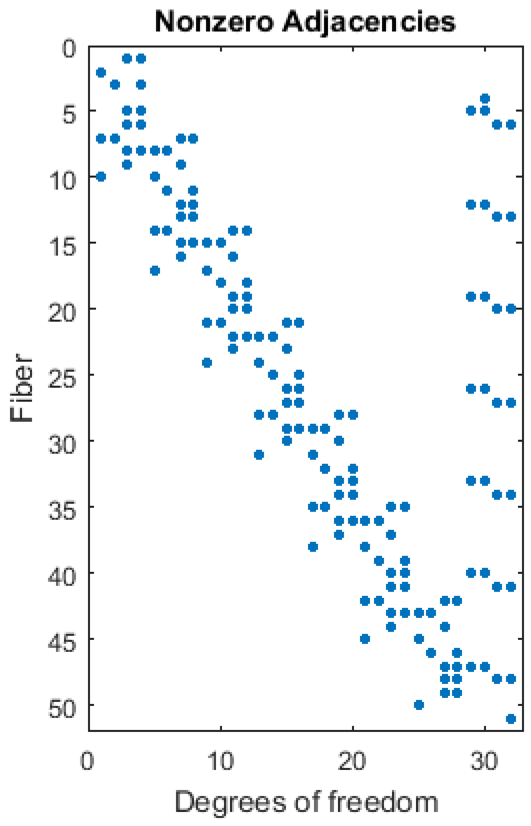
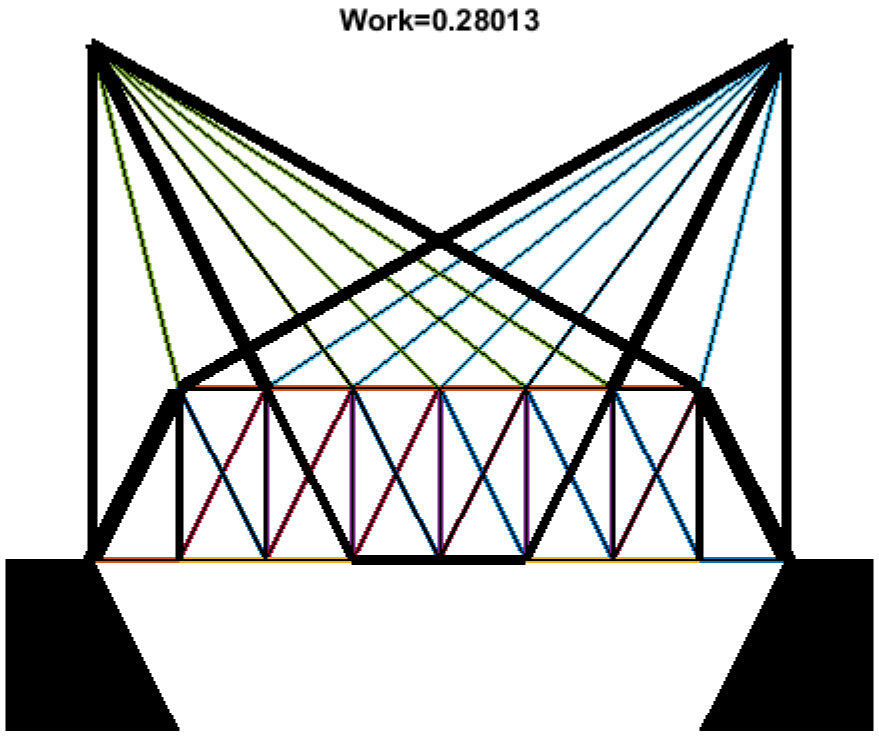
```
%Minjun Park, CAAM 210, 11/08/15
%
%This function represents optimal design of bridges, using built-in
 fmincon function
%basically a function which displays how a bridge changes in terms of
%thickness
%To use type in swallowmyload with a module #, stiffness val, & force
%Example: bridge2(6, 1, -.05)
function bridge2(r, Eaval, force)
%this function takes in 3 inputs,
%r: the number of modules the bridge has
%Eaval: the value corresponding to the "strength" of the fibers
%force: the force that will be applied to each succesive node
%it outputs 2 figures, the first being a visual representation of the
%adjacency matrix which delineates the geometry of the bridge and also
%a visual representation of the bridge itself under stressed and
%unstressed conditions also gaussian elimination is coded in
xc=[0 1; 0 1; 1 1; 0 0; 0 1; 1 r+2];    %initializes current x
 positions
yc=[0 1; 0 0; 0 1; 0 3; 3 1; 1 3];      %y positions and also
 specifies
s=1/sqrt(2);                            %s which is just a variable
 for
fibernum=5+5*r+4+2*r;                   %adjacency values of the
nodenum=4+4*r+4;                        %crossbars of the bridge
leftnodex=nodenum-3;                    %also specifies total number
leftnodey=nodenum-2;                    %of fibers and node
 coordinates
rightnodex=nodenum-1;                   %along with the two upper
 nodes'
rightnodey=nodenum;                     %coordinates, and finally it
A = zeros(fibernum,nodenum);            %initializes the A matrix
A(1, 3)=s;                              %and L vector
A(1, 4)=s;
A(2, 1)=1;
A(3, 4)=1;
A(3, 2)=-1;
A(4, leftnodey)=1;
A(5, 3)=1/sqrt(5);
A(5, 4)=-2/sqrt(5);
A(5, leftnodex)=-1/sqrt(5);
A(5, leftnodey)=2/sqrt(5);
A(6, 3)=-(r+1)/sqrt((r+1)^2+4);
A(6, 4)=-2/sqrt((r+1)^2+4);
A(6, rightnodex)=(r+1)/sqrt((r+1)^2+4);
A(6, rightnodey)=2/sqrt((r+1)^2+4);
L=[1/s 1 1 3 sqrt(5) sqrt((r+1)^2+4)];
for i=1:r
    node1x=2*(2*i-1)-1;         %the for loop is used to find the
    node2x=2*(2*i)-1;           %adjacency, length and current x and
    node3x=2*(2*i+1)-1;         %y position vals of all the fibers
    node4x=2*(2*i+2)-1;         %by using a modular system where
    node1y=2*(2*i-1);           %each iteration of the for loop
    node2y=2*(2*i);             %will get all these values for all
    node3y=2*(2*i+1);           %the fibers in one particular module
    node4y=2*(2*i+2);           %until the values for the the entire
    fiber7=7*i;                 %bridge have been calculated
