Flex

A GUI-Driven Plate Deformation Solver

Jack Kolb, Michael Hoang, Adedayo Adeyemi

ME18, Intro. to Engineering Computation

December 5th, 2017

Functionality:
This program accepts a number of dimensions from the user describing a plate and loads, and visually plots a 3D, 2D, or contour plot of the resulting deformation.

The model is simple: the GUI serves as a base for collecting information, which is sent through a control function. The control function creates a master deformation matrix and adds each load's deformations to the master matrix using sub-functions. Finally, this matrix is plotted onto the GUI's plot axes.

The control function reads loads through a cell array; this means an "infinite" number of loads can be added to the model. We were unable to create a good interface for this through GUIDE (MatLab's GUI tool), thus the GUI limits tests to a maximum of one surface and one point load. Through the command line there is no such limitation. We chose a cell array to convey our loads as it allows usage of multiple data types within one structure, as exampled below:

```
{
 'surface', 10, 0, 0;
 'point', 5, 15, 15;
}
```

To add additional loads, simply append another line to the structure.

The maximum deflection is displayed above the plot; a colorbar is shown to the side.

Overall, our program is quite intuitive: to use, fill in the desired information and press "Plot"; to clear current contents, press "Clear".

Development Procedures:
Our team utilized Git, a widely used version control software, to maintain a central codebase that we can all edit. Git allowed us to remotely push our latest code changes, resolve code conflicts, and pull the latest from our online repository, allowing our team to work independently on modules that were easily interoperated at the end. You can find our git repository at https://github.com/jackkolb/flex

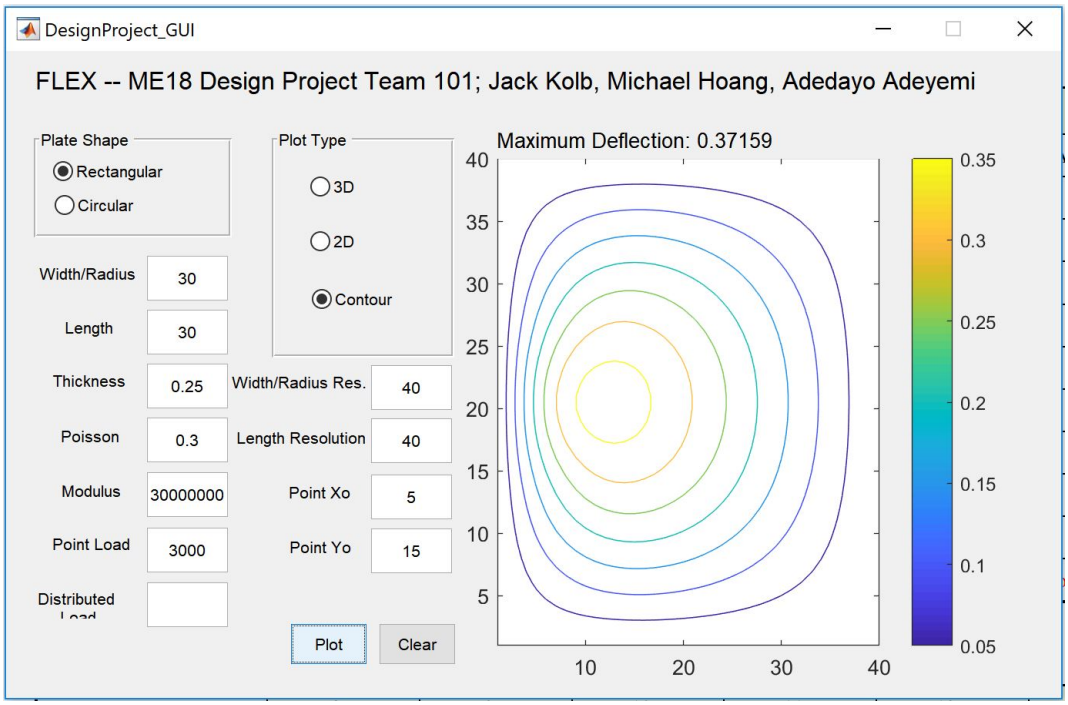
Git required us to learn basic command prompt/terminal, and proved highly effective in managing our code.

We divided the coding work throughout our group, creating several interconnected functions: the control function calls one (or several) sub-functions for the deformation calculations, one for each primary use case: rectangular_point, rectangular_surface, circular_point, and circular_surface. Through this method we did not need to meet as a group, as module deadlines and a "only work on what you are assigned" protocol prevented conflicts. Our project was very well-organized!

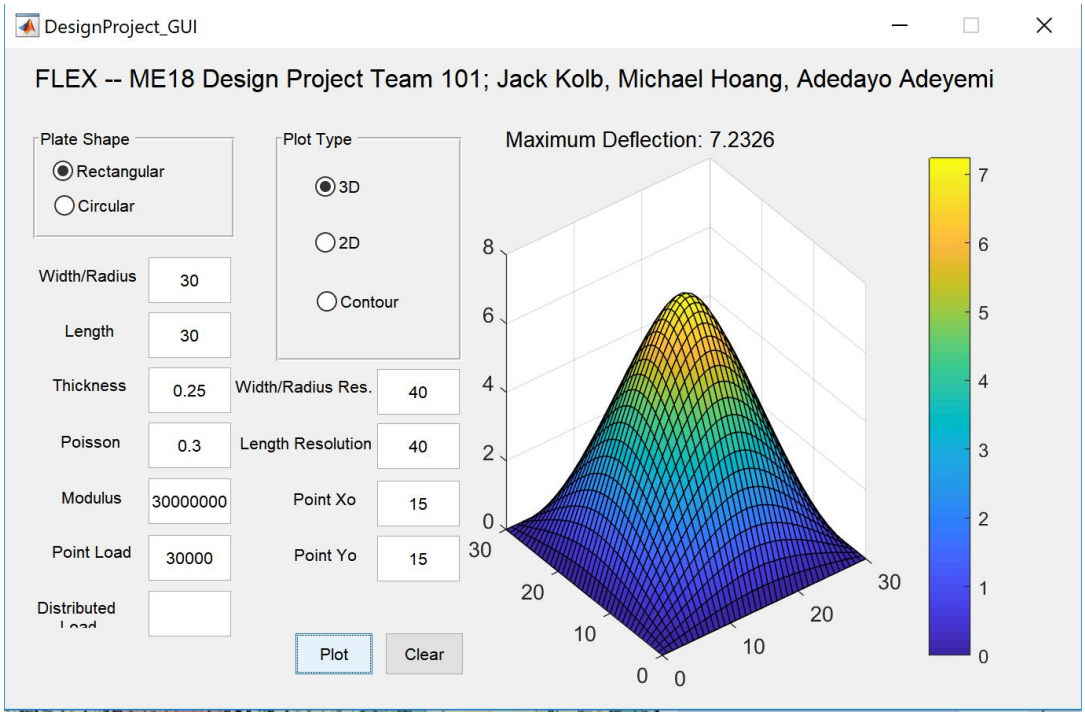
Testing Procedures:
Our team tested our results through two methods: raw calculations, and comparisan with ~5 other groups (which through majority consistency allowed us to cross-reference debug). Overall, we found the comparisan tests, which can quickly check a number of tests, superior to individual calculations, especially in rectangular test cases were a number of summations were necessary while hand-calculations were prone to error. As long as the algorithm is correct, it fits any scale.

