ME 548 Project Report

Mitchell Haglund, Ahmed Khalil

04/26/2021

Strategy

Before designing our truss, we conducted a finite element analysis (FEA) study on a solid body with the given bridge geometry to get a sense of where stresses are distributed throughout the structure. We initially found that there is a significant amount of stress being built up at the fixed locations of the 3D modeled bridge, making the entire bridge have the same gradient. In order to generate a plot which accurately translates the 3D model into the desired 2D model, we had to put a limit on the maximum stresses shown by SolidWorks. Figure 1 below shows how the stress gradient plot for this bridge geometry. Since the stresses have been capped, this is a relatively accurate model for how stresses are distributed throughout the model, with red showing high stress and blue showing low stress.

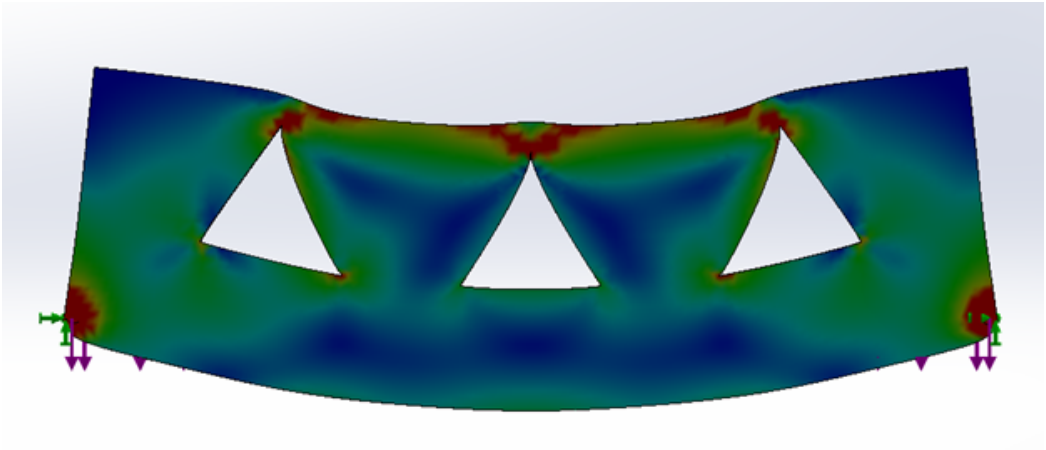

Figure 1. FEA simulation displaying stress distribution across bridge's geometry

As shown by the red and green contours in Figure 1, the areas of highest stress are around the triangle edge's, the top and bottom of the bridge, and the fixed locations. These are the areas that were most impacted by the design, and the truss designs detailed in this report largely reflect these findings. Our first iteration of the truss design prior to optimization is shown in Figure 2.

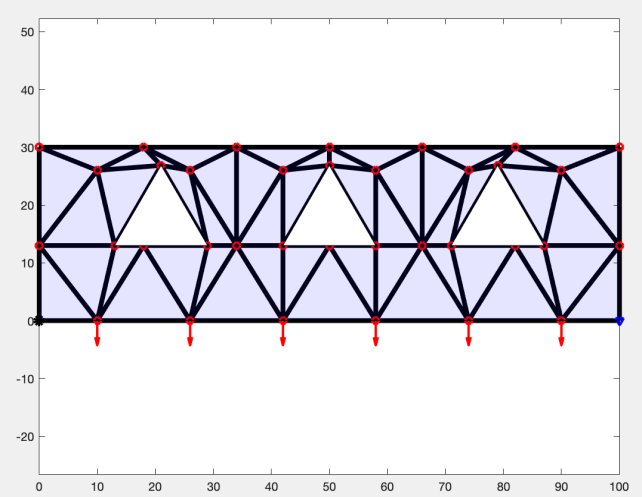

Figure 2. First Iteration of Truss Design Prior to Optimization

Figure 2 shows our first design of the truss before optimization and how we incorporated our findings from the FEA simulations to it. The first design targeted the top section of the bridge as that is where there is a large build of stresses, but did not take into account the length constraint as we only wanted to get an idea of how to start without over-complicating the design. Once we ran the optimization code with this design, we kept iterating the design to further minimize compliance and to meet the length constraint. A summary of our results is shown in the Results and Conclusion section of this report.

