Individual project work – MATLAB Simple 2D Geometry Lab

Overview of project

The aim of this project is to implement a program in MATLAB which allows the user to draw a selection of simple shapes and for it to display their intersections. To do so required axes for plotting and ways for the user to interact with theses axes.

I have done this by introducing buttons for each shape, which when selected, allow the user to create the desired shape, similar to the tool system in many drawing and photo-editing software. Once the user has created the figure they wish to calculate for, they can select another button which will calculate all the intersections between each shape. Shapes can also be selected [1] which allows them to delete an individual shape as well as clear the axes entirely.

Current features include:

- Adding geometric shapes (line and circle)
- Removing shapes
- Calculating intersection points
- Saving and loading created figures

User Interface

The user interface layout is based off the idea used by photo-editing and manipulation software where everything is organised into docks/panels. I've chosen to set a 'toolbox' area which would contain any actions which allow the creation or manipulation of a singular shape or line. The top most panel contains buttons to allow the saving and loading of figures alongside zooming tools. Underneath that is the panel which contains changeable properties such as colour and the button to call the calculation of intersections.

This type of organisation of layout allows for better scalability which is the capability of a system to be enlarged to accommodate a growth in the functionality. This means continued development can be done to add more shapes, tools or changeable properties without heavily disrupting the layout. It is also laid out in such a way to simplify it for the user, as each function built into the program is clear.

