ASEN 2001: Lab 2 - Truss Design and Analysis

Quentin H. Morton*, Jasmin J. Chadha†, Heng Deng‡, and Ajay Dhindsa§¶

Two trusses were designed and built in order to understand how support reactions, external forces, and architecture influence static equilibrium. MATLAB programming was used to find the optimal truss design through factor of safety and probability of failure. The chosen truss was tested for weight capabilities and structural integrity in a design competition, in which it held a maximum mass of 315 grams and spanned 58.3 inches.

I. Introduction

A system is said to be in static equilibrium when the sum of all acting forces and torques is equal to zero. In this lab, the truss designed was defined to be in static equilibrium if the external reaction forces acting on either three or four reaction joints, and the total weight of the bars and magnets comprising the truss would sum to zero. The total weight of the truss was placed at the center of the truss for simpler calculations. Equilibrium equations had to be created in order to computationally solve the reaction forces to balance the truss and make it statically determinant. The equilibrium equations were solved using a system of linear equations, $Ax = b$. The A matrix in the linear equation holds the coefficients of all forces and moments acting on the joints at each axis, the x vector contains the magnitude of the internal reactions forces which are being solved for, and the b vector contains the values of all the external forces acting on each joint of the truss due to gravity. The x vector of the linear system is solved by finding the inverse of the A matrix and multiplying it by the b vector, $x = A^{-1}b$.

Once the x vector is solved and the magnitudes of the external forces on each bar and joint are known, a Monte-Carlo simulation can be done to analyze the truss. The Monte-Carlo simulation accounts for uncertainty in the within the truss with the varying bar lengths and magnet strengths. The maximum possible force is used to find the Factor of Safety (FOS) of the truss design. The FOS of a truss is measurement of how structurally sound something is. Once the desired FOS was obtained for both teams designs, the design which best fit the requirements of bridging the longest length and holding the most weight was chosen. Afterwards, the chosen truss design was built using the given materials and tested to determine the maximum external load which it could withstand.

*109209382
†109635890
‡107215041
§108423004
¶Group 16

II. Design Analysis

A. Concept Models

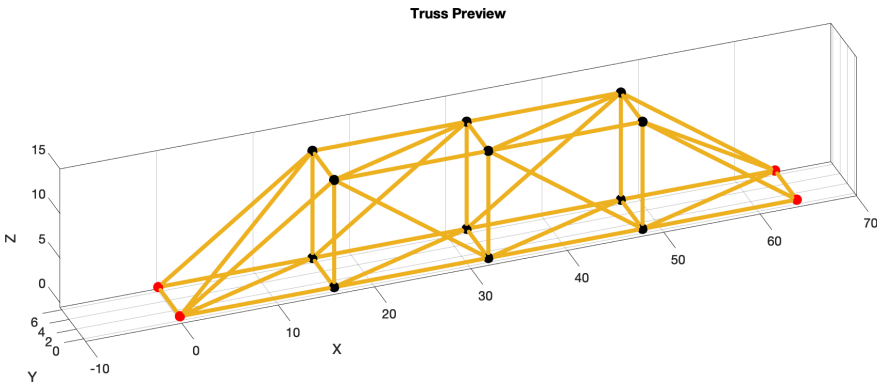

Fig. 1 Truss Design: Concept 1

Figure 1 shows the first truss, designed by subteam 1 design was 64 inches long and resembles many bridges with simple trapezoids and triangles. A simple design was created with sections on each end and the ability to add more internal sections to maximize length as materials allowed. The final design included two middle sections due to limitations on provided materials. This design included 16 joints.