Test Cases:

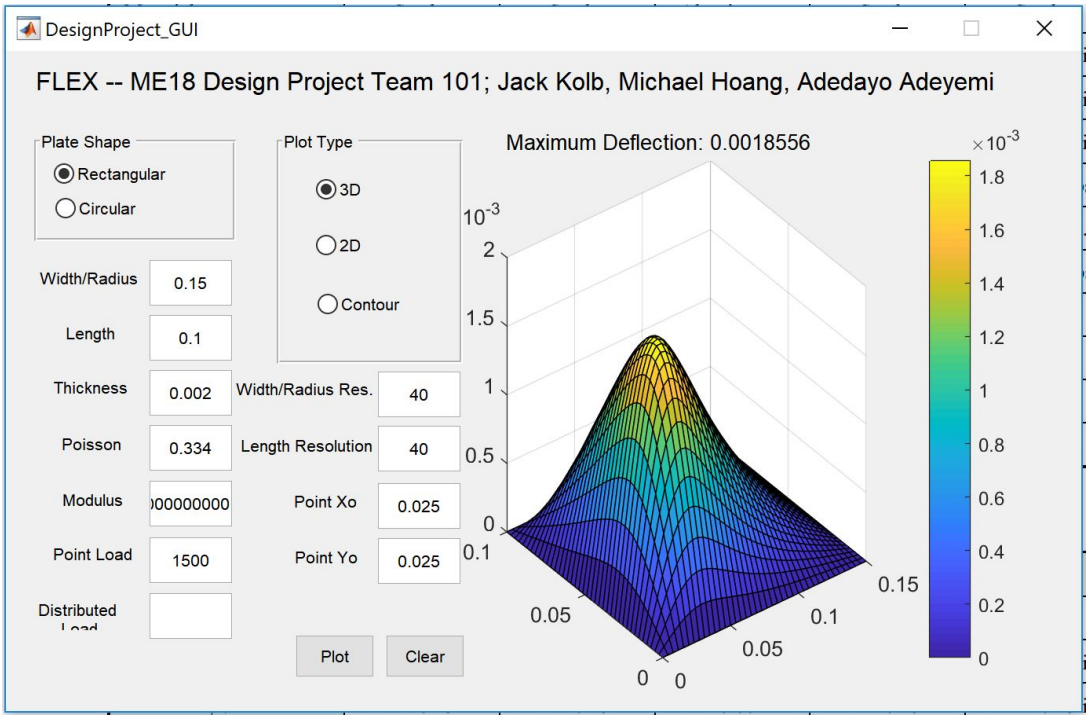
Test Case 1:



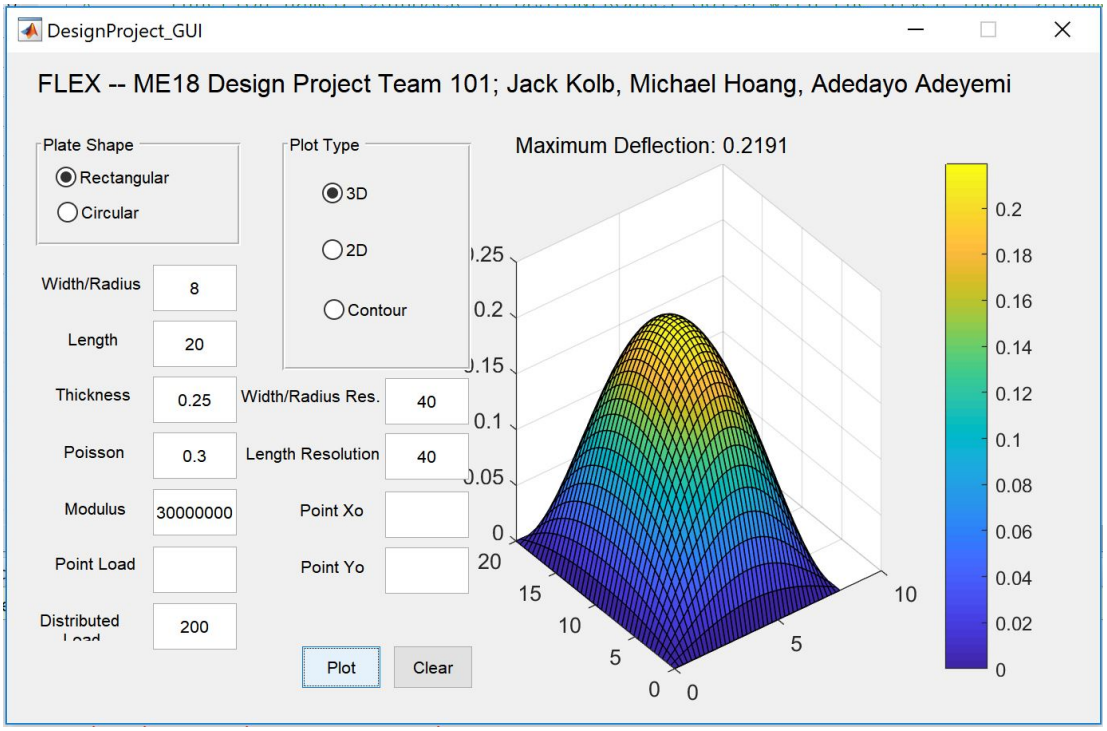
Test Case 2:



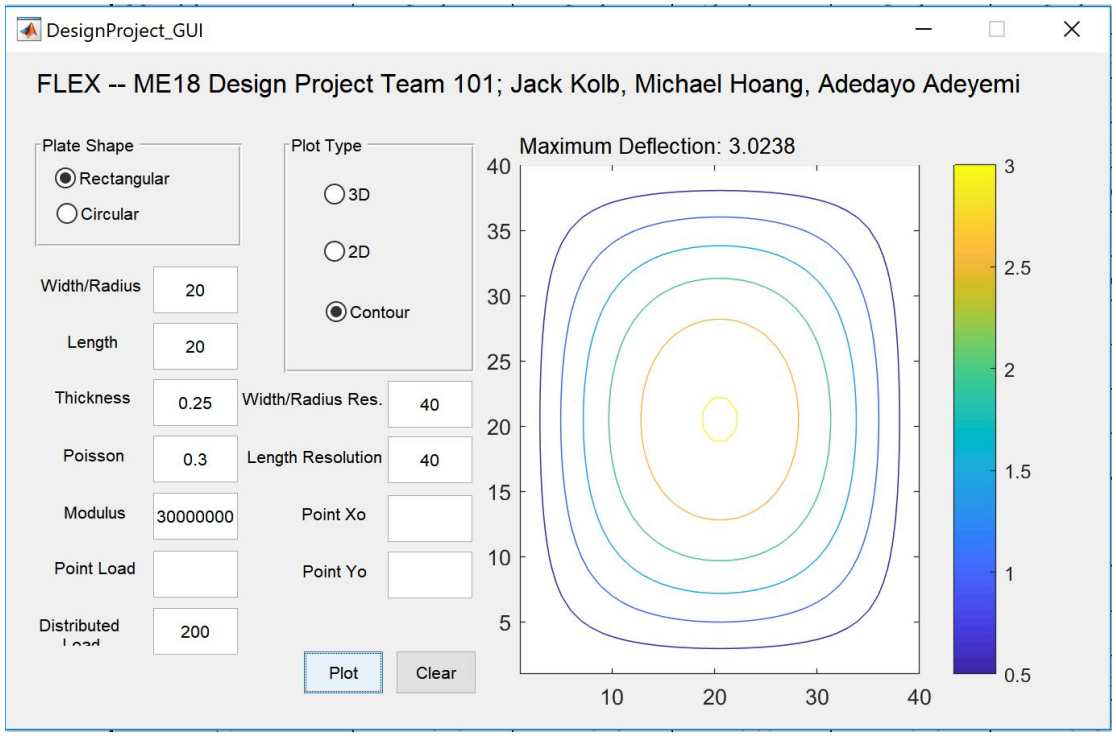
Test Case 3:



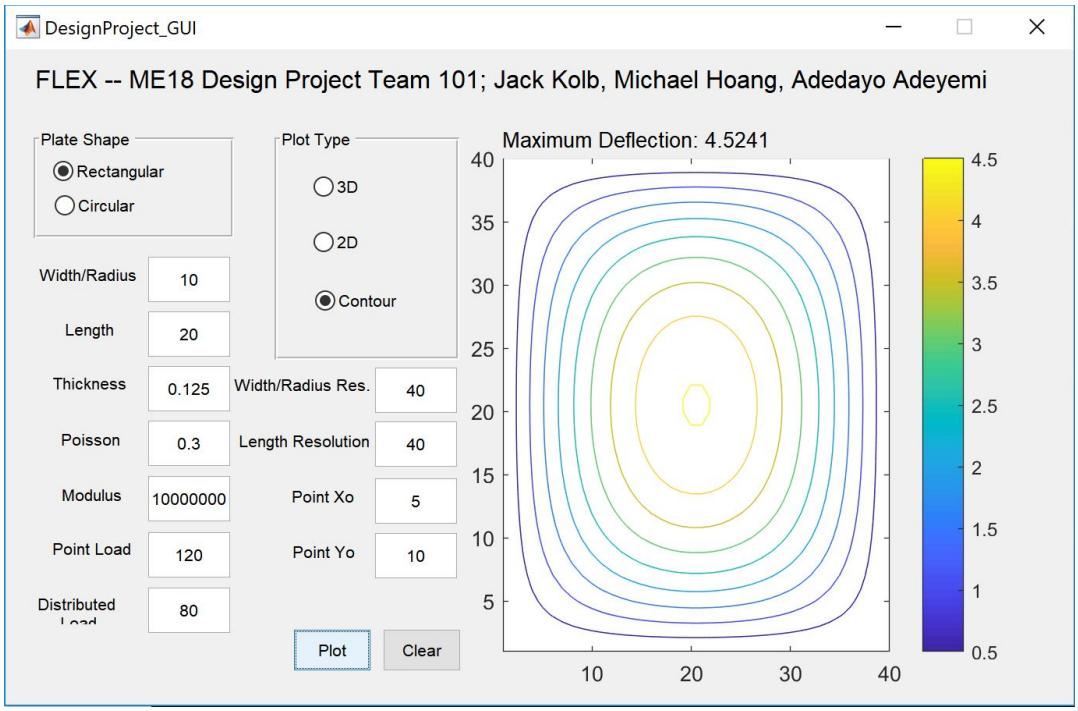
Test Case 4:



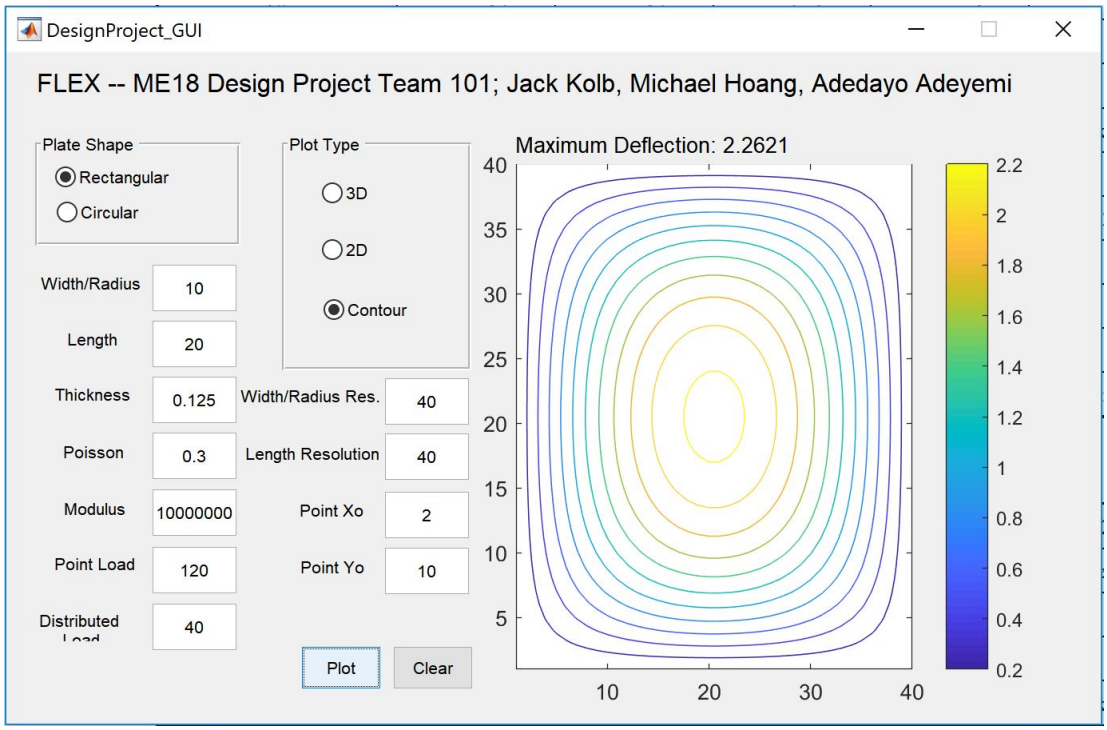
Test Case 5:



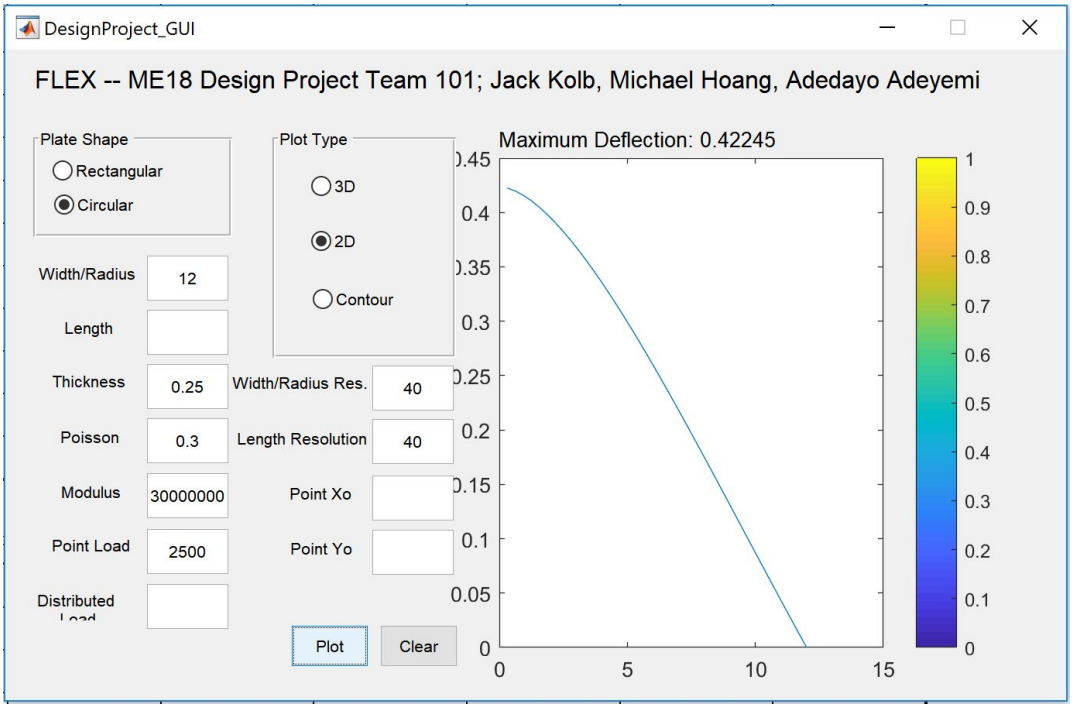
Test Case 6:



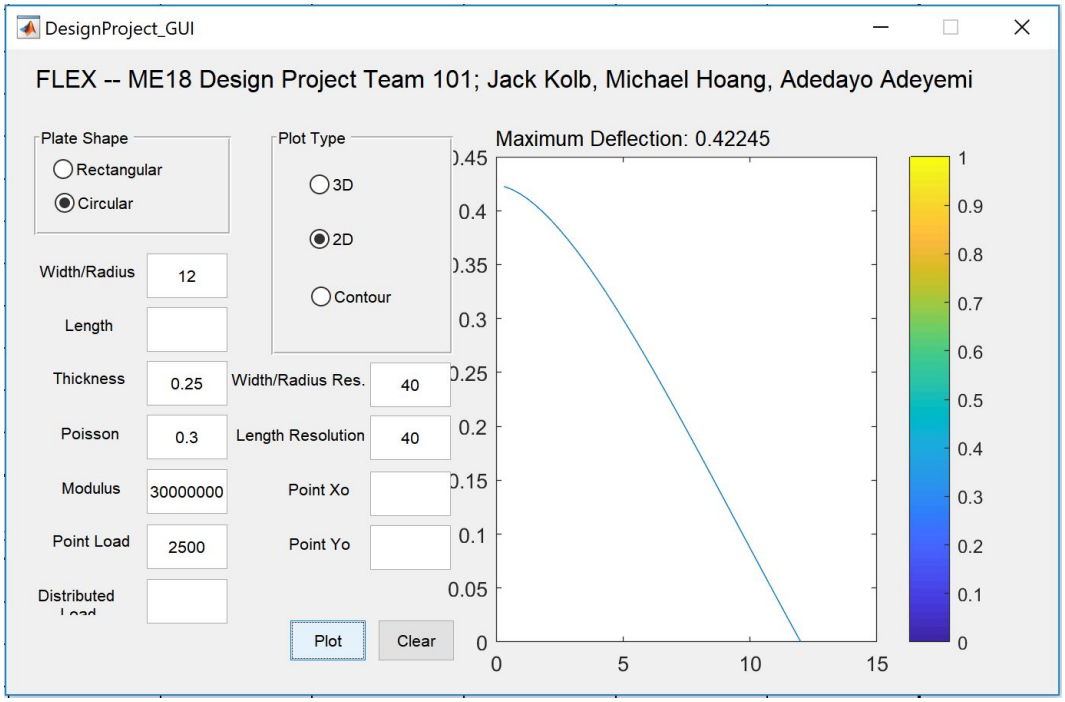
Test Case 7:



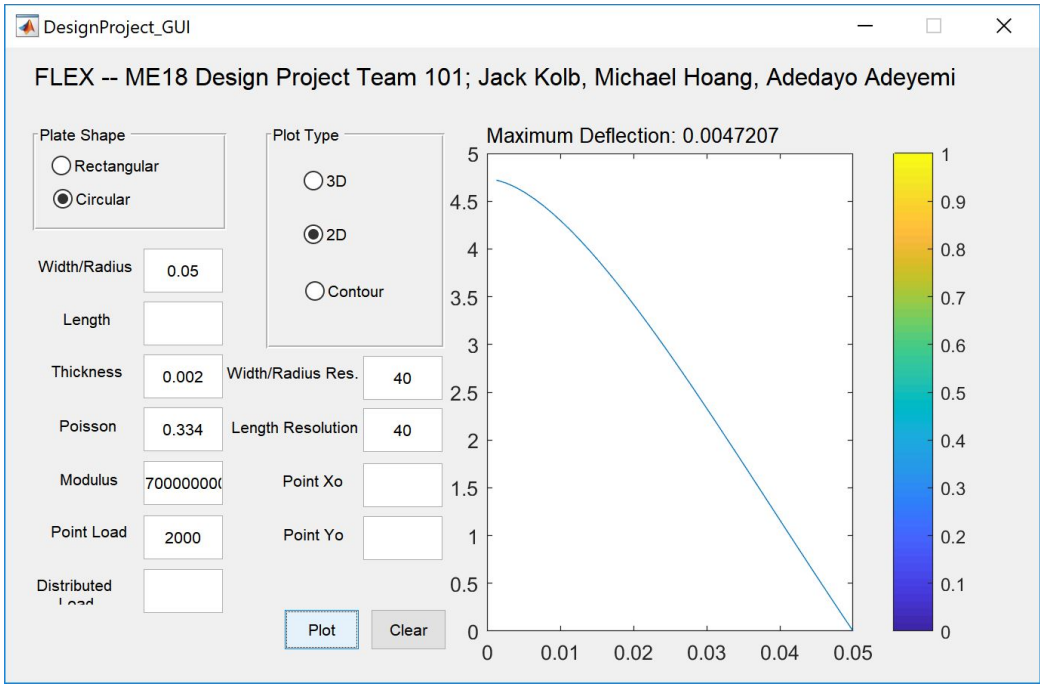
Test Case 8:



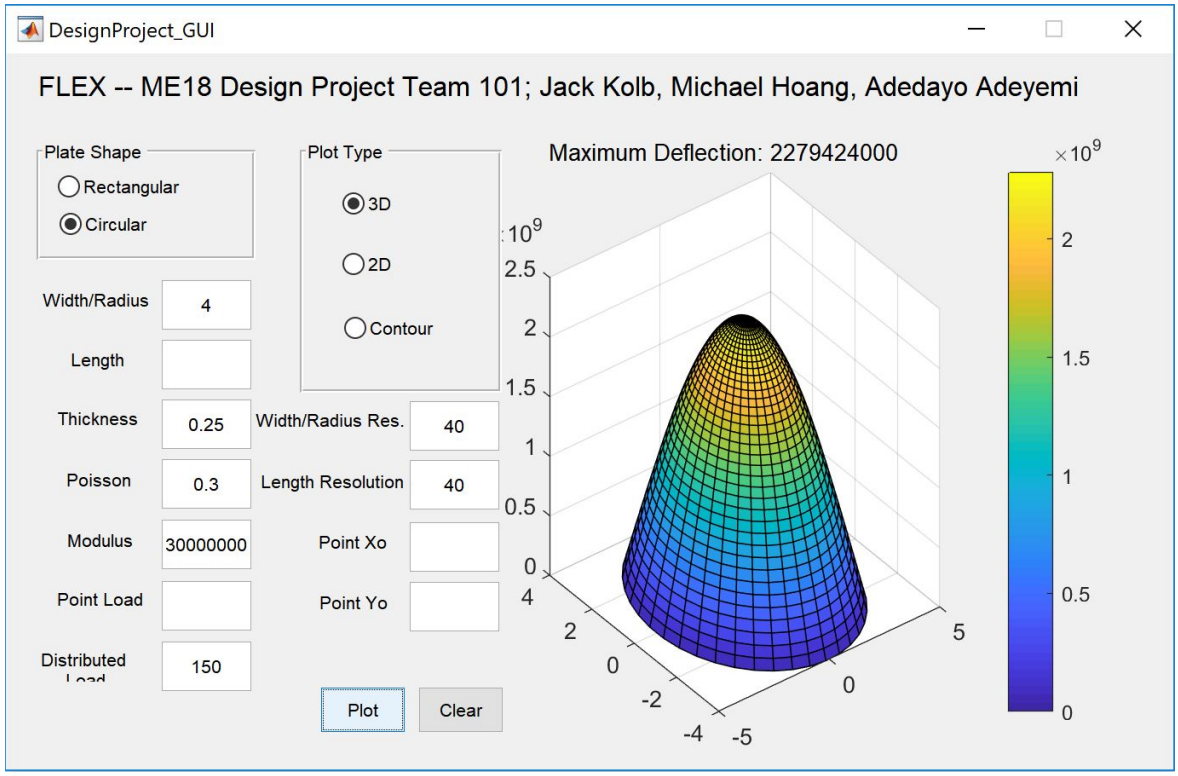
Test Case 9:



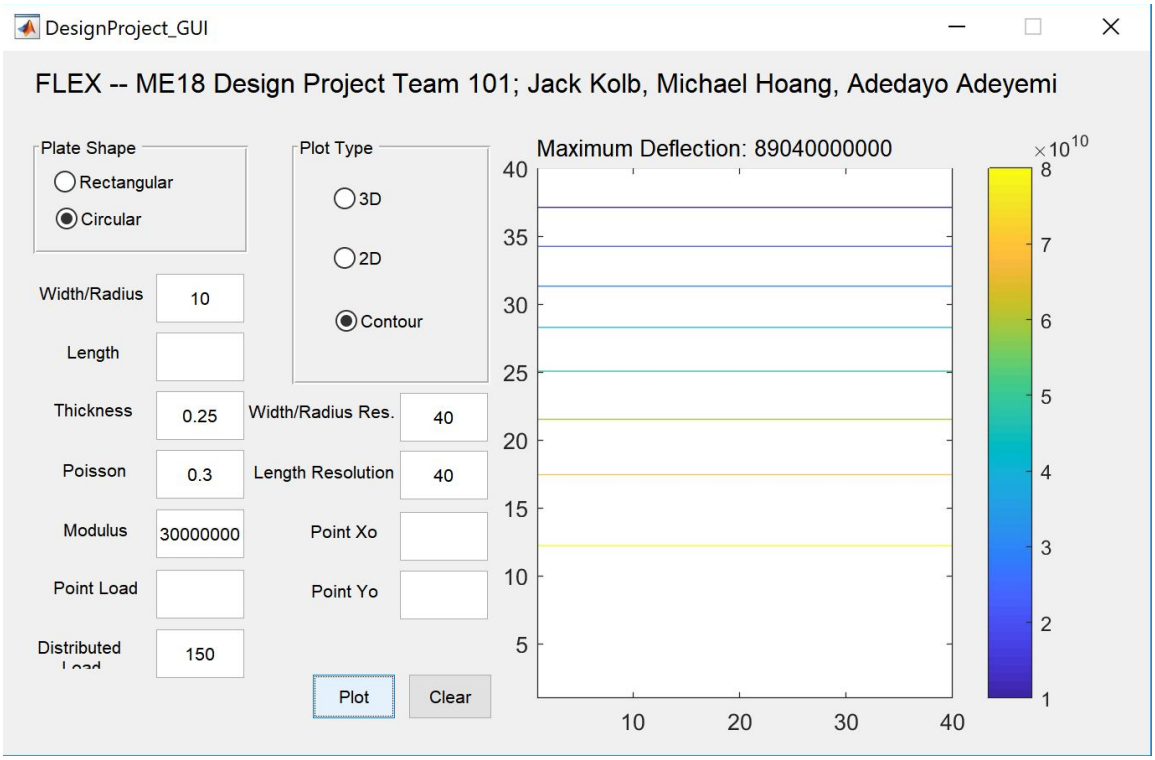
Test Case 10:



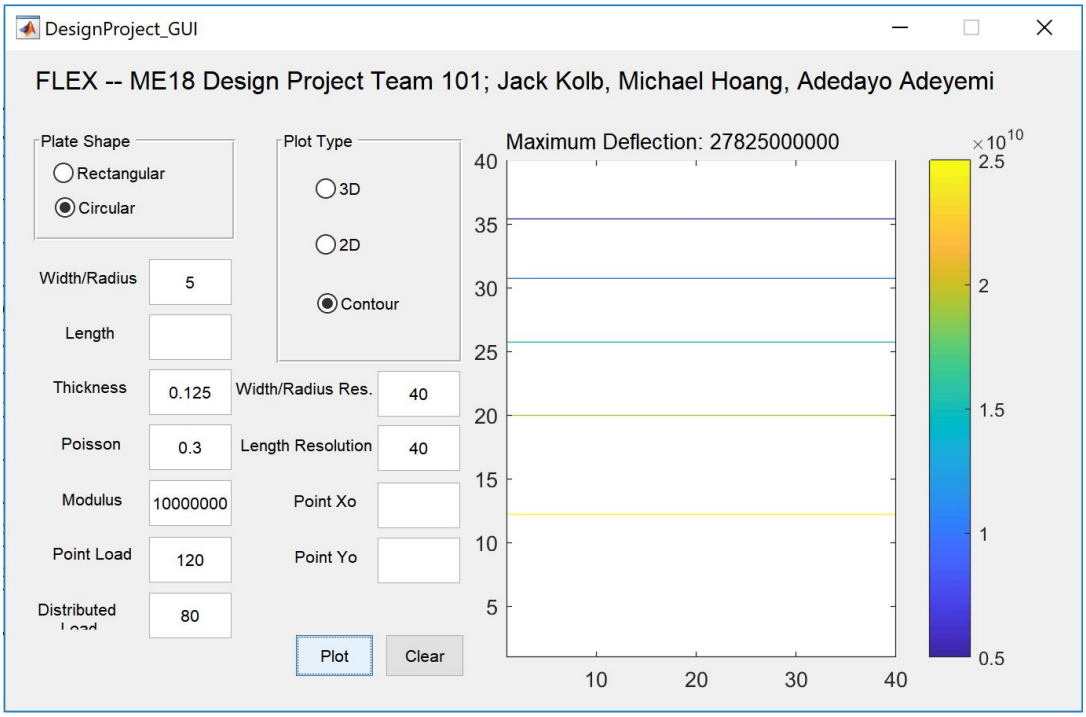
Test Case 11:



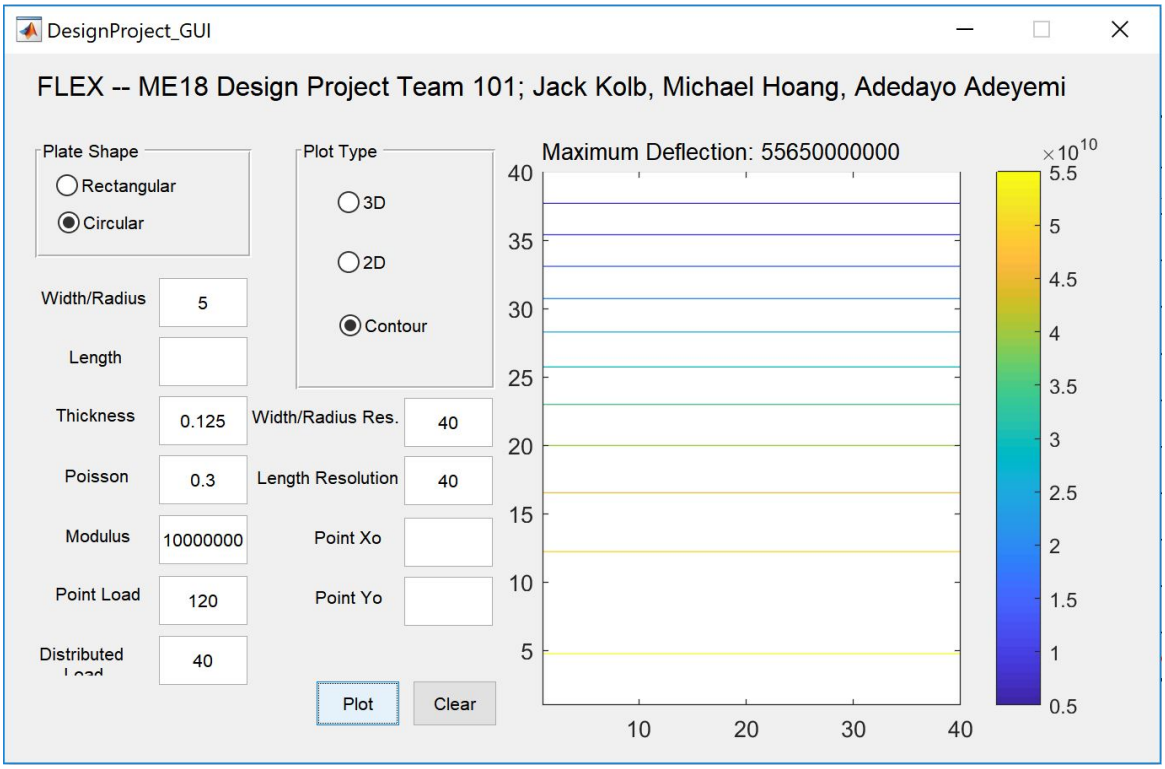
Test Case 12:



Test Case 13:



Test Case 14:



Function File (plate_deformation.m):

```
function [ max_val ] = plate_deformation( shape, loads, plot_type, x, y, xres, yres,
height, poisson, modulus, main_axes )
% Controller function for the plate deformations
% Input loads as a n by 4 cell array, of type, load, x0, y0 (use 0 for x/y surface
loads):
% ex: {'point', 50, 10, 10; 'point', 100, 30, 30; 'surface', 82, 0, 0}

% ::catch errors::
% shape, load_type, or plot_type are invalid
shapes = ['circ', 'rect'];
plot_types = ['2d', 'contour', '3d'];
load_types = ['p', 's'];

if ~ismember(shapes, shape)
    error('plate_deformation:invalid_parameter', 'inputted shape is invalid (need circ or
rect)');
elseif ~ismember(plot_types, plot_type)
    error('plate_deformation:invalid_parameter', 'inputted plot_type is invalid (need 2d,
3d, or contour)');

% x, y, xres, yres, or height <= 0; modulus, or poisson < 0
elseif x <= 0
    error ('plate_deformation:invalid_parameter', 'inputted width is <= 0');
elseif y <= 0
    error ('plate_deformation:invalid_parameter', 'inputted length is <= 0');
elseif xres <= 0
    error ('plate_deformation:invalid_parameter', 'inputted width resolution is <= 0');
elseif yres <= 0
    error ('plate_deformation:invalid_parameter', 'inputted length resolution is <= 0');
elseif modulus < 0
    error ('plate_deformation:invalid_parameter', 'inputted modulus is <= 0');
elseif poisson < 0
    error ('plate_deformation:invalid_parameter', 'inputted poisson is <= 0');
end

% set initial matrix to zeros
total_deflection_matrix = zeros(xres, yres);

% for each load, add to the deflection matrix
sz = size(loads);
for rr=1:sz(1)
    % get load data from load matrix row
    load_type = loads{rr, 1};
    load = loads{rr, 2};

    % check that load is valid
    if load < 0
        error ('plate_deformation:invalid_parameter', 'inputted load is < 0');
    elseif ~ismember(load_types, load_type)
        error ('plate_deformation:invalid_parameter', 'load type is invalid (need point
or surface)');
    end

% calculate deflection matrix of this load, add to overall deflection
% matrix
switch(shape)
    case 'rect'
        % if load is a point load...
        if strcmp(load_type, 'point')
```

```
            x0 = loads{rr, 3};
            y0 = loads{rr, 4};
            def_mat = rect_plate_point_deform(x, y, xres, yres, height, load, x0, y0,
poisson, modulus);
        % if load is a surface load...
        elseif strcmp(load_type, 'surface')
            def_mat = rect_plate_surface_deform(x, y, xres, yres, height, load, poisson,
modulus);
        end
        % Add the calculated deflection matrix to the total deflection
        % matrix
        Z = def_mat;
        [X, Y] = meshgrid(linspace(0, x, xres), linspace(0, y, yres));
        total_deflection_matrix = total_deflection_matrix + Z;

    case 'circ'
        % if load is a point load...
        if strcmp(load_type, 'point')
            [def_vec, def_mat] = circ_plate_point_deform(x, xres, yres, height, load,
poisson, modulus);
        % if load is a surface load...
        elseif strcmp(load_type, 'surface')
            [def_vec, def_mat] = circ_plate_surface_deform(x, xres, yres, height, load,
poisson, modulus);
        end
        % Add the calculated deflection matrix to the total deflection
        % matrix
        [TH,R] = meshgrid(linspace(0,2*pi,yres), linspace(0,x,xres));
        [X,Y,Z] = pol2cart(TH,R,def_mat);
        total_deflection_matrix = total_deflection_matrix + Z;
end
end

% plot the deflection matrix
if strcmp(plot_type, '2d') && strcmp(shape, 'circ')
    plot(linspace(0, x, xres), def_vec, 'parent', main_axes);
elseif strcmp(plot_type, 'contour') || strcmp(plot_type, '2d')
    contour(Z, 'parent', main_axes);
elseif strcmp(plot_type, '3d')
    surf(X, Y, Z, 'parent', main_axes);
end

colorbar();

% set the max deflection (which will be returned)
max_val = max(Z(:));
end

function [ deformation ] = rect_plate_point_deform( width, length, wres, lres, height,
load, x0, y0, poiss, modu )
% calculates the deformation of a rectangular plate (load at point x, y)
%   returns a matrix of the deformations across the plate surface

    % set x and y vectors
    x = linspace(0, width, wres);
    y = linspace(0, length, lres);

    % set deformation matrix to zeroes
    deformation = zeros([lres wres]);

    % calculate base values (D, base_deform)
    D = (modu*height^3) / (12 * (1-poiss^2));
    base_deform = (4*load)/(pi^4*width*length*D);
```

```
    % m and n are vectors used through summation in the deformation
    % calculations
    m = 1:10;
    n = 1:10;

    % iterate through x and y, calculating the deflection at that point,
    % placing the deformation into its corresponding slot in the
    % deformation matrix
    x_count = 0;
    for xx = x
        x_count = x_count + 1;
        y_count = 0;
        for yy = y
            y_count = y_count + 1;

            sum_deform = 0;
            for mm = m
               for nn = n
                   sum_deform = sum_deform + sin(mm*pi*x0/width)*sin(nn*pi*y0/length) /
(mm^2/width^2+nn^2/length^2)^2 * sin(mm*pi*xx/width)*sin(nn*pi*yy/length);
               end
            end

            deformation(y_count, x_count) = base_deform * sum_deform;
        end
    end

end

function [ deformation ] = rect_plate_surface_deform( width, length, wres, lres, height,
load, poiss, modu )
% calculates the deformation of a rectangular plate (load over surface)

    % set x and y vectors
    x = linspace(0, width, wres);
    y = linspace(0, length, lres);

    % set deformation matrix to zeros
    deformation = zeros([lres wres]);

    % calculate base constants (D, base_deform)
    D = (modu*height^3) / (12 * (1-poiss^2));
    base_deform = (16*load)/(pi^6*D);

    % m and n are described as odd in this case
    m = 1:2:11;
    n = 1:2:11;

    % iterate through x and y, calculating the deflection at that point,
    % placing the deformation into its corresponding slot in the
    % deformation matrix
    x_count = 0;
    for xx = x
        x_count = x_count + 1;
        y_count = 0;
        for yy = y
            y_count = y_count + 1;

            sum_deform = 0;
            for mm = m
               for nn = n
```

```
                    sum_deform = sum_deform + sin(mm*pi*xx/width)*sin(nn*pi*yy/length) /
(mm*nn*(mm^2/width^2+nn^2/length^2)^2);
                end
            end
            deformation(y_count, x_count) = base_deform * sum_deform;
        end
    end

end

function [ deformation_vector, polar_deformation_matrix ] =
circ_plate_point_deform( radius, rres, tres, height, load, poiss, modu )
% calculates the deformation of a rectangular plate (load at point x, y)
%   returns a matrix of the deformations across the plate surface

    % set radius vector
    r = linspace(0, radius, rres);

    % set deformation vector initially to r (values will change)
    deformation_vector = r;

    % calculate base constants (D, base_deform)
    D = (modu*height^3) / (12 * (1-poiss^2));
    base_deform = load / (16*pi*D);

    % iterate through r, calculating the deflection at that radius,
    % placing the deformation into its corresponding slot in the
    % deformation vector
    r_count = 0;
    for rr = r
        r_count = r_count + 1;
        deformation_vector(r_count) = base_deform * ( (3+poiss)/(1+poiss)*(radius^2 -
rr^2) + 2 * rr ^ 2 * log(rr/radius));
    end

    % remap the deformation vector (r) into a polar deformation matrix
    % (r, t)
    polar_deformation_matrix = zeros([rres, tres]);
    for rr = 1:rres
        for th = 1:tres
            polar_deformation_matrix(rr, th) = deformation_vector(rr);
        end
    end
end

function [ deform, polar_deformation_matrix ] = circ_plate_surface_deform(radius, rres,
tres, height, elasticity, modulus, load)
    % finds the deformation of a circular plate with a distributed load

    % set radius vector
    r = linspace(0,radius,rres);

    % set deformation vector initially to r (values will change)
    deform = r;

    % calculate base constants (D, base_deform)
    D = elasticity*height^3/(12*(1-modulus^2));
    r_count=0;

    % iterate through r, calculating the deflection at that radius,
    % placing the deformation into its corresponding slot in the
    % deformation vector
    for rr = r
```

```
        r_count = r_count +1;
        deform(r_count) = load*(radius^2-rr^2)/(64*D)*((5+modulus)/(1+modulus)*(radius^2)
- rr^2);
    end

    % remap the deformation vector (r) into a polar deformation matrix
    % (r, t)
    polar_deformation_matrix = zeros([rres, tres]);
    for rr = 1:rres
        for th = 1:tres
            polar_deformation_matrix(rr, th) = deform(rr);
        end
    end
end
```

