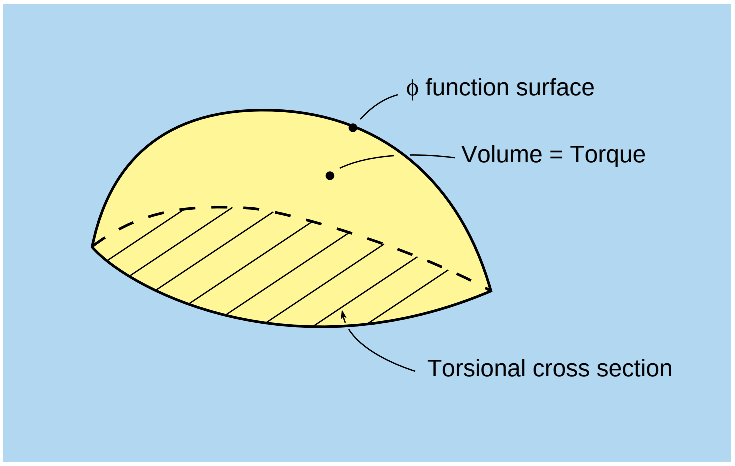
FETorsn

Finite Element Method
for Elastic Torsion
on TI's Graphing Calculators



Mark C. Bourland
mcb4588@omega.uta.edu

From “Reader” this text is searchable and the
data from the matrices may be copied.

FETorsn

Overview

FETorsn: Program. Finite element method applied to *elastic* torsion.[1]
Output: GTorsion.m; Loads.m; ShapMat.m

AnalyTor: Program. Analytical solution[2] (series) to the Poisson equation (torsion of rectangular cross sections).
Output: TorsionL.*l* (list: { ϕ, T, σ_{zx}, σ_{zy} })

FEMInput: Program. Mesh builder for torsion, plate and plane finite-element models.
Output: Joints.m; NodeVecs.m

MakeLMat: Program. “Loads” subprogram. Builds the “loads” matrices for quadratic-torsional elements.
Output: Loads.m (for quadratic elements, torsional problems).

Using FETorsn

Torsion-of-cross-section problems require solutions to the Poisson[3] equation:

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial x^2} = -2 \quad \text{in R}$$

$$\phi = 0 \quad \text{at the boundary of R.}$$

Applied to torsional problems, ϕ is the stress function, G is the shear modulus, and θ is the unit angle of twist.

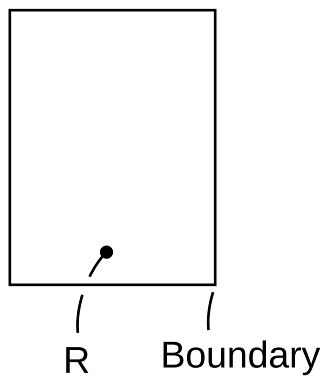
$$\nabla^2\phi = \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial x^2} = -2G\theta \quad \text{in R}$$

$\phi = 0$ at the boundary of R.

$$\text{where} \quad \sigma_{xz} = \frac{\partial\phi}{\partial y} \;; \quad \sigma_{yz} = -\frac{\partial\phi}{\partial x}$$

$$\gamma_{xz} = \frac{1}{G}\sigma_{xz} \;; \quad \gamma_{yz} = \frac{1}{G}\sigma_{yz}$$

(For linear elastic material.)



Using interpolating polynomials, approximate discreet values of the stress function $\phi[x,y]$ can be found with finite element methods.[4]

FETorsn uses these interpolating polynomials,

$\phi[x,y] = a_1 + a_2\, x + a_3\, y$
$\phi[x,y] = a_1 + a_2\, x + a_3\, y + a_4\, xy$
$\phi[x,y] = a_1 + a_2\, x + a_3\, y + a_4\, xy + a_5\, x^2 + a_6\, y^2$
$\phi[x,y] = a_1 + a_2\, x + a_3\, y + a_4\, xy + a_5\, x^2 + a_6\, y^2 + a_7\, x^2y + a_8\, xy^2$

and the following elements (C_o elements, boundary node rectangular and triangular families[5]):

linear triangle
bilinear rectangle
quadratic triangle
quadratic rectangle

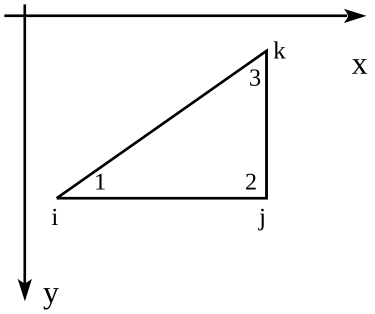
FETorsn creates a shape function matrix, ShapMat, a “torsion” matrix, TorMat, and then a “loads” matrix, QMat. It then assembles these element matrices into global matrices, GTormat and Loads, which allow for solution of discreet values of ϕ.

The shape function for an element can then be multiplied by the element’s nodal ϕ-values to find the unknown constants for the interpolation function. From the interpolating polynomial any value of ϕ or its derivatives (σ_{zx}, σ_{zy}) can be found for the element. (The linear triangle is an exception, its derivatives are constants.)

These values approximate and approach the exact solutions to the Poisson equation.

The simplest element, the linear triangle, is used here to the show the matrix algebra.

$$\text{ConsMat} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}$$



$$\text{LMat} = \begin{bmatrix} 1 & x & y \end{bmatrix}$$

$$\text{ShapMat} = \left[\text{LMat} * \text{ConsMat}^{-1} \right]^T$$

$$\text{TorMat} = \begin{bmatrix} \frac{\text{ShapMat}_i}{\partial x} & \frac{\text{ShapMat}_i}{\partial y} \\ \frac{\text{ShapMat}_j}{\partial x} & \frac{\text{ShapMat}_j}{\partial y} \\ \frac{\text{ShapMat}_k}{\partial x} & \frac{\text{ShapMat}_k}{\partial y} \end{bmatrix}$$

$$\text{Stiff} = \iint \text{TorMat} * \text{TorMat}^T \; dx\, dy$$

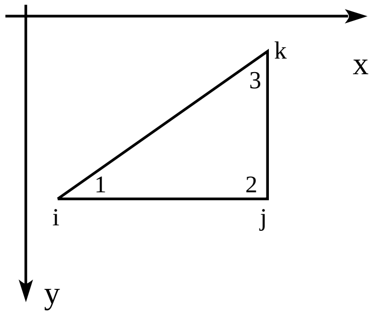
$$\text{GTorMat} = \sum_{E=1}^{nE} \text{Stiff}_E$$

$$\text{QMat} = \iint (2G\theta) * \text{ShapMat} \; dx\, dy$$

$$\text{Loads} = \sum_{E=1}^{nE} \text{QMat}_E$$

$$\{\phi\} = \text{GTorMat}^{-1} * \text{Loads}$$

$$\phi_E[x,y] = \text{ShapMat}_E^T * \{\phi\}_E \qquad (\text{for } E = 1 \text{ to } nE)$$



$$T = 2 \iint \phi \; dx\, dy = 2 \sum_{E=1}^{nE} \iint \phi_E[x,y] \; dx\, dy$$

For linear-triangular elements, $\iint \phi_E[x,y] \; dx\, dy = (A_E/3)(\phi_i + \phi_j + \phi_k)$

For bilinear-rectangular elements,

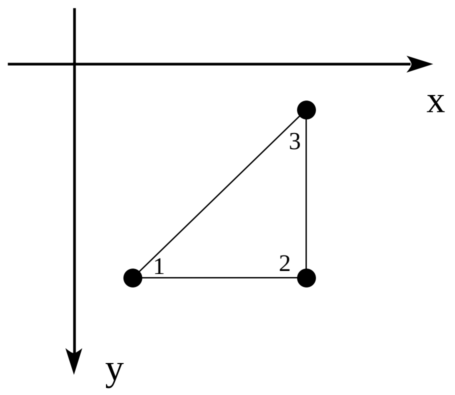
$$\iint \phi_E[x,y] \; dx\, dy = (A_E/4)(\phi_i + \phi_j + \phi_k + \phi_l)$$

$$\text{TorMat}_E^T * \{\phi\}_E = \begin{bmatrix} \dfrac{\partial\phi}{\partial x} \\ \dfrac{\partial\phi}{\partial y} \end{bmatrix}_E = \begin{bmatrix} -(\sigma_{zy}[x,y])_E \\ (\sigma_{zx}[x,y])_E \end{bmatrix}$$

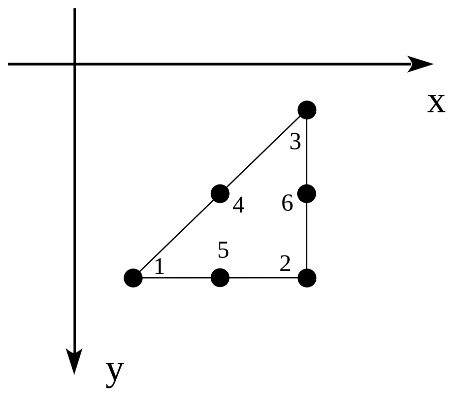
Integration is over the individual elements. For rectangular elements, and for the linear triangle, limits of integration are deduced automatically by *FETorsn*. The use of quadratic-triangular elements requires that integration limits be input for each of those elements. (Linear-triangular elements are integrated numerically.)