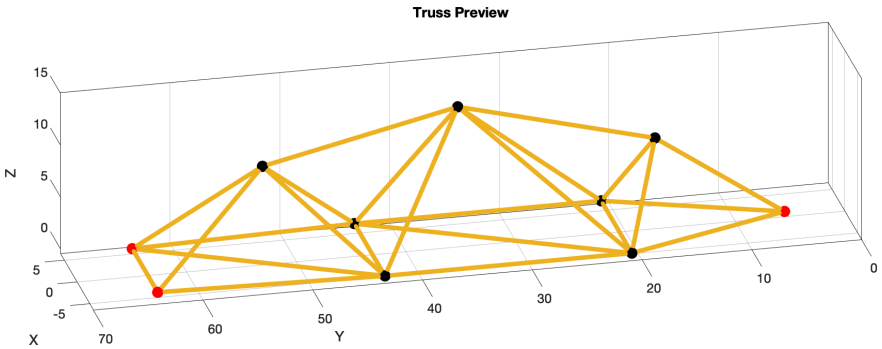

Fig. 2 Truss Design: Concept 2

Figure 2 shows the second truss, designed by subteam 2, was designed to be more stable than the first truss design. The goal in creating a more stable truss was to allow it to hold a larger external load than subteam 1's truss design, even though the first design was slightly longer. This truss design was the final design that was chosen to be built for the competition. This design was chosen due to the reliability of the design that it would definitely be able to stand under its own weight and would also be able to hold an external load.

B. Detailed Design and Computational Analysis

In order to create the chosen truss design, first a rough design was made on paper with estimates on all dimensions and lengths. Next, text input files were created with the number of bars, number of joints, and connection order of the bars themselves to be input into the MATLAB GUI provided. With these input files, a 3D model was able to be generated and from there, fine details were able to be made to the models to make them statically determinate.

In order to determine whether or not the truss would be able to withstand its own weight, computational analysis needed to be conducted. A function was created to find the amount of force which the bars and structure could withstand by determining the weight at the joints of the structure and summing these together. The weight of every ball magnet joint was taken and the weight to length ratio of the connecting rods was taken. Using the location of each ball magnet joint as a center, the connecting bars were stored as vectors and using the weight to length ratio, the weight of each bar

was placed at the center of itself and then added to the weight of the ball magnet joint at that location found previously. These resulting weight forces were all placed vertically downwards towards the structure to simulate the weight of the magnets and bars themselves.

C. Sensitivity Analysis

After creating a way to determine the weight of the truss itself, it was possible to run Monte-Carlo simulations on the truss to determine the maximum forces which the truss would undergo inside each bar. Using the maximum bar forces calculated using the analysis function written previously, a factor of safety and probability of failure for the truss was able to be calculated.

Using the known variations of joint strength, 4.8 N, the coefficient of variation of join strength being 0.8, and the coefficient of variation of join position being 1%, Monte-Carlo simulations were ran using 1,000,000 samples to obtain the most accurate results. The desired maximum chance of failure of the structure was chosen to be 5% during these simulation. Using the results from the Monte-Carlo Simulations, an inverse-cumulative distribution function was used to calculate the maximum allowable design forces which the truss could withstand, which is approximately 3.6 N. Using the maximum design force which the truss could withstand, the standard deviation and mean max design forces was calculated and with these, a factor of safety was calculated to be 1.12, which is within the original desired factor of safety margins of 1.1 - 1.8. Finally, the probability of failure was estimated to be 4.8% from the Monte-Carlo simulations.

III. Truss Design

A. Sub-team 1

Teams 1's truss' design had a final length of 64 inches. The design was comprised of 16 joints, 42 bars, 13 sleeves, and was a design with 4 reaction support points. This truss was designed to have both length and stability. The length of this truss was slightly longer than the truss which Team 2 designed by 5.7 inches; however, the bridge design was less stable and had a probability of failure of approximately 40%.

B. Sub-team 2

Team 2's truss is designed as 58.3 inches long. It has 10 joints, 24 bars and includes 6 sleeves. This truss focuses on the length; hence it is relatively weak on the stability and load capacity. We decided to use design with 3 reaction support points in order to make the cross section a triangle, which costs less beam and leads to a light self-weight.