To accelerate the design process, a Delauney triangulation built-in MATLAB class was used to create an approximate mesh given node locations. A function called 'extractConnectivity' was written to translate between triangular and simple two-point connectivities. While this function is not called specifically in the final 'trussProject' code, it was a valuable tool throughout the design process, and is thus included in the Appendix for reference. Additionally, Excel spreadsheets were used to quickly modify node locations and receive instant visual feedback when doing so.

Formulation

The present optimization study attempts to minimize the compliance of the truss systems shown in the Results and Conclusions section. Equation 1 poses this problem with the appropriate constraints.

$$\begin{aligned} &\sum A_i l_i - V_{max} \leq 0 \\ &\sigma_i - \sigma_y \leq 0 \\ &-\sigma_i - \sigma_y \leq 0 \\ &\text{subject to} \\ &Ku = f \end{aligned} \tag{1}$$

As shown in Equation 1, each bar is constrained by the 100 MPa yield stress in tension and compression, while the entire system is constrained to a total volume that cannot be exceeded (V_{max}). The entire truss must not exceed 3060 kg, though we found it more straightforward to constrain the volume to that which would yield the maximum mass, then simply apply the material density following the optimization. Minimizing the mass directly would yield equivalent results. The objective is the compliance of the entire system and is the product of the force, *f*, and the displacement at all nodes, *u*, on the truss. The first constraint is to ensure that the total volume of the truss bars do not exceed the maximum volume. That constraint is a function of the cross-sectional area, A_i, and length, l_i, of each bar. The second and third constraints ensure that the bars do not exceed yield strength in tension and compression, respectively. The stress at each bar is denoted by σ_i and the yield stress is denoted by σ_y.

Equation 1 is then scaled appropriately to ensure that the optimization does not terminate prematurely. Given that the areas are relatively small, almost the same size as the termination size for fmincon, we divided all areas by the initial area to get a reasonably accurate result. The same methodology was applied to the objective and other constraints, as shown in Equation 2.

$$\begin{aligned} &\frac{\sum x_i A_i l_i}{V_{max}} - 1 \leq 0 \\ &x_i > 0 \\ &\frac{\sigma_i}{\sigma_y} - 1 \leq 0 \\ &-\frac{\sigma_i}{\sigma_y} - 1 \leq 0 \\ &\text{subject to} \\ &Ku = f \end{aligned} \tag{2}$$

Here, J^0 is simply the initial compliance prior to optimizing. Equation 3 shows the non-dimensional variable x_i that scales the cross-sectional area of each member (A_i) relative to its initial area (A_i^0).

$$x_i = \frac{A_i}{A_i^0} \tag{3}$$

The dimensionless *x* variables are the design variables for this optimization, as the beam areas are the only variables modified to minimize the compliance.

In addition to the constraints listed above, the problem was also constrained by the length, *l,* of each bar such that they shall not exceed 10 m. Given that the coordinates of each node were predetermined, there was no method for the optimization code to vary the coordinates of each node such that they meet the necessary length constraint. However, the code was modified so it would display

an error if any of the bar lengths was greater than the desired value, notifying the designer to pick closer nodes during the next run.

Results and Conclusions

The optimization code written for this truss design is called 'truss2dMinComplianceAllConstraints' and utilizes all of the equations listed later in the formulation section of the report (see Appendix). In addition to these equations, the code written also plots each bar in the truss with line weights relative to the member with the largest area. This functionality helped us identify bars with relatively high areas and make the appropriate adjustments. All optimizations were carried out to at least 50,000 function calls.

Figure 3 shows the first iteration of the truss design, shown in Figure 2, after it has been optimized.