FETorsn requires that one mesh convention be used. That is element node numbers must be numbered in one of two ways.

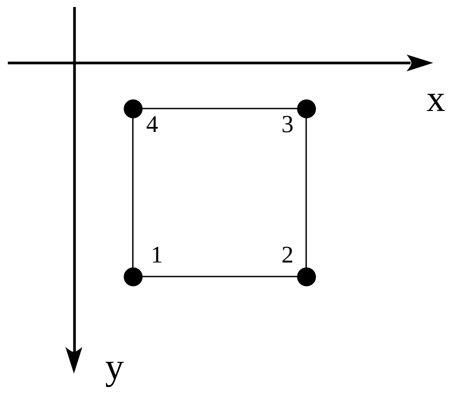
The first way is with the positive y-axis pointing downward, and the positive x-axis pointing to the right. Here are the coordinate axes and the four elements. The vertices must be numbered first as shown.



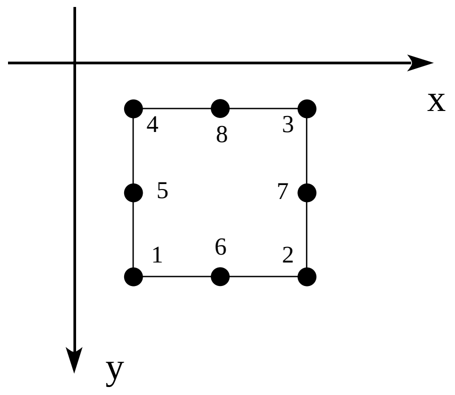
Linear Triangle



Quadratic Triangle



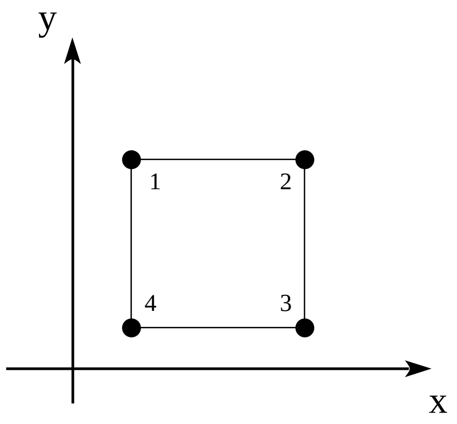
Bilinear Rectangle



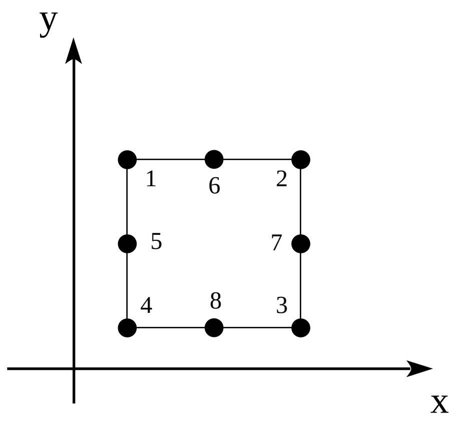
Quadratic Rectangle

The numbers here are node numbers. The black dots represent nodes. *FETorsn* requires that element nodes be numbered in this way, with this axes orientation, to correctly "auto-integrate" the rectangular elements.

The second way to number element nodes with *FETorsn* is with the positive y axis going up and the positive x axis going to the right, as shown below. (The triangular elements may be numbered as in the first case.)



Bilinear Rectangle



Quadratic Rectangle

Other conventions are as follows.

- Element nodes are numbered from 1 to n, where n is the number of nodes in the element.
- Elements themselves are numbered from 1 to n, and a box is drawn around the number, where n is the number of elements, nE.
- Joints, or global nodes, are numbered from 1 to n, and a circle is drawn around the number, where n is the number of joints, nJ.
- Possibles, or degrees of freedom, are numbered from 1 to n, and the numbers are preceded by a designator of displacement type, i.e. translation, rotation, or in the case of torsion, the stress function ϕ. Here n is the number of possibles, nP.

Example (based on Segerlind)

Find $(\phi)_{max}$. Section is a 1-cm-square shaft. G = 8E6 (N/cm^2) ; θ = 0.0001745 (rad/cm).

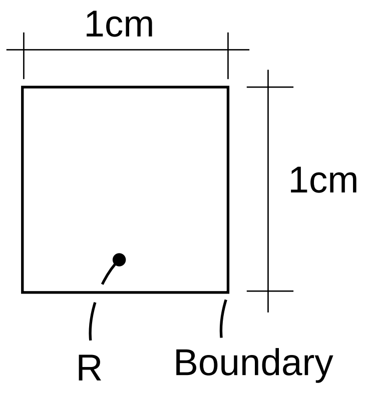
FEMInput may be used to create the “Joints” and “NodeVecs” matrices *FETorsn* needs to solve the problem (see “Using *FEMInput*” for details about its use). That is, *FETorsn* needs:

- The coordinates of the joints
- The mapping of element node numbers onto global node numbers (joints)
- The locations of the possibles (the degrees of freedom)

The use of *FEMInput* is straight forward. It precesses around the freebody: the joint coordinates are input, the “possibles” are input, finally “node vectors” are input.