IV. Model Validation

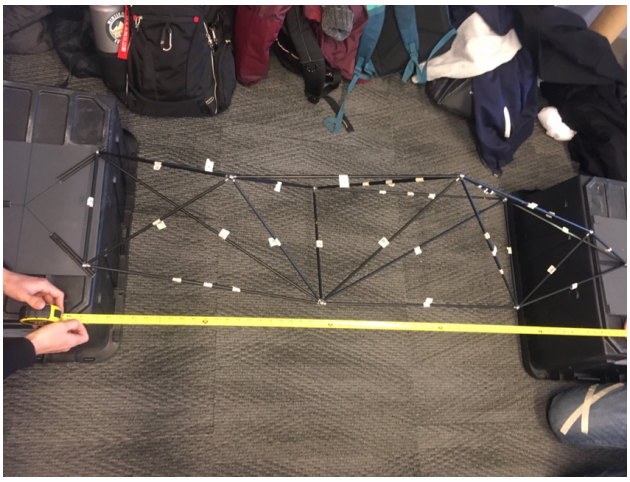

Fig. 3 Measurement of Length of Standing Truss

After constructing the truss, several team members tested the truss by setting it on supports in the PILOT Lab. The final length of the truss was 58 inches. Figure 3 shows the truss standing on its own and being measured. It easily supported its own weight, and the team members performed a loading test to find how large of an external load the truss could support before failure. The team members added magnets at the highest joint in the truss until the structure collapsed, and then weighed the magnets that were added to find the mass of the external load. Figure 4 shows the magnets being added to the joint. The truss supported 315 grams of weight before it collapsed.

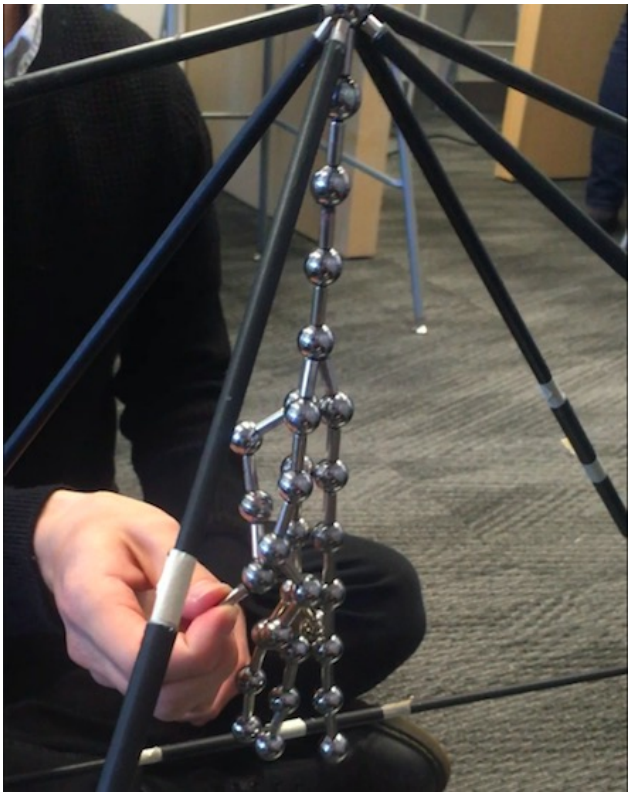

Fig. 4 Truss with Magnets Added to Test Maximum External Load

V. Discussion

Overall, it was difficult to achieve a design on MATLAB that would directly correlate to the design we built with the magnets and bars. The largest discrepancy between the theoretical truss design to the actual truss design was the bar lengths. Although we tried to compensate for values that weren't exactly the size of the bar (such as 25.5" instead of a 26" bar), it was difficult to measure out the magnets correctly and glue them in (i.e. if we wanted 0.5" added by the magnets in the above situation). This was mostly difficult because we couldn't account for slippage due to Elmer's glue if it didn't dry completely, or wasn't stronger than the pull of the ball joints. Not all forces were perfectly accounted due to interfering forces by magnets at joints. Some joints had many bars attached to them and therefore the polarity of all of them can create repulsive forces, making the joints slightly weaker. Our truss succeeded, mostly because we designed it carefully and based our construction off of specific measurements. We made sure to pick a design that had a low failure probability and high safety factor because we knew this meant it would have a higher chance of success. However, this meant we didn't take risks on having the longest truss possible, which would decrease the factor of safety. While the length was something we kept in mind, we decided that it was a trade off we were willing to accept in exchange of having a reliable, steady truss.

