc= "lightsalmon"; ic= "tomato"; jc= "firebrick3"
a = 1e5;  b = 5e5;  c = 1e6;  d = 5e6;  e = 1e7;  f = 5e7;  g = 1e8;  h = 5e8; i = 1e9; j =5e9

PlotSystem.2d(meshP, meshT, PlotVal, a, b, c, d, e, f, g, h, i, j,
          Oc, ac, bc, cc, dc, ec, fc, gc, hc, ic, jc)
```

`polyshape` *Sample geometry converted into a 2D polygon. Polygon data that specifies all coordinate, coordinates that are within the geometry and coordinates that construct the lines of the geometry. Obtained from function: SinglePoly*

Description

Sample geometry converted into a 2D polygon. Polygon data that specifies all coordinate, coordinates that are within the geometry and coordinates that construct the lines of the geometry. Obtained from function: SinglePoly

Usage

```
polyshape
```

Format

An object of class `list` of length 3.

ReducedEM.2d *ReducedEM.2d*

Description

Reduced stiffness matrix - use boundary condition to reduce matrix to smaller form by removing systems that are bound.

Usage

```
ReducedEM.2d(GMat, NodeKnownL)
```

Arguments

GMat	Global stiffness matrix
NodeKnownL	data frame with constraint parameters applied to each node in the x and y directions. Formatted for use in reduced element matrix. Generated from ApplyBC function.

Value

Produces a large matrix.

ReducedEM	Reduced element matrix.

Examples

```
data(gloMat)
data(bound)
GMat = gloMat
NodeKnownL = bound
reduc_EM = ReducedEM.2d(GMat, NodeKnownL)
```

ReducedSF.2d *ReducedSF.2d*

Description

Reduced matrix of surface forces

Usage

```
ReducedSF.2d(meshP, ExSurf)
```

Arguments

`meshP`	Matrix (2 x n) containing coordinate points of the mesh nodes.
`ExSurf`	Expanded surface matrix, output from ExpandSFT

Value

Produces a large matrix.

`RSF`	Produces a large, reduced surface force matrix

Examples

```
data(triMesh)
data(expSurf)
meshP = triMesh$MeshPts$p
ExSurf = expSurf
reduc_SF = ReducedSF.2d(meshP, ExSurf)
```

`reduc_EM` *Reduced element matrix calculated from the expanded element matrix. Obtained from function: ReducedEM*

Description

Reduced element matrix calculated from the expanded element matrix. Obtained from function: ReducedEM

Usage

```
reduc_EM
```

Format

An object of class `matrix` (inherits from `array`) with 94 rows and 94 columns.

`reduc_SF` *Reduced surface force matrix calculated from expanded element matrix. Obtained from function: ReducedSF*

Description

Reduced surface force matrix calculated from expanded element matrix. Obtained from function: ReducedSF

Usage

```
reduc_SF
```

Format

An object of class `matrix` (inherits from `array`) with 100 rows and 1 columns.

`SinglePoly.2d` *SinglePoly.2d*

Description

Generates a mesh for polygon with a single continuous geometry

Usage

```
SinglePoly.2d(x, y, ptDS, ptDL)
```

Arguments

`x`	X-coordinates for geometry.
`y`	Y-coordinates for geometry.
`ptDS`	Density of points desired within the geometry.
`ptDL`	Density of points desired at the perimeter of the geometry.

Value

Coordinate points of nodes distributed within and on the line of a given geometry.

`AllCoords`	all coordinate points distributed across the geometry.
`Within`	all coordinate points within the geometry ONLY.
`Line`	all coordinate points that lay on the perimeter of the geometry ONLY.

Examples

```
data(Cart)

x = Cart[,1]
y= Cart[,2]
ptDS = 30
ptDL = 20

polyshape = SinglePoly.2d(x, y, ptDS, ptDL)
```

SurfaceTraction.2d *SurfaceTraction.2d*

Description

Element Surface Traction - generates the column matrix for uniformly distributed surface traction. If surface traction is not present, assign SFTensile and SFShear a value of 0.

Usage

```
SurfaceTraction.2d(meshP, SFTensile, SFShear, Length, Thick, area)
```

Arguments

meshP	Matrix (2 x n) containing coordinate points of the mesh nodes.
SFTensile	Magnitude of tensile surface traction
SFShear	Magnitude of positive shear traction
Length	Truss length
Thick	Triangle element thickness
area	Triangle element area

Value

List of element matrices containing surface forces.

SurfT	List of surface forces for each element.

Examples

```
data(triMesh)
data(dime)

meshP = triMesh$MeshPts$p
SFShear = 0
SFTensile = 0
Thick = 0.001
```

```
Length = dime$TrussLength
area = dime$Area

SurfTrac = SurfaceTraction.2d(meshP, SFTensile, SFShear, Length, Thick, area)
```

SurfTrac — *List of element matrices with surface traction. Obtained from function: SurfaceTraction*

Description

List of element matrices with surface traction. Obtained from function: SurfaceTraction

Usage

```
SurfTrac
```

Format

An object of class `list` of length 50.

ThreshPts.2d — *ThreshPts.2d*

Description

Clean node distribution within or outside of geometry. Optional function for complex geometries.

Usage

```
ThreshPts.2d(coords, thresh, edge)
```

Arguments

`coords`	Nodal coordinates
`thresh`	Threshold for point removal. Ranges include: 500000-50000000
`edge`	Coordinate points of the initial geometry.

Value

Coordinate points of valid nodes.

`CleanedNodes`	Matrix of new nodes that abide by given threshold rules.
`NodeReport`	Report identifying with nodes were kept and which were removed.

Examples

```
data(polyshape)

coords = polyshape$Within
thresh = 5000000
edge = polyshape$Line

cleanpoly = ThreshPts.2d(coords, thresh, edge)
```

`triangulate0.2d` *triangulate0.2d*

Description

Triangulation by Delaunayn algorithm. Automatically generates a triangular mesh for a geometry containing nodal points.

Usage

```
triangulate0.2d(u0, edge)
```

Arguments

`u0`	Matrix (2 x n) of node coordinates within the geometry.
`edge`	Matrix (2 x n) of coordinate points on the perimeter of the geometry.

Value

Produces data for generated mesh.

`Meshpts`	Includes both new mesh coordinate points and triangulation of points.
`Centroids`	Centroid positions for each triangle element.

Examples

```
data(cleanpoly)
data(polyshape)

u0 = cleanpoly$CleanedNodes
edge = polyshape$Line

triMesh = triangulate0.2d(u0, edge)
```

`triMesh` *Meshed coordinate points obtained from function: triangulate0*

Description

Meshed coordinate points obtained from function: triangulate0

Usage

```
triMesh
```

Format

An object of class `list` of length 2.

Index