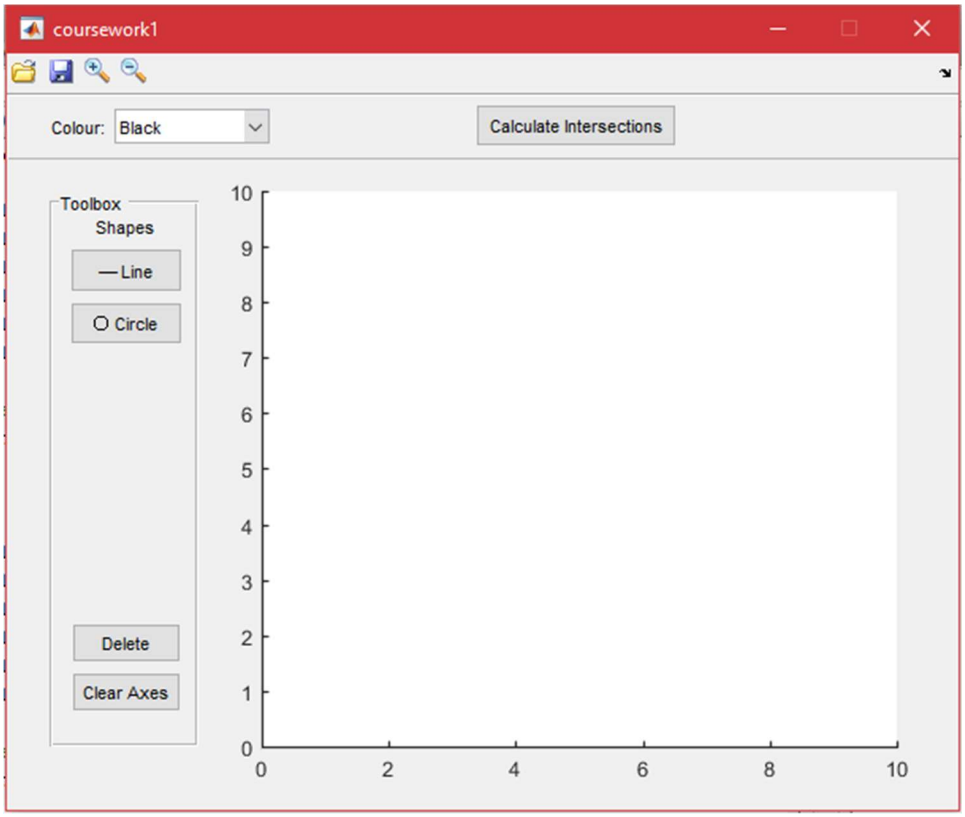

Figure 1: The initial figure upon opening

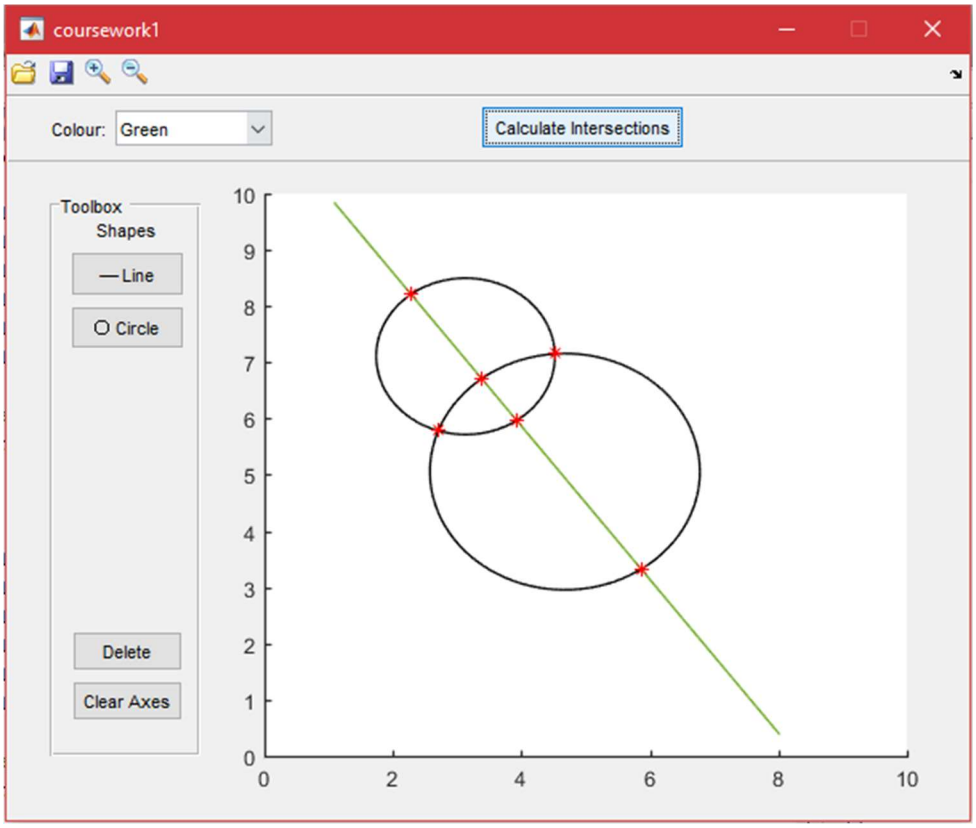

Figure 2: Placed circles and line and calculated intersections

Description of main functions and elements

Adding Shapes

There are currently two supported shapes, lines and circles. To create lines, the user selects the line tool which runs the call back for button *bntLine*. They are then prompted to select a point on the axes which is for the first end of the line. They then are required to select another. With each input the Cartesian coordinates (x and y) for both points are stored in associated variables. Using these coordinates the line is plotted using the pre-defined function *plot*. This line is also appended onto the handle for any plotted lines/shapes *handles.shapeHnds*. This is so the line may be manipulated later.

For circles we retrieve input for the point for the center and 'size' of the circle. The size point is the point which the user wants the circle to reach. Using both the radius is calculated using the *norm* function. To plot a circle using the *plot* function x and y coordinates for a closed curve forming a circle must be calculated.

This is done with the equations:

$$\mathrm{r} * \cos(\theta) + \mathrm{xcenter} = \mathrm{x}$$

$$\mathrm{r} * \sin(\theta) + \mathrm{ycenter} = \mathrm{y}$$

The circle is then plotted with the calculated x and y values and the handle for the circle is appended onto *handles.shapeHnds*.

Deleting Shapes

The delete button works when the user has selected a line/shape [1]. A shape is determined to be selected when the user clicks on it. When a point is selected the function *mouseDownCallback* is executed which checks to see if the selected point was within the axes and then checks there is a shape near the point which was selected. If there is a shape, then the index for it within the array handle of shapes *handles.shapeHnds* is stored into the handle *handles.Selected*. The line width for the selected shape is then increased to indicate that this shape has been selected.