VI. Conclusion

This lab involved designing two different truss designs, performing sensitivity analysis in order to determine which truss to build, building this truss, and testing it with an external load. The selected truss successfully stood on supports and remained intact under its own weight. It also held an external load of 315 grams on one joint before it underwent structural failure.

The truss performed as it was expected to based on design and sensitivity analysis done with MATLAB. The truss was a relatively simple design project, but projects in the real world can involve expensive and specialized materials and take many people and many years to build, so it is important to understand how structures will perform when designing them, without having the materials to build them present.

Appendix

A. MATLAB code and functions

```
clear
clc
inputfile = '1Design_1.txt';
outputfile = '1Design_1out.txt';
% read input file
[joints,connectivity,reacjoints,reacvecs,loadjoints,loadvecs]=readinput3d(inputfile);

% creates a loadvec that accounts for weight of bar
[loadvecs] = barloading3D(joints,loadjoints,loadvecs,connectivity);

% compute forces in bars and reactions
[barforces,reacforces]=forceanalysis3d(joints,connectivity,reacjoints,reacvecs,loadjoints,loadvecs);

% write outputfile
writeoutput3d(outputfile,inputfile,barforces,reacforces,joints,connectivity,reacjoints,reacvecs,loadjoi

% plot truss (used in Lab 2)
joints3D=zeros(size(joints,1),3);
joints3D(:,1:3)=joints;
plottruss3d(joints3D,connectivity,barforces,reacjoints,3*[0.025,0.04,0.05],[0 0 1 1])

function [joints,connectivity,reacjoints,reacvecs,loadjoints,loadvecs]=readinput3d(inputfile)

% open inputfile
fid=fopen(inputfile);

if fid<0;error('inputfile does not exist');end

% initialze counters and input block id
counter=0;
inpblk=1;

% read first line
line=fgetl(fid);

% read input file
while line > 0

    % check if comment
```

```
if strcmp(line(1),'#')
    % read next line and continue
    line=fgetl(fid);
    continue;
end

switch inpblk

    case 1 % read number of joints, bars, reactions, and loads

        dims=sscanf(line,'%d%d%d%d%d');

        numjoints = dims(1);
        numbars   = dims(2);
        numreact  = dims(3);
        numloads  = dims(4);

        % check for correct number of reaction forces
        if numreact~=6; error('incorrect number of reaction forces');end

        % initialize arrays
        joints       = zeros(numjoints,3);
        connectivity = zeros(numbars,2);
        reacjoints   = zeros(numreact,1);
        reacvecs     = zeros(numreact,3);
        loadjoints   = zeros(numloads,1);
        loadvecs     = zeros(numloads,3);

        % check whether system satisfies static determiancy condition
        if 3*numjoints - 6 ~= numbars
            error('truss is not statically determinate');
        end

        % expect next input block to be joint coordinates
        inpblk = 2;

    case 2 % read coordinates of joints

        % increment joint id
        counter = counter + 1;

        % read joint id and coordinates;
        tmp=sscanf(line,'%d%e%e%e');

        % extract and check joint id
        jointid=tmp(1);
        if jointid>numjoints || jointid<1
            error('joint id number need to be smaller than number of joints and larger than 0');
        end

        % store coordinates of joints
        joints(jointid,:)=tmp(2:4);

        % expect next input block to be connectivity
```

```
    if counter==numjoints
        inpblk  = 3;
        counter = 0;
    end

case 3 % read connectivity of bars

    % increment bar id