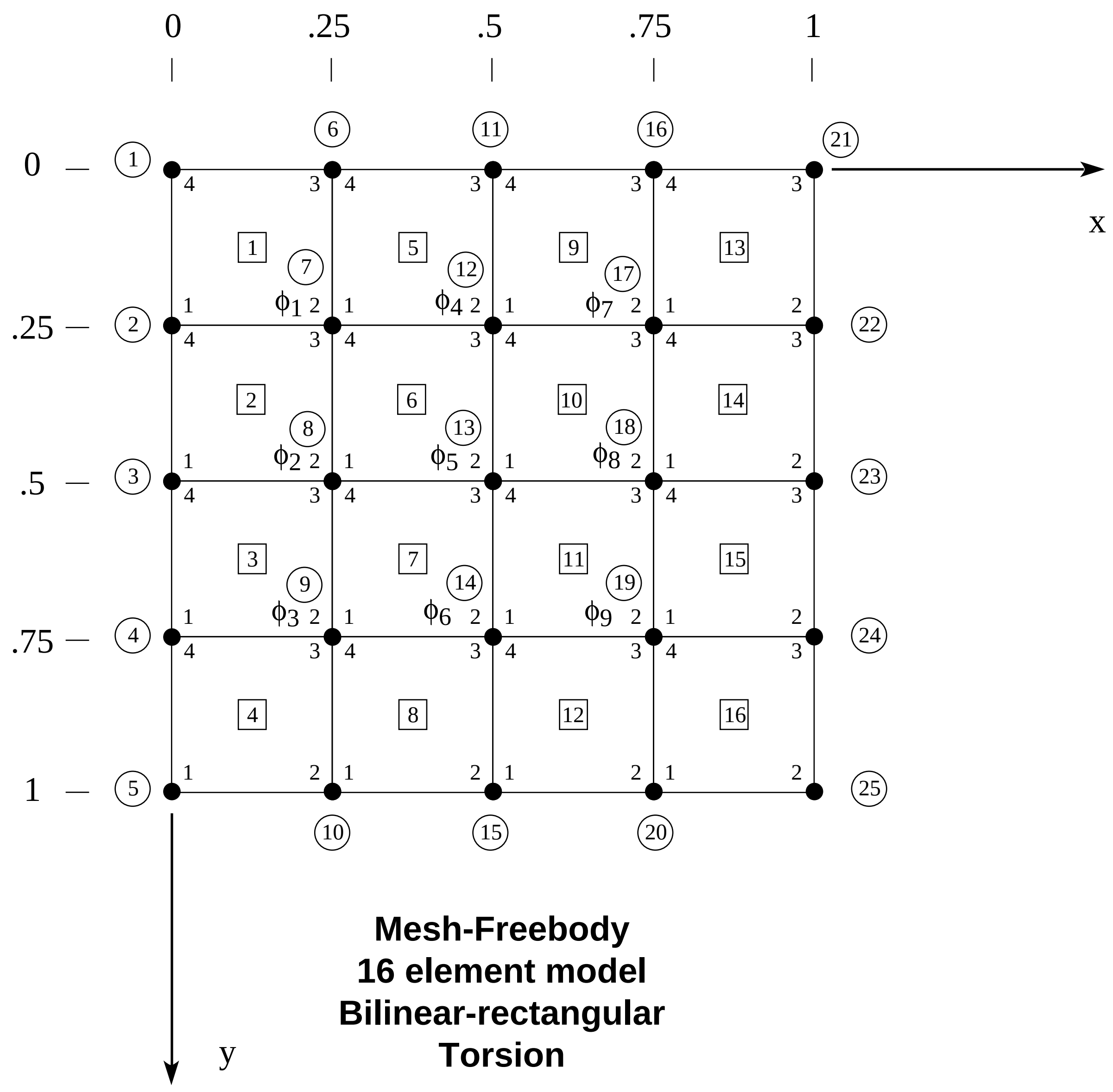
1. Boundary conditions

$\phi = -2\,G\theta$ at all points on the cross section not at the boundaries (R).
$\phi = 0$ at the boundaries of the cross section.



Square Shaft

Here is the freebody with numbering of joints, elements, nodes and “possibles.”

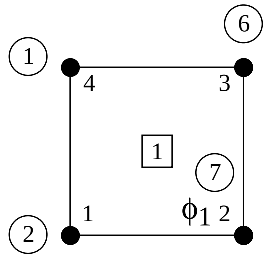


Mesh-Freebody
16 element model
Bilinear-rectangular
Torsion

The node vector matrix is a mapping of the element node numbers onto the joint numbers (global node numbers).

For example, the “node vector” of element 1 is

[2, 7, 6, 1]



FETorsn needs the Joints and NodeVecs matrices.

Run *FETorsn*.

> The number of elements nE is 16.
> The number of possibles nP is 9.
> The number of joints nJ is 25.
> Answer “1” (true) to make new global matrices (GTorsion.m and Loads.m)
> Answer “0” (false) to not keep copies of element matrices.
> Start with element 1 and end with element 16.

The resulting matrices are as follows. (“Joints” and “NodeVecs” matrices may be built using the TI’s matrix editor as well.)

From FEMInput:

Joint Nos.	x_i	y_i	ϕ_n
Joints =	0	0	0
	0	0.25	0
	0	0.5	0
	0	0.75	0
	0	1	0
	0.25	0	0
	0.25	0.25	1
	0.25	0.5	2
	0.25	0.75	3
	0.25	1	0
	0.5	0	0
	0.5	0.25	4
	0.5	0.5	5
	0.5	0.75	6
	0.5	1	0
	0.75	0	0
	0.75	0.25	7
	0.75	0.5	8
	0.75	0.75	9
	0.75	1	0
	1	0	0
	1	0.25	0
	1	0.5	0
	1	0.75	0
	1	1	0

Elem. Nos.	$(Nd_1)_i$	$(Nd_2)_i$	$(Nd_3)_i$	$(Nd_4)_i$
NodeVecs =	2	7	6	1
	3	8	7	2
	4	9	8	3
	5	10	9	4
	7	12	11	6
	8	13	12	7
	9	14	13	8
	10	15	14	9
	12	17	16	11
	13	18	17	12
	14	19	18	13
	15	20	19	14
	17	22	21	16
	18	23	22	17
	19	24	23	18
	20	25	24	19

The resulting global matrices are

$$\text{GTormat} = \begin{vmatrix} 2.6667 & -0.33337 & 0 & -0.33375 & -0.33333 & 0 & 0 & 0 & 0 \\ -0.33337 & 2.6665 & -0.33328 & -0.33333 & -0.333319 & -0.33336 & 0 & 0 & 0 \\ 0 & -0.3338 & 2.66696 & 0 & -0.333336 & -0.333368 & 0 & 0 & 0 \\ -0.33375 & -0.33333 & 0 & 2.66664 & -0.333334 & 0 & -0.33263 & -0.333337 & 0 \\ -0.3333 & -0.33319 & -0.33336 & -0.33334 & 2.666659 & -0.33328 & -0.333337 & -0.333267 & -0.333336 \\ 0 & -0.33336 & -0.33368 & 0 & -0.333328 & 2.666661 & 0 & -0.33336 & -0.3333293 \\ 0 & 0 & 0 & -0.33263 & -0.333337 & 0 & 2.66645 & -0.33321 & 0 \\ 0 & 0 & 0 & -0.33337 & -0.333267 & -0.33336 & -0.333321 & 2.666649 & -0.333328 \\ 0 & 0 & 0 & 0 & -0.333336 & -0.333293 & 0 & -0.333328 & 2.666586 \end{vmatrix}$$

$$\text{Loads} = \begin{array}{l} 0.125*G*\theta \\ 0.125*G*\theta \\ 0.125*G*\theta \\ 0.125*G*\theta \\ 0.125*G*\theta \\ 0.125*G*\theta \\ 0.125*G*\theta \\ 0.125*G*\theta \\ 0.125*G*\theta \end{array}$$

The ϕ-values are: GTorMat^{-1} * Loads =

$$\{\phi\} = \begin{array}{l} 0.096428*G*\theta \\ 0.120535*G*\theta \\ 0.096428*G*\theta \\ 0.120535*G*\theta \\ 0.155357*G*\theta \\ 0.120536*G*\theta \\ 0.096429*G*\theta \\ 0.120536*G*\theta \\ 0.096429*G*\theta \end{array}$$

(3 significant figures)

GTorMat, Loads and $\{\phi\}$ are ordered by “possible number.” That is the row numbers of the matrices correspond to the consecutive ϕ_n numbers, 1 to 9.

With G=8E6, and θ= 0.0001745, $(\phi)_{max}$ is found.

$$(\phi)_{max} = 0.155357 * (8\text{E}6) * 0.0001745 = 216.87 = 217$$

2. **Results Checking** (gauging model performance).

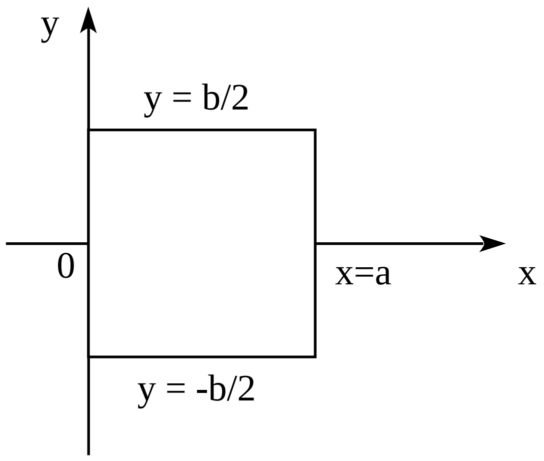
AnalyTor is a closed-form series solution to the Poisson equation for torsion of *rectangular* cross-sections. It's used here to check the performance of our model.

Conventions used by *AnalyTor*

- The freebody of the cross section must be oriented with its far left edge coincident with the y axis, which is positive pointing up, with respect to this page.
- The centroid of the section with respect to the y axis must be coincident on the x axis.
- G is the shear modulus.
- The symbol α is the unit angle of twist (note that α is given the symbol θ in *FETorsn*).
- The horizontal length, the width, is given the symbol a.
- The vertical length, the depth, is given the symbol b.
- The coordinate pair (x,y) describes a point on the cross section.
- The integer n is the number of terms used in the series portions of the solutions. (An n of nine (9) seems to work well.)