When the delete button is selected it checks if there is selected shape by checking that *handles.Selected* is not empty. If it is not then the line handle at the index in *handles.shapeHnds* is deleted and set to empty. Then *handles.Selected* is also set to empty. This removes the selected shape from the axes.

Changing Colours

There a two types of colour changes, changing the default plot colour and changing the colour of a selected shape. When a colour is selected the call back executes and performs a check to see whether there is a selected shape or not. If there isn't then a switch case is run which sets the colour order of the axes to the selected colour. This means all future plots will be set to this colour.

If a shape is selected (*handles.Selected* contains an index) however a different switch case is run. The line colour of the selected shape is set the colour selected in the pop up menu.

To ensure the program keeps consistence with the information displayed back to the user, the value that is displayed in the unselected pop up menu *popColour* is always the colour set as the default colour. This is done with *handle.SelectedColOrder* which stores the value of the current colour selected as the colour order. Each time a line colour is changed the value of the pop up menu is set to the value stored in *handle.SelectedColOrder.*

Calculating Intersections

To calculate the intersections, the program makes use of the *handle.shapeHnds*. When the button btnIntersections is selected all the current shapes are used to calculate intersects in the call back.

The function has for loop which runs through all the shapes in the handle, storing their indexes in variable *i*. There is a second for loop which goes from the current *i + 1* to the length of the handle array of shapes (*handles.shapeHnds*). The indexes for this loop are stored in variable *j*. i and j represent the indexes of shapes for calculating the intersects between them.

Using the indexes, the handles are stored into two separate variables and then the *XData* and *YData* for both is retrieved with the *get* function.

The function makes use of the *polyxpoly* function to calculate intersects. The function returns the intersection point using x and y coordinates for both lines and outputs two column vectors containing the x and y coordinates for the points the first shape intersects segments of the second. Using the intersection coordinates a single point is plotted on the axes for each intersection.

Clear Axes

Goes through each handle for the shapes and deletes each handle without deleting the array handle entirely. The prevent a deleted object being left behind, each of the indexes in the array handle are also set to empty.

To clear the graphic objects from figure *cla* is used which removes all objects from the axes. Finally, all changes are saved.

Save and Load

The load function can be executed using the open button in the top most toolbar. The loading function works by allowing the user to open up *.fig* files within it, allowing them to select and open a saved figure files. The function *uigetfile* prompts the user to select a file and its filename is stored in a variable. Then it is used by the *openfig* function to load the figure. When doing so the previous open file is closed.

To save the currently opened figure, the user can select the save button. This runs the function *uiputfile* to prompt the user to select where the file is to be saved.