    counter = counter + 1;

    % read connectivity;
    tmp=sscanf(line,'%d%d%d');

    % extract bar id number and check
    barid=tmp(1);
    if barid>numbars || barid<0
        error('bar id number needs to be smaller than number of bars and larger than 0');
    end

    % check joint ids
    if max(tmp(2:3))>numjoints || min(tmp(2:3))<1
        error('joint id numbers need to be smaller than number of joints and larger than 0');
    end

    % store connectivity
    connectivity(barid,:)=tmp(2:3);

    % expect next input block to be reaction forces
    if counter==numbars
        inpblk  = 4;
        counter = 0;
    end

case 4 % read reation force information

    % increment reaction id
    counter = counter + 1;

    % read joint id and unit vector of reaction force;
    tmp=sscanf(line,'%d%e%e%e');

    % extract and check joint id
    jointid=tmp(1);
    if jointid>numjoints || jointid<1
        error('joint id number need to be smaller than number of joints and larger than 0');
    end

    % extract untit vector and check length
    uvec=tmp(2:4);
    uvec=uvec/norm(uvec);

    % store joint id and unit vector
    reacjoints(counter)  = jointid;
    reacvecs(counter,:)  = uvec;
```

```
            % expect next input block to be external loads
            if counter==numreact
                inpblk  = 5;
                counter = 0;
            end

        case 5 % read external load information

            % increment reaction id
            counter = counter + 1;

            % read joint id and unit vector of reaction force;
            tmp=sscanf(line,'%f%f%f%f');

            % extract and check joint id
            jointid=tmp(1);
            if jointid>numjoints || jointid<1
                error('joint id number need to be smaller than number of joints and larger than 0');
            end

            % extract force vector
            frcvec=tmp(2:4);

            % store joint id and unit vector
            loadjoints(counter) = jointid;
            loadvecs(counter,:) = frcvec;

            % expect no additional input block
            if counter==numloads
                inpblk  = 99;
                counter = 0;
            end

        otherwise
            %fprintf('warning: unknown input: %s\n',line);
    end

    % read next line
    line=fgetl(fid);
end

% close input file
fclose(fid);

end

function [barforces,reacforces]=forceanalysis3d(joints,connectivity,reacjoints,reacvecs,loadjoints,loadv


numjoints = size(joints,1);
numbars   = size(connectivity,1);
numreact  = size(reacjoints,1);
numloads  = size(loadjoints,1);
```

```
% number of equilibruim equations
numeqns = 3 * numjoints;

% allocate arrays for linear system
Amat = zeros(numeqns);
bvec = zeros(numeqns,1);

% build Amat - loop over all joints

for i=1:numjoints

   % equation id numbers
   idx = 3*i-2;
   idy = 3*i-1;
   idz = 3*i;

   % get all bars connected to joint
   [ibar,ijt]=find(connectivity==i);

   % loop over all bars connected to joint
   for ib=1:length(ibar)

       % get bar id
       barid=ibar(ib);

       % get coordinates for joints "i" and "j" of bar "barid"
       joint_i = joints(i,:,:);
       if ijt(ib) == 1
           jid = connectivity(barid,2);
       else
           jid = connectivity(barid,1);
       end
       joint_j = joints(jid,:,:);

       % compute unit vector pointing away from joint i
       vec_ij = joint_j - joint_i;
       uvec   = vec_ij/norm(vec_ij);

       % add unit vector into Amat
       Amat([idx idy idz],barid)=uvec;
   end
end

% build contribution of support reactions
for i=1:numreact