GUI File (plate_deformation.m):

```
function varargout = DesignProject_GUI(varargin)
% DESIGNPROJECT_GUI MATLAB code for DesignProject_GUI.fig
%      DESIGNPROJECT_GUI, by itself, creates a new DESIGNPROJECT_GUI or raises the
existing
%      singleton*.
%
%      H = DESIGNPROJECT_GUI returns the handle to a new DESIGNPROJECT_GUI or the handle
to
%      the existing singleton*.
%
%      DESIGNPROJECT_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DESIGNPROJECT_GUI.M with the given input arguments.
%
%      DESIGNPROJECT_GUI('Property','Value',...) creates a new DESIGNPROJECT_GUI or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before DesignProject_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to DesignProject_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help DesignProject_GUI

% Last Modified by GUIDE v2.5 05-Dec-2017 01:24:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @DesignProject_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @DesignProject_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before DesignProject_GUI is made visible.
function DesignProject_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to DesignProject_GUI (see VARARGIN)

% Choose default command line output for DesignProject_GUI
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes DesignProject_GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = DesignProject_GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function width_input_Callback(hObject, eventdata, handles)
% hObject    handle to width_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of width_input as text
%        str2double(get(hObject,'String')) returns contents of width_input as a double


% --- Executes during object creation, after setting all properties.
function width_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to width_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function length_input_Callback(hObject, eventdata, handles)
% hObject    handle to length_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of length_input as text
%        str2double(get(hObject,'String')) returns contents of length_input as a double


% --- Executes during object creation, after setting all properties.
function length_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to length_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



function thickness_input_Callback(hObject, eventdata, handles)
% hObject    handle to thickness_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of thickness_input as text
%        str2double(get(hObject,'String')) returns contents of thickness_input as a
double


% --- Executes during object creation, after setting all properties.
function thickness_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to thickness_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function poisson_input_Callback(hObject, eventdata, handles)
% hObject    handle to poisson_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of poisson_input as text
%        str2double(get(hObject,'String')) returns contents of poisson_input as a double


% --- Executes during object creation, after setting all properties.
function poisson_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to poisson_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function modulus_input_Callback(hObject, eventdata, handles)
% hObject    handle to modulus_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of modulus_input as text
%        str2double(get(hObject,'String')) returns contents of modulus_input as a double
```

```
% --- Executes during object creation, after setting all properties.
function modulus_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to modulus_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function point_input_Callback(hObject, eventdata, handles)
% hObject    handle to point_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of point_input as text
%        str2double(get(hObject,'String')) returns contents of point_input as a double


% --- Executes during object creation, after setting all properties.
function point_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to point_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function x0_input_Callback(hObject, eventdata, handles)
% hObject    handle to x0_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of x0_input as text
%        str2double(get(hObject,'String')) returns contents of x0_input as a double


% --- Executes during object creation, after setting all properties.
function x0_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to x0_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function y0_input_Callback(hObject, eventdata, handles)
% hObject    handle to y0_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of y0_input as text
%        str2double(get(hObject,'String')) returns contents of y0_input as a double


% --- Executes during object creation, after setting all properties.
function y0_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to y0_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in plot_button.
function plot_button_Callback(hObject, eventdata, handles)
% hObject    handle to plot_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Determine if the selected shape is 'circle' or 'rectangle'
radio_buttons = get(get(handles.shape_box,'SelectedObject'),'Tag');
plot_shape_result = 'NONE';
if strcmp(radio_buttons, 'radio_rect')
    plot_shape_result = 'rect';
end
if strcmp(radio_buttons, 'radio_circ')
    plot_shape_result = 'circ';
end

% Determine if the selected plot type as '3d', '2d', 'contour'
radio_buttons = get(get(handles.type_box,'SelectedObject'),'Tag');
plot_type_result = 'NONE';
if strcmp(radio_buttons, 'radio_3d')
    plot_type_result = '3d';
end
if strcmp(radio_buttons, 'radio_2d')
    plot_type_result = '2d';
end
if strcmp(radio_buttons, 'radio_contour')
    plot_type_result = 'contour';
end

% Grab all values from the input boxes
width = get(handles.width_input, 'String');
length = get(handles.length_input, 'String');
thickness = get(handles.thickness_input, 'String');
poisson = get(handles.poisson_input, 'String');
modulus = get(handles.modulus_input, 'String');
xres = get(handles.xres_input, 'String');
yres = get(handles.yres_input, 'String');
x0 = get(handles.x0_input, 'String');
y0 = get(handles.y0_input, 'String');
```

```
point = get(handles.point_input, 'String');
dist = get(handles.dist_input, 'String');

% Create the load array that will be passed into the function
load_cell_array = {};
load_count = 1;
if ~(point == 0)
    load_cell_array{load_count,1} = 'point';
    load_cell_array{load_count,2} = str2double(point);
    load_cell_array{load_count,3} = str2double(x0);
    load_cell_array{load_count,4} = str2double(y0);
    load_count = load_count + 1;
end
if ~(dist == 0)
    load_cell_array{load_count,1} = 'surface';
    load_cell_array{load_count,2} = str2double(dist);
    load_cell_array{load_count,3} = str2double(0);
    load_cell_array{load_count,4} = str2double(0);
end

disp(plot_type_result);

d = plate_deformation(plot_shape_result, load_cell_array, plot_type_result,
str2double(width), str2double(length), str2double(xres), str2double(yres),
str2double(thickness), str2double(poisson), str2double(modulus), handles.main_axes);
% Set maximum deflection string
set(handles.def_result, 'string', ['Maximum Deflection: ' num2str(d)]);

% --- Executes on button press in clear_button.
function clear_button_Callback(hObject, eventdata, handles)
% hObject    handle to clear_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Set every input box to blank
set(handles.width_input, 'string', '');
set(handles.length_input, 'string', '');
set(handles.thickness_input, 'string', '');
set(handles.poisson_input, 'string', '');
set(handles.modulus_input, 'string', '');
set(handles.xres_input, 'string', '');
set(handles.yres_input, 'string', '');
set(handles.x0_input, 'string', '');
set(handles.y0_input, 'string', '');

set(handles.point_input, 'string', '');
set(handles.dist_input, 'string', '');

set(handles.def_result, 'string', ['Maximum Deflection: ']);

cla reset;


function xres_input_Callback(hObject, eventdata, handles)
% hObject    handle to xres_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of xres_input as text
%        str2double(get(hObject,'String')) returns contents of xres_input as a double


% --- Executes during object creation, after setting all properties.
```

```
function xres_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xres_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function yres_input_Callback(hObject, eventdata, handles)
% hObject    handle to yres_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of yres_input as text
%        str2double(get(hObject,'String')) returns contents of yres_input as a double


% --- Executes during object creation, after setting all properties.
function yres_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to yres_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function dist_input_Callback(hObject, eventdata, handles)
% hObject    handle to dist_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dist_input as text
%        str2double(get(hObject,'String')) returns contents of dist_input as a double


% --- Executes during object creation, after setting all properties.
function dist_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dist_input (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

[End]