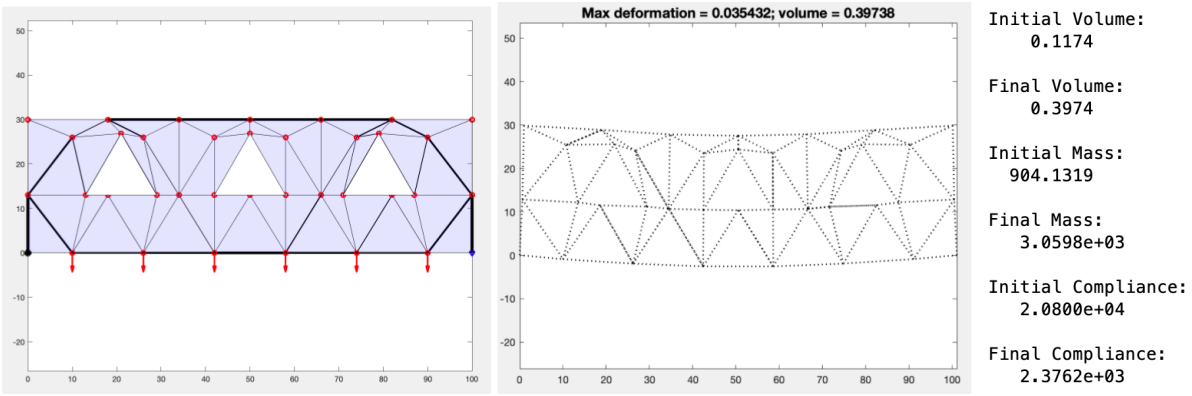

Figure 3. First Iteration of Truss Design After Optimization

This result verifies that the optimization code is working correctly as it meets all of the constraints defined by the optimization class. The final mass is maximized to the allowed value, the compliance is reduced by a factor of 10, and none of the bars exceed the yield strength. The only constraint not met by this design is that many members exceed 10 m in length. Much greater node density in later designs satisfies this constraint. Figures 4 and 5 show the second iteration of the truss design before and after optimization, respectively.