If *AnalyTor* is given a, b, n, x and y, and it returns ϕ, T, σ_{zx} and σ_{zy}, in that order. It then gives an option to find another set of solutions for another point. If "0" is given to the question "Find another point?", *AnalyTor* writes the solutions found to the list "TorsionL" and quits. (*AnalyTor* works with rectangular cross sections.)

Here's the freebody for the 1-cm-square-shaft problem.



After running *AnalyTor* and entering these data, 1, 1, .5, 0, 0, we can key-in the list name and the post-fix “with” operator to find the analytical value of ϕ.

TorsionL | G=8E6 and α=.0001745

The TI returns {205.69, 196.25, 0, 0}

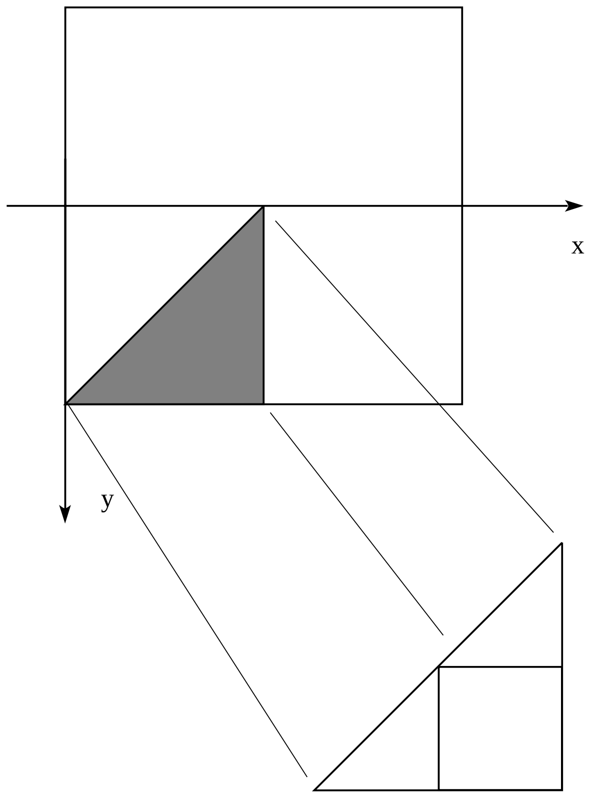
As expected, the shearing stresses are zero at the center of the cross section and the value of the stress function ϕ is 206.

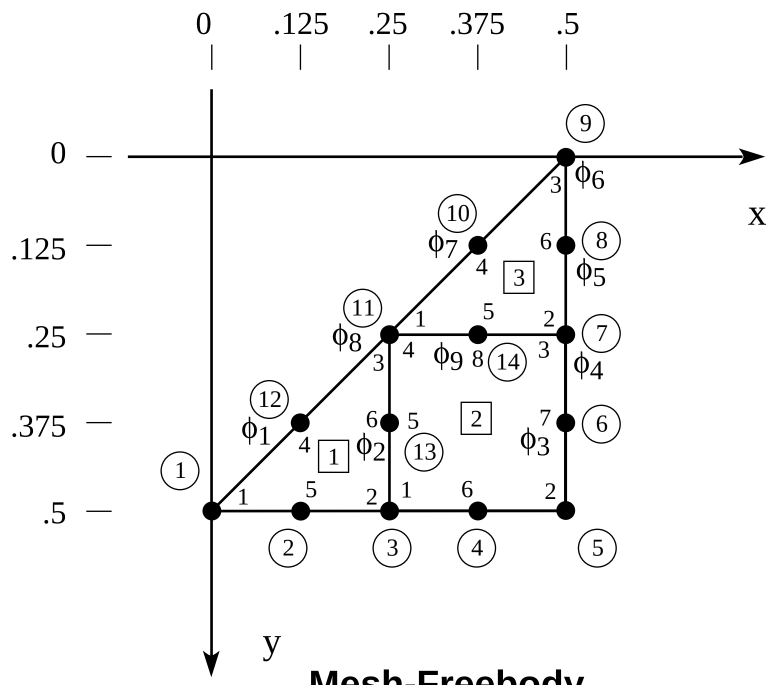
The error of the approximate value from the 16-element model we used is 5.4%.

3. A Better FE Model

Because our cross section is square, there is much symmetry with this problem. Only 1/8 of the section is needed to model the entire cross section.

We will use 1/8 of the cross section and divide it into 3 quadratic elements, two triangles and a rectangle. Graphically we have:





Mesh-Freebody
3 element model
Quadratic rect and tri-angular
Torsion

4. Limits Lists for Quadratic Triangles

When *FETorsn* finds a quadratic-triangular element, it asks for a list of integration limits. It expects a *list* in this form:

$$\{(x)_{lower}, (x)_{upper}, (y)_{lower}, (y)_{upper}\}$$

Numbers should be used, no symbols except y, because of the matrix algebra.

If the element has piecewise integration limits, then the limit list *FETorsn* expects is:

$$\{(x_1)_{lower}, (x_1)_{upper}, (y_1)_{lower}, (y_1)_{upper}, (x_2)_{lower}, \ldots\}$$

The limits are very important with respect to the QMat, as the sign of the integration can have a huge affect on the ϕ values.

For the 3-element model, the integration lists for the two quadratic-triangular elements are:

Element 1: {0.5 - y, 0.25, 0.25, 0.5}
Element 3: {0.5 - y, 0.5, 0, 0.25}

We can check the limits by integrating over them, huh?

$$A_{E_1} = \int_{0.25}^{0.5}\int_{0.5-y}^{0.25} \mathrm{d}x\,\mathrm{d}y = \int_{0.25}^{0.5} y - 0.25\ \mathrm{d}y = \frac{1}{32}$$

The *FEMInput*-generated matrices are:

$$\text{Joints} = \left|\begin{array}{lll}
0 & 0.5 & 0 \\
0.125 & 0.5 & 0 \\
0.25 & 0.5 & 0 \\
0.375 & 0.5 & 0 \\
0.5 & 0.5 & 0 \\
0.5 & 0.375 & 3 \\
0.5 & 0.25 & 4 \\
0.5 & 0.125 & 5 \\
0.5 & 0 & 6 \\
0.375 & 0.125 & 7 \\
0.25 & 0.25 & 8 \\
0.125 & 0.375 & 1 \\
0.25 & 0.375 & 2 \\
0.375 & 0.25 & 9
\end{array}\right| \qquad \text{NodeVecs} = \left|\begin{array}{llllllll}
1 & 3 & 11 & 12 & 2 & 13 & 0 & 0 \\
3 & 5 & 7 & 11 & 13 & 4 & 6 & 14 \\
11 & 7 & 9 & 10 & 14 & 8 & 0 & 0
\end{array}\right|$$

And the global matrices are:

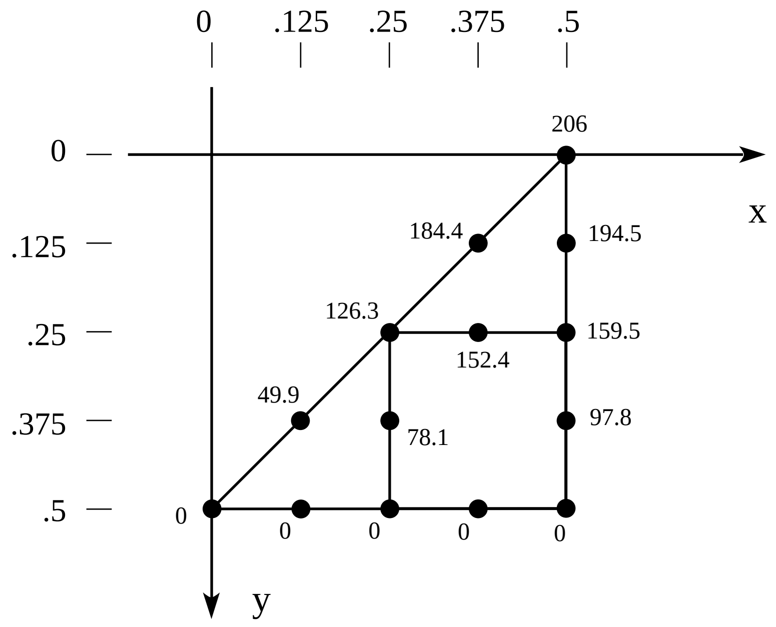
$$\text{GTorMat} = \left|\begin{array}{lllllllll}
2.666 & -1.3336 & 0 & 0 & 0 & 0 & 0 & 0. & 0 \\
-1.336 & 4.97778 & 0.35531 & -0.51123 & 0 & 0 & 0 & -1.48858 & -3.44E-11 \\
0 & 0.35531 & 2.311133 & -0.8222193 & 0 & 0 & 0 & -0.51142 & -6.3E-12 \\
0 & -0.51123 & -0.82193 & 2.155559 & -0.66666 & 0.16667 & -5E-13 & 0.66075 & -1.488955 \\
0 & 0 & 0 & -0.6666 & 2.6665 & -0.66668 & -1.333283 & 0. & 2.4E-12 \\
0 & 0 & 0 & 0.166667 & -0.66668 & 0.5 & 0. & 0. & 0. \\
0 & 0 & 0 & -5E-13 & -1.33283 & 0. & 2.66662 & 0. & -1.333328 \\
0. & -1.488858 & -0.51142 & 0.666075 & 0. & 0. & 0. & 2.15555735 & -1.488496 \\
0 & -3.44E-11 & -6.3E-12 & -1.488955 & 2.4E-12 & 0. & -1.33328 & -1.48496 & 4.977268
\end{array}\right|$$