    % get joint id at which reaction force acts
    jid=reacjoints(i);

    % equation id numbers
    idx = 3*jid-2;
    idy = 3*jid-1;
    idz = 3*jid;
```

```
    % add unit vector into Amat
    Amat([idx idy idz],numbars+i)=reacvecs(i,:);
end

% build load vector
% build contribution of support reactions
for i=1:numreact

    % get joint id at which reaction force acts
    jid=reacjoints(i);

    % equation id numbers
    idx = 3*jid-2;
    idy = 3*jid-1;
    idz = 3*jid ;

    % add unit vector into Amat
    Amat([idx idy idz],numbars+i)=reacvecs(i,:);
end

% build load vector

for i=1:numloads

    % get joint id at which external force acts
    jid=loadjoints(i);

    % equation id numbers
    idx = 3*jid-2;
    idy = 3*jid-1;
    idz = 3*jid ;

    % add unit vector into bvec (sign change)
    bvec([idx idy idz])= -loadvecs(i,:);

    for i = 1:length(joints(:,1))
        bvec(3*i,1) = bvec(3*i,1) + loadvecs(i,3);
    end

end
% check for invertability of Amat
if rank(Amat) ~= numeqns
    error('Amat is rank defficient: %d < %d\n',rank(Amat),numeqns);
end

% solve system
xvec=Amat\bvec;

% extract forces in bars and reaction forces
barforces=xvec(1:numbars);
reacforces=xvec(numbars+1:end);

end
```

```
function [weight] = weight(density,joints,connectivity,mass_joint)
%WEIGHT [weight] = weight(1.214,joints,connectivity,8.36)
mass = zeros(length(joints),1);
for i = 1:length(joints)
    [a,b] = find (connectivity == i);
    for j = 1:length(a)
        id1 = connectivity(a(j),b(j));
        if b(j)==1
            id2 = connectivity(a(j),2);
        elseif b(j)==2
            id2 = connectivity(a(j),1);
        end
        len = (1/2)*((joints(id1,1)-joints(id2,1))^2+(joints(id1,2)-joints(id2,2))^2+(joints(id1,3)-joi
    mass(i) = mass(i) + len;
    end
end

mass = density .* mass;
mass = mass + mass_joint;
weight = -9.81 .* mass;
%weight = -386.220681 .* mass;

end

function plottruss3d(xyz,topo,eforce,fbc,rads,pltflags)

figure(1);
clf;

% check input

if nargin < 6; error('routine requires 6 input parameters'); end

% extract

[numnode, dim]=size(xyz);
numelem = size(topo,1);

if dim ~= 3
   display('Error in plottruss: 3 coordinates are needed for array xyz');
   return;
end

% extract min and max force values

minfrc=floor(min(eforce));
maxfrc=ceil(max(eforce));

% define radius of bars

if length(rads) == 1
    radb = rads(1);
    radj = 1.75*radb;
```

```
    radk = 1.5*radj;
else
    radb = rads(1);
    radj = rads(2);
    radk = rads(3);
end

% plot bars

figure(1)

for i=1:numelem
    na=topo(i,1);
    nb=topo(i,2);
    [xc,yc,zc]=plotbar(xyz(na,:),xyz(nb,:),radb);
    cc=eforce(i)*ones(size(xc));
    surf(xc,yc,zc,cc,'EdgeColor','none');
    if pltflags(2) > 0
        text((xyz(na,1)+xyz(nb,1))/2+1.8*radb,(xyz(na,2)+xyz(nb,2))/2+1.8*radb,(xyz(na,3)+xyz(nb,3))/2+
    end
    if pltflags(3) > 0
        text((xyz(na,1)+xyz(nb,1))/2+1.8*radb,(xyz(na,2)+xyz(nb,2))/2+1.8*radb,(xyz(na,3)+xyz(nb,3))/2+
    end
    hold on;
end

% plot node id numbers

if pltflags(1) > 0
    for i=1:numnode
            text(xyz(i,1)+1.8*radj,xyz(i,2)+1.5*radj,xyz(i,3)+1.3*radj,num2str(i));
    end
end

colorbar;
caxis([minfrc maxfrc])

% plot ball joints

for i=1:numnode
    [sx,sy,sz]=plotnode(xyz(i,:),radj);
    surf(sx,sy,sz,'EdgeColor','none','FaceColor','black');
    hold on;
end

% plot supported nodes

for i=1:length(fbc)
    na=fbc(i);
    [sx,sy,sz]=plotnode(xyz(na,:),radk);
    surf(sx,sy,sz,'EdgeColor','none','FaceColor','red');
    hold on;
end
```

```
% plot parameters

axis('equal');
title('Member forces');

lightangle(-45,30)