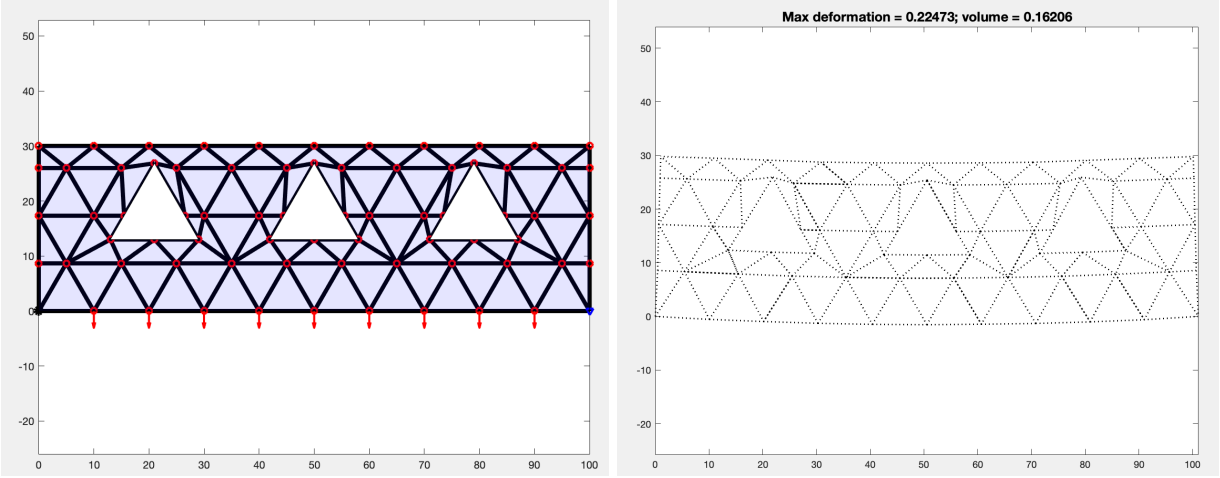

Figure 4. Second Iteration of Truss Design Prior to Optimization

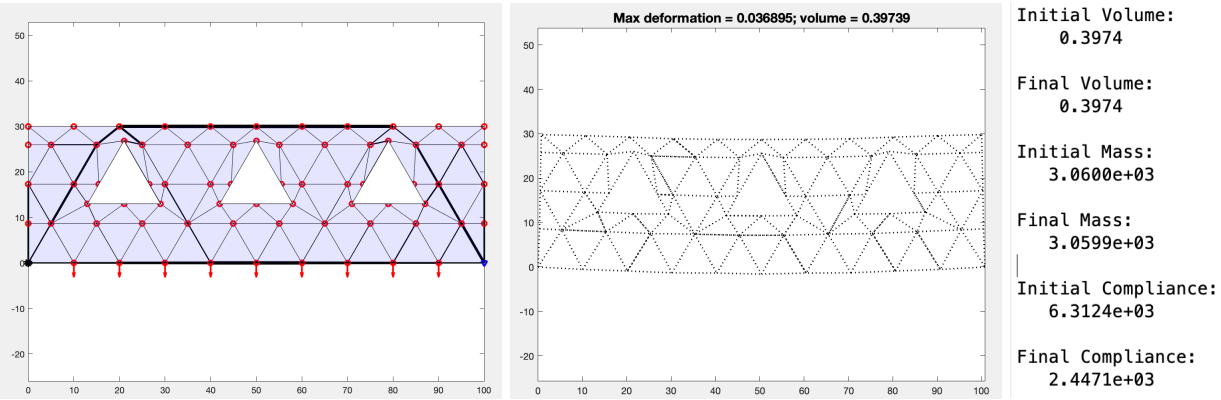

Figure 5. Second Iteration of Truss Design After Optimization

The second design iteration more closely follows the stress heatmap shown in Figure 1, where there is a higher node density particularly around the triangle cutouts. Interestingly, the second iteration actually performed slightly worse than the first. However, given that the first iteration did not fully meet the design constraints, the second iteration is still an improvement given that all constraints are met without a significant decrease in performance.

For our final iteration, we targeted the first layer from the bottom, the area surrounding the triangles, and the edges of the bridge. The y-values of the first layer were reduced so the angle between the horizontal and the bars is smaller, allowing these members to carry more horizontal force and reduce the force carried by the final bottom layer. Similarly, for the area surrounding the triangles and the edges of the bridge, creating a smaller mesh of bars allows for greater optimization flexibility at these critical locations. The edges were also modified such that more bars are angled and can therefore distribute the vertical force that the edges experience. Figures 6 and 7 show the third and final iteration of the truss design before and after optimization, respectively.