$$\text{Loads} = \left|\begin{array}{l} 0.0208333293*G*\theta \\ 0.0624999965*G*\theta \\ 0.0416666672*G*\theta \\ -0.0104166728*G*\theta \\ 0.0208333331*G*\theta \\ 0. \\ 0.0208333331*G*\theta \\ -0.0104166714*G*\theta \\ 0.0624999978*G*\theta \end{array}\right|$$

And solving for $\{\phi\}$ we get:

$$\{\phi\}_i = \left|\begin{array}{l} 0.035776988112496*G*\theta \\ 0.055928976224736*G*\theta \\ 0.070083097865563*G*\theta \\ 0.1142528033412*G*\theta \\ 0.13932654613304*G*\theta \\ 0.14768446039742*G*\theta \\ 0.13205946039736*G*\theta \\ 0.090484872782214*G*\theta \\ 0.10916737466216*G*\theta \end{array}\right| = \begin{array}{l} 49.94 \\ 78.08 \\ 97.84 \\ 159.50 \\ 194.50 \\ 206.17 \\ 184.36 \\ 126.32 \\ 152.40 \end{array}$$

This approximation is very good.



	FETorsn	*AnalyTor*
ϕ_1	49.95	50.81
ϕ_2	78.06	78.59
ϕ_3	97.85	97.44
ϕ_4	159.50	159.98
ϕ_5	194.50	194.60
ϕ_6	206.17	205.69
ϕ_7	184.36	184.37
ϕ_8	126.32	126.44
ϕ_9	152.40	152.01

Stresses, strains and twisting moments

Memory Management

1. Using FETorsn with GraphLink

By using *FETorsn* it is not possible to know just how much algebra is required to do FE problems. It’s a lot. *FETorsn* builds global matrices on the fly and throws out the element matrices, so that there is more memory available, so that larger models can be used. Memory is going to be a problem, so *FETorsn* is set up to work piecewise. That is you can use it with GraphLink. Whenever *FETorsn* is paused, for input or otherwise, you can use GraphLink to move copies of matrices off of the TI and onto a disk. (You may want to store copies of the “Joints” and “NodeVecs” matrices too.)

For example you can “copy off” the element matrices as they are being built by telling *FETorsn* to “keep copies of element matrices.” Then, after each set of element matrices is made, there’s a prompt that asks whether it’s okay to delete them. During that prompt, or pause, you can copy off the element matrices onto a disk for later use.

Another example is with superposition of GTorMat and Loads matrices. Suppose you have a model with 75 possibles (nP=75). The possibles define GTorMat, that matrix would be 75 x 75. *FETorsn* allows you to divide that work up into say 3 parts. You could calculate 1/3 of the elements, then copy off the global matrices, calculate the next 3rd of the elements using a new set of global matrices, and so on. You’ll end up with 3 sets of global matrices that need to be superpositioned.

2. Post Management of Element Matrices

When copies of element matrices are moved onto a disk, and they are renamed with a manager other than the TI's, you should realize that those file-name changes will not necessarily change the “TI file name.” For example, you copy off a TorMat matrix for element 3, and then change the file name from TorMat to TorMat3 with your computer. Now if you use GraphLink to move TorMat3 onto your TI, your TI recognizes the file as TorMat, not TorMat3. The file name change you made on the computer is not recognized by the TI. To make file-name changes your TI will recognize, use the TI.

When *FETorsn* is told to keep copies of element matrices, it renames element matrices so that their names reflect which elements they refer. The names are changed and the program pauses with the message “Delete element matrices?” At this point you can copy off the element matrices with GraphLink then answer “1,” yes, or true, to the TI to let it delete the copies of the element matrices there. If you answer “0,” no, or false, to that prompt, then *FETorsn* keeps the element matrices with the element-specific names on your TI. Here are the names *FETorsn* assigns element matrices when the option of “keeping element matrices” is true:

TorMat	⇒	TorMt[i]
ShapMat	⇒	ShpMt[i]
Stiff	⇒	Stiff[i]
QMat	⇒	QMat[i]

(where [i] is the number of the element)

3. Limits on Model Size

Matrix column size on the TI is going to limit the size of your models. Another problem will be matrix algebra. The TI simplifies symbolic expressions as it works with them. (It’s possible for the TI to simplify an expression into a form that will return “undef” when integrated. Therefore, *FETorsn* uses the “Expand()” function in front of integrands.) One may notice that the TorMat matrices that are 5 x 2 and larger take a long time to “integrate.” It is actually the algebra that is taking up time. You can give *FETorsn* a quadratic rectangle with 8 possibles, take a copy of TorMat, and then multiply TorMat by its transpose to see how much time it takes. The TI’s double integration of matrices is speedy and impressive. Similarly, matrix inversion is relatively slow and takes much memory. You could end up with a GTorMat matrix that you can not invert. In this case you simply copy off the global matrices and do the algebra with some other application or use another method, Choleski decomposition for example, on the TI.

4. Stresses

To find the stresses in an element you multiply the element's ϕ values by its gradient. Another way is to differentiate ϕ[x, y] for the element.

With the gradient method, you first get the ϕ values. This is what we did for the square section with two models. Next we need the gradient matrix. The gradient matrix is the TorMat matrix. We choose which part of the cross section we are interested in and then use *FETorsn* to generate TorMat matrices for those elements. We can investigate element 2, the quadratic-rectangular element from the second model we built.

Our cross section is square so we'll find maximum shear stress at point (0.5, 0.5). We can find $(\tau)_{max}$.

We run *FETorsn* and answer the input as before accept we tell *FETorsn* to "keep element matrices" and we start with element 2 and end with element 2. When *FETorsn* asks whether it's okay to delete the element matrices we answer "0" for "No," or false. When *FETorsn* quits we'll find four element matrices: ShpMt2, TorMt2 , QMat2 and Stiff2.

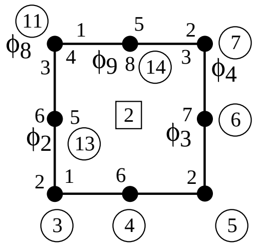
We'll take the ϕ values for element 2 at the nodes, put them in a column matrix and multiply them by the transpose of TorMt2. The TI will return a column matrix with two rows. In row one is dϕ/dx and in row two is dϕ/dy.

The second way to get the stresses is to differentiate ϕ[x, y]. As before we use *FETorsn* to get the element matrices we need. To make ϕ[x, y] for element 2 we need that element's ShapMat. We then take the element's ϕ values at the nodes including the zero values, put them into a column matrix, and multiply them by the transpose of the ShapMat. The TI returns a column matrix with a single element which contains an expression for ϕ[x, y].

Here are the matrix expressions.