set(gcf,'Renderer','openGl')
set(findobj(gca,'type','surface'),...
    'FaceLighting','phong',...
    'AmbientStrength',.3,'DiffuseStrength',.8,...
    'SpecularStrength',.9,'SpecularExponent',25,...
    'BackFaceLighting','unlit')

if pltflags(4)
    % use default
else
    % set view point to 0,0,1
    view([0 0 1]);
end
end

%function truss3d(inputfile,outputfile)
% function truss2d(inputfile,outputfile)
%
% Analysis of 3-D statically determinate truss
%
% Input:  inputfile  - name of input file
%         outputfile - name of output file
%
% Author: Kurt Maute for ASEN 2001, Sept 21 2011


%function truss3dmcs(inputfile)
%
% Stochastic analysis of 3-D statically determinate truss by
% Monte Carlo Simulation. Only positions and strength of joints
% treated as random variables
%
% Assumption: variation of joint strength and positions described
%             via Gaussian distributions
%
%             joint strength : mean = 4.8
%                              coefficient of variation = 0.4
%             joint position :
%                              coefficient of variation = 0.01
%                              (defined wrt to maximum dimension of truss)
%
%             number of samples is set to 1e5
%
% Input:  inputfile  - name of input file
%
% Author: Kurt Maute for ASEN 2001, Oct 13 2012
clear
clc
```

```
inputfile = 'trussqm.inp';
% parameters
jstrmean   = 4.8;   % mean of joint strength 4.8 N (converted to lbs*in /s^2)
%jstrmean   = 416.50416192;
jstrcov    = 0.08;  % coefficient of variation (sigma/u) of joint strength = 0.4/4.8
jposcov    = 0.01;  % coefficient of variation of joint position percent of length of truss (ext)
numsamples = 1e4;   % number of samples

% read input file
[joints,connectivity,reacjoints,reacvecs,loadjoints,loadvecs]=readinput3d(inputfile);

% determine extension of truss
ext_x=max(joints(:,1))-min(joints(:,1));   % extension in x-direction
ext_y=max(joints(:,2))-min(joints(:,2));   % extension in y-direction
ext_z=max(joints(:,3))-min(joints(:,3));   % extension in z-direction
ext  =max([ext_x,ext_y,ext_z]);

% loop overall samples
numjoints=size(joints,1);       % number of joints
maxforces=zeros(numsamples,1);  % maximum bar forces for all samples
maxreact=zeros(numsamples,1);   % maximum support reactions for all samples
failure=zeros(numsamples,1);    % failure of truss

for is=1:numsamples

    % generate random joint strength limit
    varstrength = (jstrcov*jstrmean)*randn(1,1);

    jstrength = jstrmean + varstrength;

    % generate random samples
    varjoints = (jposcov*ext)*randn(numjoints,3);

    % perturb joint positions
    randjoints = joints + varjoints;

    % compute forces in bars and reactions
    [barforces,reacforces] = forceanalysis3d(randjoints,connectivity,reacjoints,reacvecs);

    % determine maximum force magnitude in bars and supports
    maxforces(is) = max(abs(barforces));
    maxreact(is)  = max(abs(reacforces));

    % determine whether truss failed
    failure(is) = maxforces(is) > jstrength || maxreact(is) > jstrength;
end

FB_mean = mean(maxforces);
FB_dev = std(maxforces);
F_design = icdf('Normal', 0.05, FB_mean, FB_dev)

n = (FB_mean - F_design) / (FB_dev);
FOS = 1 / (1 - n*(FB_dev / FB_mean))
```

```
figure(1);
subplot(1,2,1);
histogram(maxforces,30);
title('Histogram of maximum bar forces');
xlabel('Magnitude of bar forces');
ylabel('Frequency');

subplot(1,2,2);
histogram(maxreact,30);
title('Histogram of maximum support reactions');
xlabel('Magnitude of reaction forces');
ylabel('Frequency');

fprintf('\nFailure probability : %e \n\n',sum(failure)/numsamples);
```