    fiber8=7*i+1;
    fiber9=7*i+2;
    fiber10=7*i+3;
    fiber11=7*i+4;
    fiber12=7*i+5;
    fiber13=7*i+6;
    A(fiber7, node1x)=-s;
    A(fiber7, node1y)=-s;
    A(fiber7, node4x)=s;
    A(fiber7, node4y)=s;
    A(fiber8, node2x)=-s;
    A(fiber8, node2y)=s;
    A(fiber8, node3x)=s;
    A(fiber8, node3y)=-s;
    A(fiber9, node2x)=-1;
    A(fiber9, node4x)=1;
    A(fiber10, node1x)=-1;
    A(fiber10, node3x)=1;
    A(fiber11, node4y)=1;
    A(fiber11, node3y)=-1;
    A(fiber12, leftnodex)=-(i+1)/sqrt((i+1)^2+4);
    A(fiber12, leftnodey)=2/sqrt((i+1)^2+4);
    A(fiber12, node4x)=(i+1)/sqrt((i+1)^2+4);
    A(fiber12, node4y)=-2/sqrt((i+1)^2+4);
    A(fiber13, rightnodex)=(-i+1+r)/sqrt((-i+1+r)^2+4);
    A(fiber13, rightnodey)=2/sqrt((-i+1+r)^2+4);
    A(fiber13, node4x)=-(-i+1+r)/sqrt((-i+1+r)^2+4);
    A(fiber13, node4y)=-2/sqrt((-i+1+r)^2+4);
    xc=[xc; i i+1; i i+1; i i+1; i i+1; i+1 i+1; 0 i+1; i+1 r+2];
    yc=[yc; 0 1; 1 0; 1 1; 0 0; 0 1; 3 1; 1 3];
    L=[L 1/s 1/s 1 1 1 sqrt((i+1)^2+4) sqrt((-i+1+r)^2+4)];
end
A(fibernum-2, nodenum-5)=-s;            %finishes up the last
A(fibernum-2, nodenum-4)=s;             %few fibers which the
A(fibernum-1, nodenum-7)=-1;            %for loop didn't
A(fibernum, rightnodey)=1;
xc=[xc; r+1 r+2; r+1 r+2; r+2 r+2];
yc=[yc; 1 0; 0 0; 0 3];
L=[L 1/s 1 3];
spy(A);
xlabel('Degrees of freedom');
ylabel('Fiber');
title('Nonzero Adjacencies');
Ea=ones(1, length(L));                  %sets up young's modulus stuff
Ea(Ea==1)=Eaval;                        %and stiffness matrix along
 with
k = Ea./L;                              %the S matrix which is the
K = diag(k);                            %k in F=kx
S = A'*K*A;
```

```
f=ones(1, length(S));
f(1:2:end)=0;
f(4:4:end)=0;
f(nodenum+1:end)=0;
f(end-3:end)=0;
f=f.*force;
x = gauss(S,f');                        %Calls on gauss to do the
xdisp=x(1:2:end);                       %gaussian elim to solve for x
ydisp=x(2:2:end);
%dx=xc;
%dy=yc;
%dx(1,2)=xc(1,2)+xdisp(2);               %uses the x values seperated
%dx(2,2)=xc(2,2)+xdisp(1);               %out into xdisp and ydisp to
%dx(3,1)=xc(3,1)+xdisp(1);               %calculate the dx and dy, or
%dx(3,2)=xc(3,2)+xdisp(2);               %final positions of all the
%dx(4,2)=xc(4,2)+xdisp(2*r+3);           %fibers, here the dx and
%dx(5,1)=xc(5,1)+xdisp(2*r+3);           %dy matrices are initialized
%dx(5,2)=xc(5,2)+xdisp(2);
%dx(6,1)=xc(6,1)+xdisp(2);
%dx(6,2)=xc(6,2)+xdisp(2*r+4);
%dy(1,2)=yc(1,2)+ydisp(2);
%dy(2,2)=yc(2,2)+ydisp(1);
%dy(3,1)=yc(3,1)+ydisp(1);
%dy(3,2)=yc(3,2)+ydisp(2);