Gradient method:

$$\text{TorMat}_E^T * \{\phi\}_E = \begin{bmatrix} \dfrac{\partial\phi}{\partial x} \\ \dfrac{\partial\phi}{\partial y} \end{bmatrix}_E = \begin{bmatrix} -(\sigma_{zy}[x,y])_E \\ (\sigma_{zx}[x,y])_E \end{bmatrix}$$



ϕ_1	49.95
ϕ_2	78.06
ϕ_3	97.85
ϕ_4	159.50
ϕ_5	194.50
ϕ_6	206.17
ϕ_7	184.36
ϕ_8	126.32
ϕ_9	152.40

Element 2

For Element 2:

TorMat = TorMt2

TorMt2 = [[-24. + 128.*x - 16.*y - 256.*x*y + 128.*y^2, 12. - 16.*x - 128.*x^2 - 64.*y + 256.*x*y]
[-72. + 128.*x + 208.*y - 256.*x*y - 128.*y^2, -72. + 208.*x - 128.*x^2 + 128.*y - 256.*x*y]
[32. - 192.*y + 256.*y^2, 96.000 - 192.*x - 256.*y + 512.*x*y]
[-32. + 192.*y - 256.*y^2, -48. + 192.*x + 128.*y - 512.*x*y]
[96.000 - 256.*x - 192.*y + 512.*x*y, 32. - 192.*x + 256.*x^2]]

Note that TorMt2 does not have terms for zero values of ϕ.

$\{\phi\}_2$ = [[159.49691][126.31688][78.07685][97.836004][152.397655]]

$\text{TorMt2}^T * \{\phi\}_2$ = [[1075.1405 - 2429.6338*x -1744.6313*y + 4859.2676*x*y - 811.2994*y^2]
[495.1218 - 1744.6313*x + 2429.6338*x^2 - 1503.9067*y - 1622.5988*x*y]]

$\sigma_{zx}[x, y]_2$ = 495.1218 - 1744.6313*x + 2429.6338*x^2 - 1503.9067*y - 1622.5988*x*y

$\sigma_{zy}[x, y]_2$ = –(1075.1405 - 2429.6338*x -1744.6313*y + 4859.2676*x*y - 811.2994*y^2)

$\sigma_{zx}[0.5, 0.5]_2 = -927.3884\ (\text{N/cm}^2)$

$\sigma_{zy}[0.5, 0.5]_2 = 0\ (\text{N/cm}^2)$

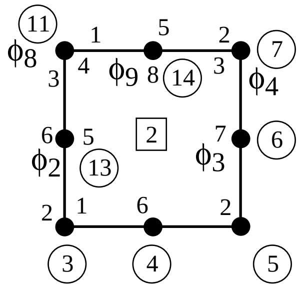
Compare with *AnalyTor*:

$\sigma_{zx}[0.5, 0.5]_2 = -948.24\ (\text{N/cm}^2)$

$\sigma_{zy}[0.5, 0.5]_2 = 0\ (\text{N/cm}^2)$

Differential method:

$$\phi_E[x,y] = \text{ShapMat}_E^T * \{\phi\}_E$$



ϕ_1	49.95
ϕ_2	78.06
ϕ_3	97.85
ϕ_4	159.50
ϕ_5	194.50
ϕ_6	206.17
ϕ_7	184.36
ϕ_8	126.32
ϕ_9	152.40

Element 2

For Element 2:

ShapMat = ShpMt2

ShpMt2 = [[x^2*(128.*y-32.)+x*(-128.*y^2-16.*y+12.)+64.*y^2-24.*y+2.]
[x^2*(128.*y-32.)+x*(128.*y^2-176.*y+36.)-32.*y^2+36.*y-7.]
[x^2*(-128.*y+64.)+x*(128.*y^2-16.*y-24.)-32.*y^2+12.*y+2.]
[x^2*(-128.*y+64.)+x*(-128.*y^2+208.*y-72.)+64.*y^2-72.*y+20.]
[x*(256.*y^2-192.*y+32.)-128.*y^2+96.*y-16.]
[x^2*(-256.*y+64.)+x*(192.*y-48.)-32.*y+8.]
[x*(-256.*y^2+192.*y-32.)+64.*y^2-48.*y+8.]
[x^2*(256.*y-128.)+x*(-192.*y+96.)+32.*y-16.]]

$\{\phi\}_2$ = [[0][0][159.49691][126.31688][78.07685][0][97.836004][152.397655]]

$\phi_2[x,y]$ = ShpMt2^T* $\{\phi\}_2$ = [[-59.57258 + 1075.14050*x - 1214.81691*x^2 +
495.12185*y - 1744.63131*x*y + 2429.63382*x^2*y -
751.95336*y^2 - 811.29940*x*y^2]]

$\sigma_{zx}[x, y]_2$= $d[\phi_2[x,y], y]$ = 495.12184 - 1744.63131*x +
2429.63382*x^2 - 1503.90673*y - 1622.59880*x*y

$\sigma_{zy}[x, y]_2$= $-d[\phi_2[x,y], x]$ = -1075.14050 + 2429.63382*x +
1744.63131*y - 4859.26763*x*y + 811.29940*y^2

$\sigma_{zx}[0.5, 0.5]_2$ = -927.3884 (N/cm^2)

$\sigma_{zy}[0.5, 0.5]_2$ = 0 (N/cm^2)

5. Strains

Once we have the stresses, strains are found by multiplying the stresses by the reciprocal of the shearing modulus.

$$\gamma_{xz} = \frac{1}{G}\sigma_{xz} \;; \qquad \gamma_{yz} = \frac{1}{G}\sigma_{yz}$$

(For linear elastic material.)

6. Twisting Moments

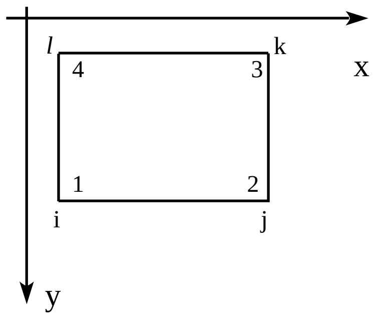
To get the torsional moment T, integrate the $\phi[x, y]$ expressions over the cross section, element by element. Here are the expressions.

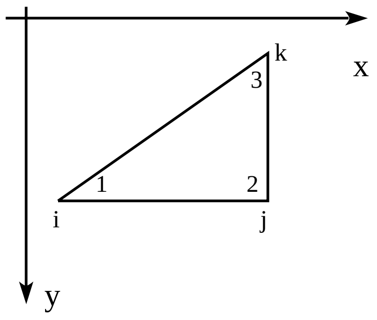
$$T = 2 \int\!\!\int \phi \; dx\, dy \; = \; 2 \sum_{E=1}^{nE} \int\!\!\int \phi_E[x,y] \; dx\, dy$$

For linear-triangular elements, $\int\!\!\int \phi_E[x,y] \; dx\, dy = (A_E/3)(\phi_i + \phi_j + \phi_k)$

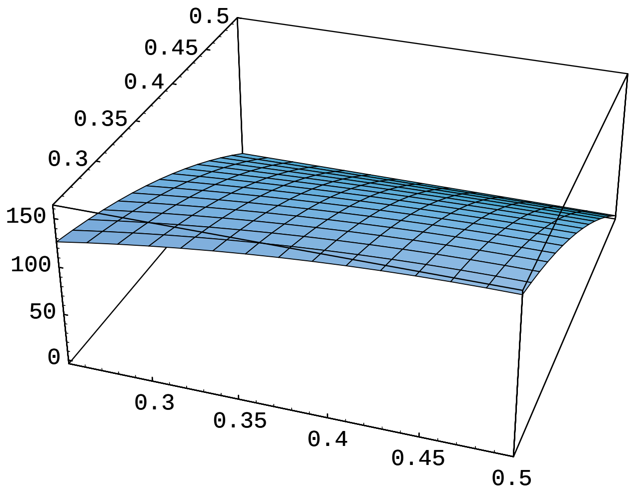
For bilinear-rectangular elements,

$$\int\!\!\int \phi_E[x,y] \; dx\, dy = (A_E/4)(\phi_i + \phi_j + \phi_k + \phi_l)$$