Code

```
% Conor Gibbs | 1524815

function varargout = coursework1(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @coursework1_OpeningFcn, ...
                   'gui_OutputFcn',  @coursework1_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 coursework1 is made visible.
function coursework1_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
set(hObject, 'WindowButtonDownFcn', @mouseDownCallback);

guidata(hObject, handles);

function varargout = coursework1_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

%-------------------REFERENCE--------------------
%Hayes, Geoff. (Online). https://uk.mathworks.com.
%check if any created lines are selected
function mouseDownCallback(figureHandle,varargin)
%distance constant to determine selected line
DisTheshold = 2;
%get the handles structure
handles = guidata(figureHandle);

%get position where selected
currentPoint = get(figureHandle, 'CurrentPoint');
x = currentPoint(1,1);
y = currentPoint(1,2);

%get axes position on the figure
axesPos = get(handles.axes1, 'Position');
minx = axesPos(1);
miny = axesPos(2);
maxx = minx + axesPos(3);
maxy = miny + axesPos(4);

%was the axes selected
if x>=minx && x<=maxx && y>=miny && y<=maxy
    %on a line
    if isfield(handles, 'shapeHnds')
```

```
        %get the position of the mouse down event within the axes
        currentPoint = get(handles.axes1, 'CurrentPoint');
        x = currentPoint(2,1);
        y = currentPoint(2,2);
        %we are going to use the x and y data for each line
        %and determine which one is closest to the selected point
        minDist = Inf;
        minHndIdx = 0;
        for k = 1:length(handles.shapeHnds)
            xData = get(handles.shapeHnds(k), 'XData');
            yData = get(handles.shapeHnds(k), 'YData');
            dist  = min((xData-x).^2+(yData-y).^2);
            if dist<minDist && dist<DisTheshold
                minHndIdx = k;
                minDist = dist;
            end
        end
        %if there is a line on axes near to selected point then
        %save the index of line in handles.shapeHnds
        if minHndIdx~=0
            handles.Selected = minHndIdx;
        else
            handles.Selected = [];
        end
        %change the line style of the selected object
        for k = 1:length(handles.shapeHnds)
            if k == minHndIdx
                set(handles.shapeHnds(k), 'LineWidth', 2);
            else
                set(handles.shapeHnds(k), 'LineWidth', 1);
            end
        end
        guidata(figureHandle, handles);
    end
end

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

%disable buttons to prevent errors
disableButtons(hObject, eventdata, handles);
%check if handle for lines/shapes exists
%if not then create the handle
if ~isfield(handles,'shapeHnds');
    handles.shapeHnds = [];
end

%take user input (points selected)
axis manual;
hold on;
[x y] = ginput(2);
x = [x(1),x(2)];
y = [y(1),y(2)];

%plot the line on graph using the x and y coordinates
l = plot(handles.axes1, x, y);
set(l, 'LineWidth', 1);
```

```
set(l, 'Tag', 'line');
hold off;

handles.shapeHnds = [handles.shapeHnds ; l];
%save changes and enable buttons
enableButtons(hObject, eventdata, handles);
guidata(hObject,handles);

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

%disable buttons to prevent errors
disableButtons(hObject, eventdata, handles);
%check if handle for lines/shapes exists
%if not then create the handle
if ~isfield(handles,'shapeHnds');
    handles.shapeHnds = [];
end

%take user input of center and radius
%calculate both using the x and y given
axis manual;
hold on;
[x y] = ginput(2);
r = norm([x(2) - x(1), y(2) - y(1)]);

theta = linspace(0,2*pi);
x = r*cos(theta) + x(1);
y = r*sin(theta) + y(1);

c = plot(x, y);
set(c, 'LineWidth', 1);
set(c, 'Tag', 'circle');
hold off;

handles.shapeHnds = [handles.shapeHnds ; c];

%save changes
enableButtons(hObject, eventdata, handles);
guidata(hObject,handles);


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

%go through each
for i = 1:length(handles.shapeHnds)
    for j = (i + 1):length(handles.shapeHnds)
        a = handles.shapeHnds(i);
        b = handles.shapeHnds(j);

        %get x and y values for both lines/circles
        x1 = get(a, 'XData');
```

```
        y1 = get(a, 'YData');
        x2 = get(b, 'XData');
        y2 = get(b, 'YData');

        %use polyxpoly function to calculate intersections
        %of any lines/polygon edges
        hold on;
        [xi, yi] = polyxpoly(x1,y1,x2,y2);
        plot(xi, yi, 'r*'); %plot points to graph
        hold off;
    end
end

% --- Executes on button press in btnDelete.
function btnDelete_Callback(hObject, eventdata, handles)
% hObject    handle to btnDelete (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if isfield(handles,'Selected') && ~isempty(handles.Selected)
    %delete selected line
    %use index of the currently selected line
    delete(handles.shapeHnds(handles.Selected));
    handles.shapeHnds(handles.Selected) = [];
    handles.Selected = [];

    %save changes
    guidata(hObject,handles);
end



% --- Executes on selection change in popColour.
function popColour_Callback(hObject, eventdata, handles)
% hObject    handle to popColour (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popColour
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
popColour

%change axes colour order
%next plot added follows the colour order
%if the field doesn't exist then define it
if ~isfield(handles, 'SelectedColOrder');
    handles.SelectedColOrder = 1;
end

%check whether it is a default colour change or selected line change
if ~isfield(handles, 'Selected') || isempty(handles.Selected)
    %for each default colour, a handle is kept to remember the selected
    switch get(handles.popColour, 'Value')
        case 1
            set(handles.axes1, 'ColorOrder', [0 0 0]);
            handles.SelectedColOrder = 1;
        case 2
            set(handles.axes1, 'ColorOrder', [0 0.4470 0.7410]); %blue
            handles.SelectedColOrder = 2;
        case 3
```

```
            set(handles.axes1, 'ColorOrder', [0.8500 0.3250 0.0980]);
%orange
            handles.SelectedColOrder = 3;
        case 4
            set(handles.axes1, 'ColorOrder', [0.9290 0.6940 0.1250]);
%yellow
            handles.SelectedColOrder = 4;
        case 5
            set(handles.axes1, 'ColorOrder', [0.4940 0.1840 0.5560]);
%purple
            handles.SelectedColOrder = 5;
        case 6
            set(handles.axes1, 'ColorOrder', [0.4660 0.6740 0.1880]);
%green
            handles.SelectedColOrder = 6;
    end

    %save changes
    guidata(hObject,handles);
else
    %change line colour to selected
    %and set value of popup to the selected default colour
    switch get(handles.popColour, 'Value')
        case 1
            set(handles.shapeHnds(handles.Selected), 'Color', [0 0 0]);
            set(handles.popColour, 'Value', handles.SelectedColOrder)
        case 2
            set(handles.shapeHnds(handles.Selected), 'Color', [0 0.4470
0.7410]) %blue
            set(handles.popColour, 'Value', handles.SelectedColOrder)
        case 3
            set(handles.shapeHnds(handles.Selected), 'Color', [0.8500
0.3250 0.0980]) %orange
            set(handles.popColour, 'Value', handles.SelectedColOrder)
        case 4
            set(handles.shapeHnds(handles.Selected), 'Color', [0.9290
0.6940 0.1250]) %yellow
            set(handles.popColour, 'Value', handles.SelectedColOrder)
        case 5
            set(handles.shapeHnds(handles.Selected), 'Color', [0.4940
0.1840 0.5560]) %purple
            set(handles.popColour, 'Value', handles.SelectedColOrder)
        case 6
            set(handles.shapeHnds(handles.Selected), 'Color', [0.4660
0.6740 0.1880]) %green
            set(handles.popColour, 'Value', handles.SelectedColOrder)
    end
    set(handles.shapeHnds(handles.Selected), 'LineWidth', 1)
    handles.Selected = [];

    %save changes
    guidata(hObject,handles);
end

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