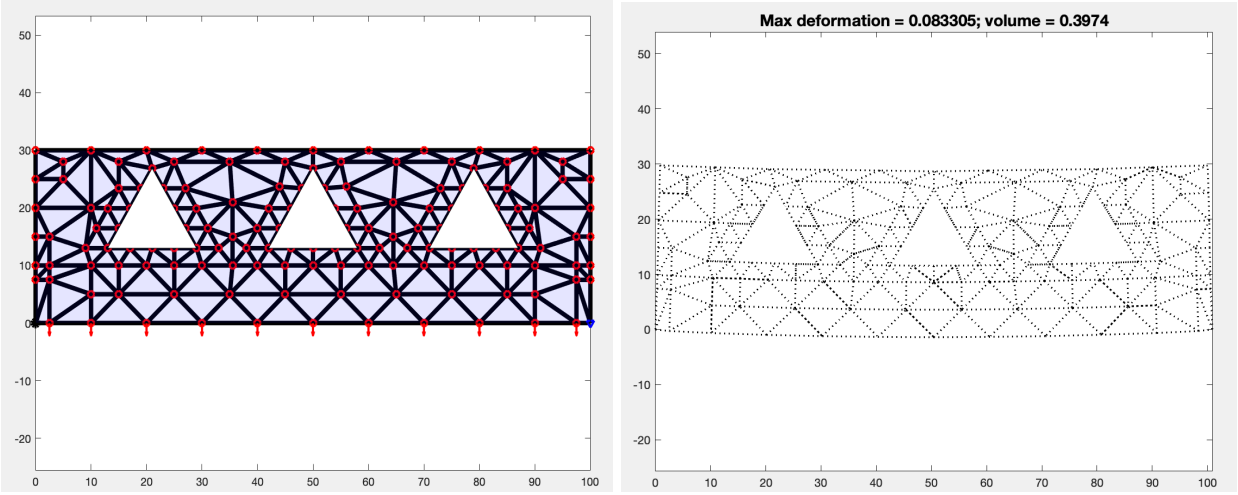

Figure 6. Final Iteration of Truss Design Prior to Optimization

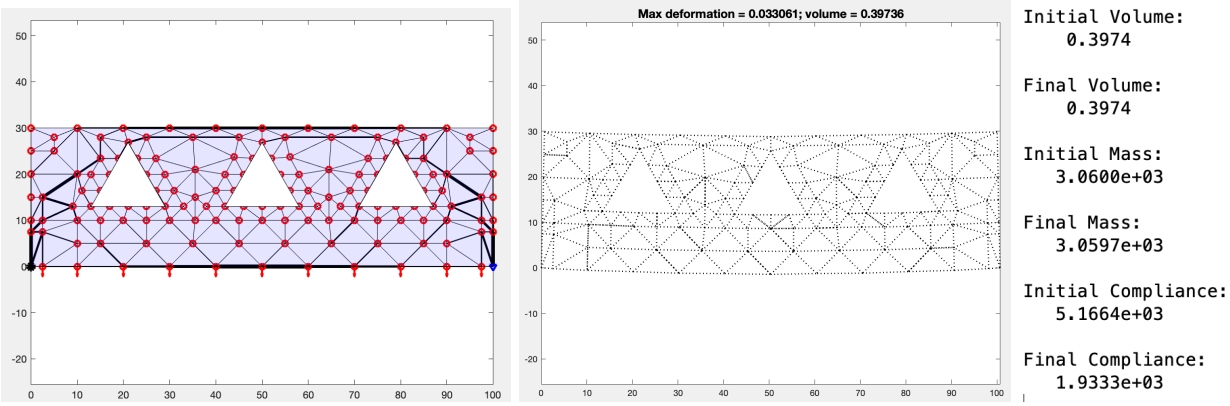

Figure 7. Final Iteration of Truss Design After Optimization

Figure 7 verifies our reasoning and reflects the conclusions from the FEA results. The code generated results where the final mass is maximized, the final compliance is reduced to just 1,933 N-m, and the maximum deformation is 0.033 m. However, after comparing our first iteration and final iteration results, it was concluded that the number of nodes and bars does not significantly decrease the final compliance.

We do not believe that there is an attainable optimal solution that could be found manually simply due to the quantity of variables at play and the time required for each iteration. Another potential method of solving this problem would be to build a neural network that would essentially be iterating as we are doing, however the neural network could compute and find trends much faster than we could by hand. This method is significantly more efficient in terms of time and would also generate a more optimal solution. However, given the time constraint of this project, we were not able to build a neural network or any other machine learning model in the allocated time.

Appendix A: testProject.m

NOTE: This code takes a long time to run (5+ minutes)

Consider lowering the maximum function calls in truss2dMinComplianceAllConstraints for testing

Contents

NOTE: extractConnectivity attached at the end, for reference

```
load("xy2_new.mat");
load("C2_new.mat");
```

```
clc;
trussClass = 'truss2dMinComplianceAllConstraints';
t = feval(trussClass,xy2_new,C2_new);% initialize model
t = t.fixXofNodes([1]);
t = t.fixYofNodes([1 2]);
forceNodes = [140,14,15,16,17,18,19,20,21,22,141];
totalForce = -1e5;
t = t.applyForce(forceNodes,[0; totalForce/numel(forceNodes)]);
t = t.assignE(2e11); % for all members
t = t.assignA(8.5e-05); % for all members
t = t.assignDensity(7700); % for all members
maxMass = 3060; % maximum mass in kg
t = t.assignMaxVolume(maxMass/t.myDensity); % maximum volume

t = t.assemble();
t = t.solve();
t = t.computeStresses();
t = t.optimize();

t.plot(0);
projectDesignSpace;
figure;
t.plotDeformed();
if (max(t.myL) > 10)
    disp('WARNING: MEMBERS DO NOT MEET LENGTH CONSTRAINT');
end
disp('Initial Volume: '); disp(t.myInitialVolume)
disp('Final Volume: '); disp(t.myFinalVolume);
disp('Initial Mass: '); disp(t.myInitialMass)
disp('Final Mass: '); disp(t.myFinalMass);
disp('Initial Compliance: '); disp(t.myInitialCompliance)
disp('Final Compliance: '); disp(t.myFinalCompliance);
```