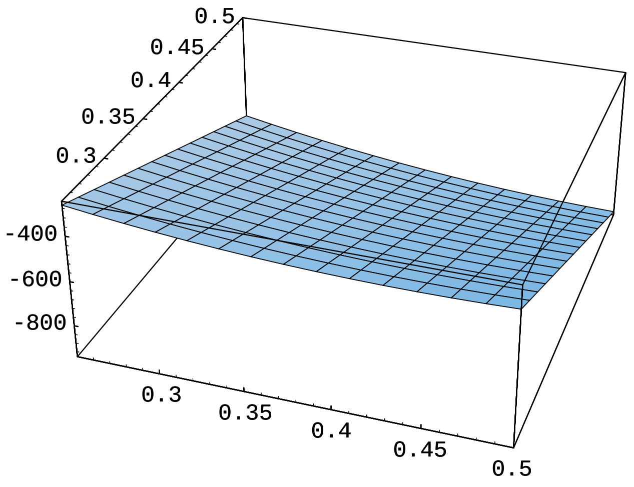




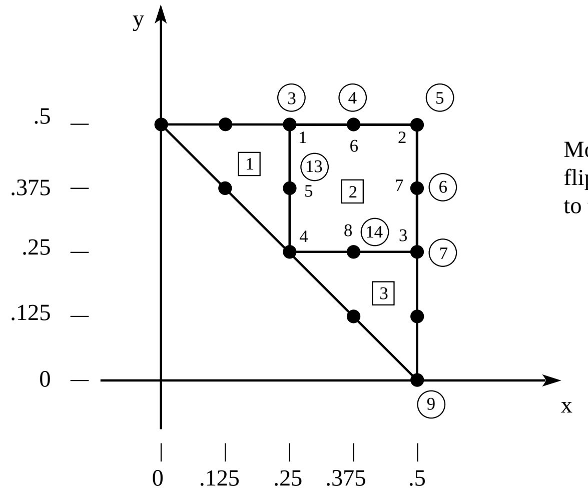
Here are plots of the stress function and one of the shearing functions for element 2.



$\phi[x, y]$ for element 2.



$\sigma_{zx}[x, y]$ for element 2.



Model has been flipped from bottom to top to align with plots.

Linear-Triangular Elements

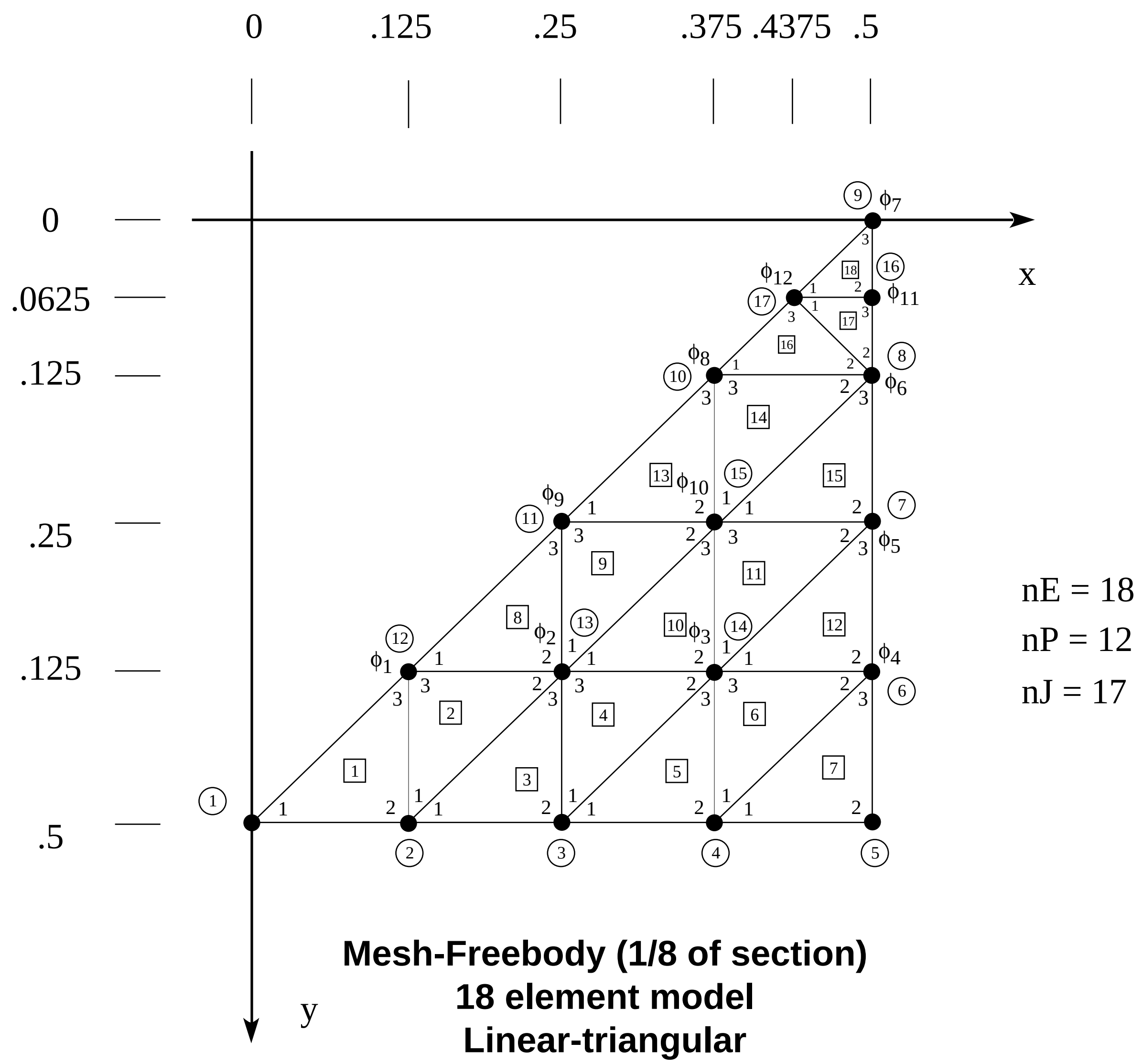
Using FETorsn with the TI-85 and TI-86

With the TI's that do not allow "unevaluated" expressions in matrices *FETorsn* is simplified. The changes are:

- Only the linear-triangular element is supported.
- The ShapMat matrix is replaced with shape functions, N1[x, y] to N3[x, y].
- The TorMat matrix contains only constants. (That is the gradients of the shape functions are constants.)
- Linear triangles are called "constant strain triangles," CST elements. The stresses are constant in the element, and those values are assumed to represent the value of the strains at the center of the element.
- G and θ are factored out of the Loads matrix. (They don't show up in the expressions. They have to be applied.)
- *FETorsn* may not work "on the fly" with GraphLink, (element matrices may be made one-at-a-time) and element matrices are not re-named.
- With the TI-85 variables have to be deleted with the TI's file manager.
- The program *MakeLMat* is not needed.
- With *AnalyTor*, G and α are values to be input along with a, b, n, x and y.

Here is the example problem worked with a linear-triangular mesh that gives good results for the problem.

Here's the model:



Mesh-Freebody (1/8 of section)
18 element model
Linear-triangular
Torsion

The matrices from *FEMInput* are:

```
Joints =  [[  0  .5  0 ]
           [  .125  .5  0 ]
           [  .25  .5  0 ]
           [  .375  .5  0 ]
           [  .5  .5  0 ]
           [  .5  .375  4 ]
           [  .5  .25  5 ]
           [  .5  .125  6 ]
           [  .5  0  7 ]
           [  .375  .125  8 ]
           [  .25  .25  9 ]
           [  .125  .375  1 ]
           [  .25  .375  2 ]
           [  .375  .375  3 ]
           [  .375  .25  10 ]
           [  .5  .0625  11 ]
           [  .4375  .0625  12  ]]
```

```
NodeVecs =  [[  1  2  12  ]
             [  2  13  12  ]
             [  2  3  13  ]
             [  3  14  13  ]
             [  3  4  14  ]
             [  4  6  14  ]
             [  4  5  6  ]
             [  12  13  11  ]
             [  13  15  11  ]
             [  13  14  15  ]
             [  14  7  15  ]
             [  14  6  7   ]
             [  11  15  10  ]
             [  15  8  10   ]
             [  15  7  8  ]
             [  10  8  17   ]
             [  17  8  16  ]
             [  17  16  9   ]]
```

The global matrices are:

```
GTorMat =  [[  2 –1  0  0  0  0  0  0  0  0  0  0 ]
            [ –1  4 –1  0  0  0  0  0 –1  0  0  0 ]
            [  0 –1  4 –1  0  0  0  0  0 –1  0  0 ]
            [  0  0 –1  2 –.5  0  0  0  0  0  0  0 ]
            [  0  0  0 –.5  2 –.5  0  0  0 –1  0  0 ]
            [  0  0  0  0 –.5  2  0 –.5  0  0 –.5 –.5 ]
            [  0  0  0  0  0  0  .5  0  0  0 –.5  0 ]
            [  0  0  0  0  0 –.5  0  2  0 –1  0 –.5 ]
            [  0 –1  0  0  0  0  0  0  2 –1  0  0 ]
            [  0  0 –1  0 –1  0  0 –1 –1  4  0  0 ]
            [  0  0  0  0  0 –.5 –.5  0  0  0  2 –1 ]
            [  0  0  0  0  0 –.5  0 –.5  0  0 –1  2  ]]
```