```
% Hint: popupmenu 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 btnClear.
function btnClear_Callback(hObject, eventdata, handles)
% hObject    handle to btnClear (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%remove each line in handles.shapeHnds
for k = 1:length(handles.shapeHnds);
    delete(handles.shapeHnds(1));
    handles.shapeHnds(1) = [];
end
cla; %clear axes

%save changes
guidata(hObject,handles);


% --------------------------------------------------------------------
function uipushtool2_ClickedCallback(hObject, eventdata, handles)
% hObject    handle to uipushtool2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[filename] = uiputfile('*.fig', 'Save Figure As');
savefig(filename);


% --------------------------------------------------------------------
function uipushtool1_ClickedCallback(hObject, eventdata, handles)
% hObject    handle to uipushtool1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

clear;
[filename] = uigetfile('*.fig', 'Open Figure');
close(gcf);
openfig(filename);

%disables all buttons and popups
function disableButtons(hObject, eventdata, handles)
set(handles.btnLine, 'Enable', 'off');
set(handles.btnCircle, 'Enable', 'off');
set(handles.btnIntersections, 'Enable', 'off');
set(handles.btnDelete, 'Enable', 'off');
set(handles.btnClear, 'Enable', 'off');
set(handles.popColour, 'Enable', 'off');

%save changes
guidata(hObject,handles);

%enables all buttons and popups
```

```
function enableButtons(hObject, eventdata, handles)
set(handles.btnLine, 'Enable', 'on');
set(handles.btnCircle, 'Enable', 'on');
set(handles.btnIntersections, 'Enable', 'on');
set(handles.btnDelete, 'Enable', 'on');
set(handles.btnClear, 'Enable', 'on');
set(handles.popColour, 'Enable', 'on');

%save changes
guidata(hObject,handles);
```

References

[1] Hayes, Geoff. (2015). Mathworks. *How can I select a graph by mouse clicking and delete it?* – MATLAB Central. [Online] Avaliable at: https://uk.mathworks.com/matlabcentral/answers/261400-how-can-i-select-a-graph-by-mouse-clicking-and-delete-it [Accessed: 30th November 2016]