%dy(4,2)=yc(4,2)+ydisp(2*r+3);
%dy(5,1)=yc(5,1)+ydisp(2*r+3);
%dy(5,2)=yc(5,2)+ydisp(2);
%dy(6,1)=yc(6,1)+ydisp(2);
%dy(6,2)=yc(6,2)+ydisp(2*r+4);
%for i=1:r
 %   node1=2*i-1;                     %for loop functions much
  %  node2=2*i;                       %in the same way as the last
   % node3=2*i+1;                     %by modularly getting dx
    %node4=2*i+2;                     %and dy vectors
    %leftnode=2*r+3;
    %rightnode=2*r+4;
    %fiber7=7*i;
    %fiber8=7*i+1;
    %fiber9=7*i+2;
    %fiber10=7*i+3;
    %fiber11=7*i+4;
    %fiber12=7*i+5;
    %fiber13=7*i+6;
    %dx(fiber7,1) = xc(fiber7,1)+xdisp(node1);
    %dx(fiber7,2) = xc(fiber7,2)+xdisp(node4);
    %dx(fiber8,1) = xc(fiber8,1)+xdisp(node2);
    %dx(fiber8,2) = xc(fiber8,2)+xdisp(node3);
    %dx(fiber9,1) = xc(fiber9,1)+xdisp(node2);
    %dx(fiber9,2) = xc(fiber9,2)+xdisp(node4);
    %dx(fiber10,1) = xc(fiber10,1)+xdisp(node1);
    %dx(fiber10,2) = xc(fiber10,2)+xdisp(node3);
    %dx(fiber11,1) = xc(fiber11,1)+xdisp(node3);
    %dx(fiber11,2) = xc(fiber11,2)+xdisp(node4);
    %dx(fiber12,1) = xc(fiber12,1)+xdisp(leftnode);
    %dx(fiber12,2) = xc(fiber12,2)+xdisp(node4);
    %dx(fiber13,1) = xc(fiber13,1)+xdisp(node4);
    %dx(fiber13,2) = xc(fiber13,2)+xdisp(rightnode);
    %dy(fiber7,1) = yc(fiber7,1)+ydisp(node1);
    %dy(fiber7,2) = yc(fiber7,2)+ydisp(node4);
    %dy(fiber8,1) = yc(fiber8,1)+ydisp(node2);
    %dy(fiber8,2) = yc(fiber8,2)+ydisp(node3);
    %dy(fiber9,1) = yc(fiber9,1)+ydisp(node2);
%    dy(fiber9,2) = yc(fiber9,2)+ydisp(node4);
%    dy(fiber10,1) = yc(fiber10,1)+ydisp(node1);
%    dy(fiber10,2) = yc(fiber10,2)+ydisp(node3);
%    dy(fiber11,1) = yc(fiber11,1)+ydisp(node3);
%    dy(fiber11,2) = yc(fiber11,2)+ydisp(node4);
%    dy(fiber12,1) = yc(fiber12,1)+ydisp(leftnode);
%    dy(fiber12,2) = yc(fiber12,2)+ydisp(node4);
%    dy(fiber13,1) = yc(fiber13,1)+ydisp(node4);
%    dy(fiber13,2) = yc(fiber13,2)+ydisp(rightnode);
%end
%dx(fibernum-2,1)=xc(fibernum-2,1)+xdisp(2*r+2);     %finishes up
%dx(fibernum-1,1)=xc(fibernum-1,1)+xdisp(2*r+1);     %last few fibers
%dx(fibernum,2)=xc(fibernum,2)+xdisp(2*r+4);         %dx and dy
%dy(fibernum-2,1)=yc(fibernum-2,1)+ydisp(2*r+2);
%dy(fibernum-1,1)=yc(fibernum-1,1)+ydisp(2*r+1);
%dy(fibernum,2)=yc(fibernum,2)+ydisp(2*r+4);

figure
hold on
line(xc',yc','linewidth',2);    %plots original bridge
%line(dx',dy','linewidth',2);    %plots deformed bridge
baseleftx=[0 1; -1 -1; 1 -1; -1 0];     %base for artsy aesthetics
baselefty=[0 -1; -1 0; -1 -1; 0 0];     %as well as work value
baserightx=[(nodenum-4)/4+1 (nodenum-4)/4; (nodenum-4)/4+2