Loads = [[.015625]
[.03125]
[.03125]
[.015625]
[.015625]
[.014322916667]
[.001302083333]
[.013020833333]
[.015625]
[.03125]
[.002604166667]
[.005208333333]]

$\{\phi\} = (G\theta) * \text{GTorMat}^{-1} * \text{Loads}$

= 49.87
77.93
92.58
97.25
169.20
196.80
206.37
184.12
125.64
151.54
202.74
200.24

$\phi_{max} = 206.37$
($G=8\text{E}6$ and $\theta=.0001745$)

For Element 18, the gradient matrix = TorMat_{18} =

[[–16 0]
[16 16]
[0 –16]]

And the stresses for the center of element 18 are,

$\text{TorMat}_{18}{}^{T} * \{\phi\}_{18}$ = [[40][-58.08]]

Where $\{\phi\}_{18}$ =
[[200.24]
[202.74]
[206.37]]

Therefore, $\sigma_{zx} = -58.08$ and $\sigma_{zy} = -40$

Compare with *AnalyTor* at (x, y) = (23/48, 1/24):

$\sigma_{zx} = -58.203$ and $\sigma_{zy} = -28.88$

TorMat matrices do not contain terms for zero values. The TorMat for element one for example would have one row and its {ϕ} matrix is [[49.87]]. (Don't include the two zero values.)

For the cross section, T =

$(2)\{(1/384)\ [(3)(\phi_1) + (6)(\phi_2) + (6)(\phi_3) + (3)(\phi_4) + (3)(\phi_5) + (2)(\phi_6) + (2)(\phi_8) +$
$(6)(\phi_{10}) + (3)(\phi_4)] +$
$(1/768)[\phi_8 + \phi_6 + \phi_{12}] +$
$(1/1536)[\phi_6 + (2)(\phi_{11}) + (2)(\phi_{12}) + \phi_7]\}$

= (2) [10.398489 + .75933229 + .786868] = 23.88936

(See “6. Twisting Moments”)

Our model was 1/8 of the cross section so T = 8 * 23.88936 = 191.1 (N•cm)

Compare with *AnalyTor*, T= 196.25 (N•cm)

The error is 2.6%.

Using FEMInput

Using *FEMInput* is simply a programmatic way of getting information from a free-body diagram into matrices. It is not a mesh generator, and because of its simplicity one should check the matrices themselves before they are used, for reasons below.

FEMInput precesses about a free body. First it asks for coordinates of joints. It expects an x-y pair for each joint, a TI *vector*. So the form of input for coordinates is [x, y]. *FEMInput* lets you start from joint i of n. But you cannot skip completely the input of joints. At the least, you would have to rebuild joint n.

Next the program asks for “possibles,” the degrees of freedom that are consecutively numbered from 1 to n. It does not matter what order the possibles are entered, but “0’s” are reserved for degrees of freedom that are restrained (these are default values) and the numbering from 1 to n must be consecutive and positive. Possibles are *integers* greater than zero.

For example, with a torsion-of-a-square-cross-section problem, where the model is a 4-element mesh of bilinear rectangles, there are 9 joints. At each joint there is one degree of freedom, ϕ. But at the boundaries $\phi = 0$. That is there are restrained degrees of freedom at the boundaries. That leaves only one possible. That possible would be numbered 1. That means we would have only to enter one possible because they are all zero by default.

This default numbering can be a problem if possibles are being re-entered to correct a mistake. If a possible that should be zero is given some other number then one would have to choose to input that possible again and enter a zero for its value. In other words, choosing not to enter a possible does not make that possible zero. (They are all zero by default, but once you make one none zero, it can only be made zero again by choosing to enter a zero value. Or of course the matrix editor can be used to make any changes whatsoever.) Like with the input of joints, there is an option to start at joint i of n inputs of possibles. And the least joints one can re-input will be to start with joint n. (Possibles are associated with joints in that they are coincident.)

The third and final loop is the input of node vectors. Node vectors are TI *vectors* that are the mapping of element nodes onto global nodes, onto the joints. *FEMInput* is set up to work with elements that have up to 8 nodes. For clarity, the sizes of node vectors for the different elements (used by *FETorsn* and its cousins for plane stress and for plates) are:

Linear triangle	1x3
Bilinear rectangle	1x4
Quadratic triangle	1x6
Quadratic rectangle	1x8

Again, it's possible to skip the input of a number of the vectors, just as with the two previous loops.

FEMInput has a selector to choose from "Torsion, Plane stress and Plates." With torsion, there is one possible, ϕ. With plane stress here there are two possibles, translation in the x and y directions. With plates here there are three possibles, translation in z and rotation about x and y.

The symbols used by *FEMInput* for those possibles are:

Torsion	nPϕ
Plane stress	nPx, nPy
Plate	nPw, nPθx, nPθy

FEMInput first asks for the numbers of elements, possibles and joints. Those queries are answered with integers. Next a "mode" is chosen from a menu.

When *FEMInput* asks questions for which there are not answers in the form of some integer or some vector, then answer with a boolean-style reply. Use "1" for "yes," and use "0" (zero) for "no." To interrupt *FEMInput* press "ON." The TI's are adept at recovering from some "input" errors so don't hesitate to press the "GOTO" button when an error pops up.

Element Compatibility

The linear triangle and the bilinear rectangle work together. The quadratic triangle and and quadratic rectangle work together. If one will map these out together, he or she will see that they fit together at their nodes. If linear elements are mixed with quadratic elements, then there will be singular nodes, in which case results would be no good.

In some cases one might be able to mix linear and quadratic elements, depending on the shape of the cross section or by varying the sizes of elements.

Increased Accuracy

Accuracy is increased by using more and higher order elements. Using many linear triangles is going to work very well when they become so small that constant-strain-per-element is a realistic assumption.

Trouble Shooting

A singular GTorMat matrix means at least one row of it is either all zeros or is a linear copy of another row. This means an impossible free-body or an error in the "Joints" or "NodeVecs" matrices. Check the Joints matrix first for the possibles. They begin in column 3. Next check the NodeVecs matrix against the free-body.

Finally, check the coordinates, columns 1 and 2 of “Joints.” If no error is found rework the free-body.

If *FETorsn* “blows up” at “building shape matrix,” then there is a mistake with the coordinates. Zienkiewicz shows that the “ConsMat” matrix, the matrix being built at this point, is never singular. That is, unless there is a mistake: two or more nodes with the same location, for example.

If *FETorsn* “blows up,” then the mistake is found, you may be able to skip the evaluation of the elements up to the point where the problem occurred. If the blow up occurs before the element matrices are “assembled,” then the global matrices do not contain terms from that element. So one could choose to *not* make new global matrices and to start evaluation from the element which was the problematic. FETorsn displays its operations so that it’s possible to know whether an element was assembled.

If the $\{\phi\}$ values don’t look right (symmetry?), or if they are way off from other methods, check the numbering of the element nodes. Reversing the order of the nodes from the required element-node number scheme would cause problems with the “Loads” and “ShapMat” matrices.

When quadratic triangles are used, the integration limits can be a source of error. The integration should bring a positive value, the area, no matter where the element is located. And the integration limits are element location specific, a single set of limits will not work for a family of same-sized elements.

Piece-wise integration limits are not a problem. See “4. Limits Lists for Quadratic Triangles.”

For a good introduction to FEM see first Boresi, then see Segerlind for torsion.

[1] See: “Applied Finite Element Analysis”, L. J. Segerlind (1984) and “Advanced Mechanics of Materials,” Boresi et al (1993).

[2] “Elementary Differential Equations with Boundary Value Problems,” C. H. Edwards and D. E. Penney (1993).

[3] *Id.*

[4] “Numerical Analysis for Applied Mathematics, Science, and Engineering,” D. Greenspan and V. Casulli (1988).

[5] “The Finite Element Method,” O. C. Zienkiewicz and R. L. Taylor (1988).