B. CDR Slides

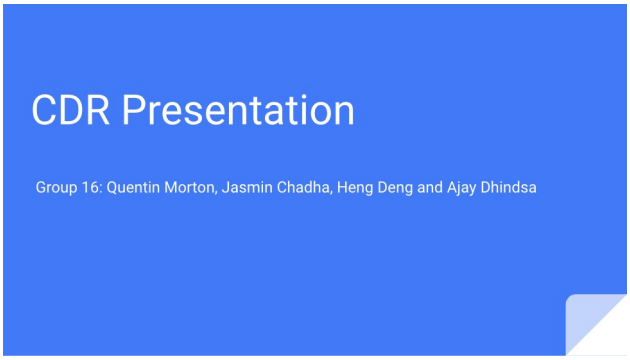

Fig. 5 CDR Slide 1

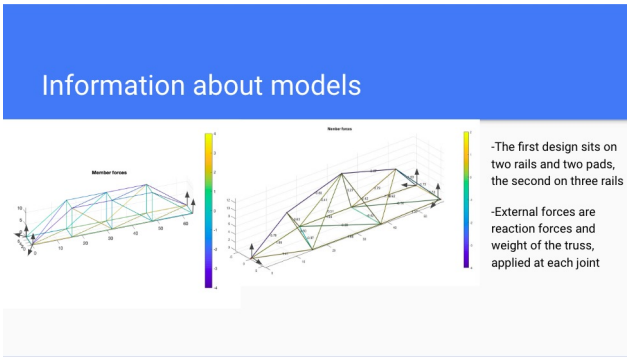

Fig. 6 CDR Slide 2

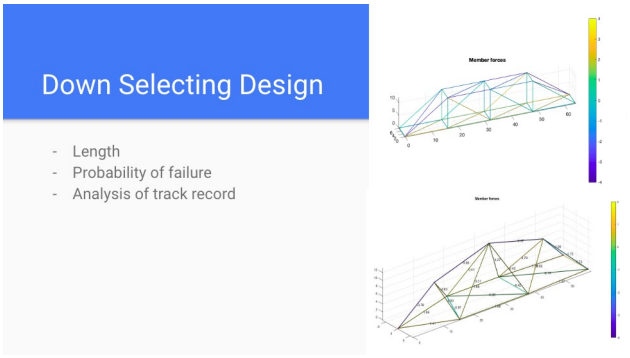

Fig. 7 CDR Slide 3

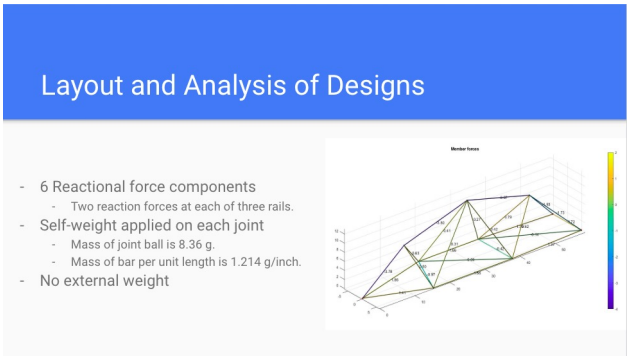

Fig. 8 CDR Slide 4

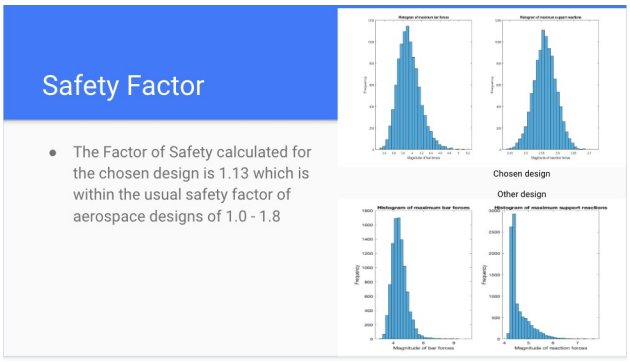

Fig. 9 CDR Slide 5

References

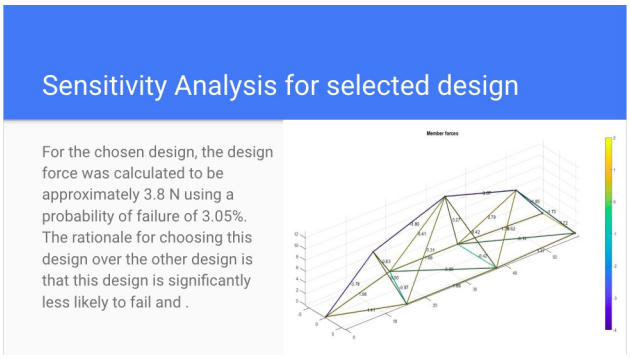

Fig. 10 CDR Slide 6

Fig. 11 CDR Slide 7

Lab 2 Description 2019 PDF

Hibbeler, R. C., Statics and Mechanics of Materials, 5th ed., Pearson, Hoboken, 2017.