 (nodenum-4)/4+2; (nodenum-4)/4 ...
    (nodenum-4)/4+2; (nodenum-4)/4+2 (nodenum-4)/4+1];
baserighty=[0 -1; -1 0; -1 -1; 0 0];
fill(baseleftx, baselefty, 'k', baserightx, baserighty, 'k')
Work=sum(f'.*x)
tit=strcat('Work=', num2str(Work))
title(tit)
axis off
V = sum(L); %adding all fiber lengths
a0 = ones(length(L),1); %vector (size #fibers x 1) of initial guesses
 of cross-sectional area
alow_fibvec=ones(length(L),1)*(.1); %lower bound
ahigh_fibvec=ones(length(L),1)*(10); % upper bound
options = optimset('MaxFunEvals', 20000);
[opts]=fmincon(@(a) work(a,A,L,f), a0, [], [], L, V, alow_fibvec,
 ahigh_fibvec, [], options);
%a = variable to be manipulated. Is a row vector of cross-sectional
 area of
%each fiber
%A = the adjacency matrix
%L = row vector of length of each fiber
%force vector
```

```
for i=1:size(opts)
    line(xc(i,:), yc(i,:), 'LineWidth', ...
        2*opts(i), 'Color', 'k');
end
%{this plots optimal design, with the color(black)}
end
function x=gauss(S,f)
n = size(S);                        %gets original size of S vector
S = [S f];                          %augments matrix S with f
for i=1:n
    if S(i,i)==0                    %if statement to swap out
        S([i n],:)=S([n i],:);      %problematic 'pivot' zeros
    end
    S(i,:)=S(i,:)/S(i,i);               %normalizes rows
    for j=i+1:n
        S(j,:)=S(j,:)-S(i,:)*S(j,i);%pivots kill everything below
    end
end
f=S(:,end);                         %gets new f vector
S(:,end)=[];                        %gets new S matrix
x=trisolve(S,f);                    %inputs S and f into trisolve
end
function x=trisolve(S,f)
n=length(f);                        %gets size of f vector
x=zeros(n,1);                       %plans x vector same size as f
x(n)=f(n)/S(n,n);                   %solves last case
for i=n-1:-1:1;
    x(i)=(f(i)-S(i,(i+1):n)*x((i+1):n))/S(i,i); %bubbles up
end
end
function [val, grad] = work(a,A,L,f)
K = diag(a./L'); %Definition of K
S = A'*K*A; %how to get this from E,a,L
x = gauss(S,f');                %using gauss function
val=f*x; %value of the work
grad= -(A*x).^2./L';%gradient formula
end
%this is a given protocol, which is later needs to be returned both
 the
%value of the work and its gradient


Work =

    0.2801


tit =

Work=0.28013


Local minimum found that satisfies the constraints.
```

```
Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the default value of the function
 tolerance,
and constraints are satisfied to within the default value of the
 constraint tolerance.
```





Published with MATLAB® R2015a