Appendix B: truss2dMinComplianceAllConstraints.m

```
classdef truss2dMinComplianceAllConstraints < truss2d
    % stress-constrained volume minimization
    properties(GetAccess = 'public', SetAccess = 'private')
        myInitialVolume;
        myInitialArea;
        myInitialCompliance;
        myFinalArea;
        myFinalCompliance;
        myFinalVolume;
        myYieldStress;
        myLambda;
        myDensity;
        myInitialMass;
        myFinalMass;
        myMaxVolume;
    end
    methods
        function obj = truss2dMinComplianceAllConstraints(xy,connectivity)
            obj = obj@truss2d(xy,connectivity);
            obj.myYieldStress(1:obj.myNumTrussBars) = 100e6; % default
        end
        function obj = assignYieldStress(obj,yieldStress,members)
            % assign sMax to one or more members
            % if members is not given, then assign to all
            if (nargin == 2)
                members = 1:obj.myNumTrussBars;
            else
                assert(max(members) <= obj.myNumTrussBars);
                assert(min(members) >=  1);
            end
            obj.myYieldStress(members) = yieldStress;
        end
        function obj = assignMaxVolume(obj,maxVolume)
            obj.myMaxVolume = maxVolume;
        end
        function obj = assignDensity(obj,density)
            obj.myDensity = density;
        end
        function JRelative = complianceObjective(obj,x)
            Area = x.*obj.myInitialArea;
            obj = obj.assignA(Area);
            obj = obj.assemble();
            obj = obj.solve();
            J = obj.getCompliance();
            JRelative = J/obj.myInitialCompliance;
        end
        function [cineq,ceq] = allConstraints(obj,x)
            Area = x.*obj.myInitialArea;
            obj = obj.assignA(Area);
            obj = obj.assemble();
            obj = obj.solve();
            nConstraints = 2*obj.myNumTrussBars + 1; % two stress constraints per bar and volume constraint
            cineq = zeros(1,nConstraints);
            constraint = 1;
            for m = 1:obj.myNumTrussBars
                cineq(constraint) = obj.myStress(m)/obj.myYieldStress(m)-1;% tension
                cineq(constraint+1) = -obj.myStress(m)/obj.myYieldStress(m) -1;%compression
                constraint = constraint+2; % increment
            end
            cineq(constraint) = sum(obj.myArea.*obj.myL)/obj.myMaxVolume - 1; % cannot exceed max volume
            ceq = [];
        end
        function obj = initialize(obj)
            obj.myInitialArea = obj.myArea;
            obj.myInitialVolume = sum(obj.myArea.*obj.myL);
            obj.myInitialMass = obj.myInitialVolume*obj.myDensity;
            obj = obj.assemble();
            obj = obj.solve();
            obj.myInitialCompliance =  obj.getCompliance();
        end
        function processLambda(obj)
            ineqnonlin = obj.myLambda.ineqnonlin;
```

```
            maxValue = max(abs(ineqnonlin));
            ineqnonlin = ineqnonlin/maxValue; % scaled
            for m = 1:obj.myNumTrussBars
                if (ineqnonlin(2*m-1) > 0.0001)
                    disp(['Bar ' num2str(m) ': Tensile stress active']);
                elseif (ineqnonlin(2*m) > 0.0001)
                    disp(['Bar ' num2str(m) ': Compressive stress active']);
                else
                    disp(['Bar ' num2str(m) ': Stress inactive']);
                end
            end
        end
        function obj = optimize(obj)
            obj = obj.initialize();
            opt = optimset('fmincon');
            opt.MaxFunEvals = 50000; %NOTE: 50000 function calls will take 5+ minutes to run; consider lowering for test
            x0 = ones(1,obj.myNumTrussBars); % unitless quantities
            LB = 1e-12*ones(1,obj.myNumTrussBars); % small non-zero values
            [xMin,~,~,~,Lambda]  = fmincon(@obj.complianceObjective,x0, ...
                   [],[],[],[],LB,[],@obj.allConstraints,opt);
            obj = obj.assignA(xMin.*obj.myInitialArea);
            obj = obj.assemble();
            obj = obj.solve();
            obj.myFinalArea= obj.myArea;
            obj.myFinalVolume = sum(obj.myArea.*obj.myL);
            obj.myFinalCompliance =  obj.getCompliance();
            obj.myLambda = Lambda;
            obj.myFinalMass = obj.myDensity * obj.myFinalVolume;
        end
    end
end
```

Published with MATLAB® R2020b

Appendix C: extractConnectivity.m

```
function C = extractConnectivity(xy)
    DT = delaunayTriangulation(xy); % initialize triangulation
    connectivityTriangles = DT.ConnectivityList; % connectivity of each triangle
    [rows,cols] = size(connectivityTriangles);
    connectivity = [];
    for (i = 1:rows)
        connectivity = [connectivity;
                        connectivityTriangles(i,1) connectivityTriangles(i,2);
                        connectivityTriangles(i,3) connectivityTriangles(i,2)];
    end
    C = connectivity;
end
```

Published with MATLAB® R2020b