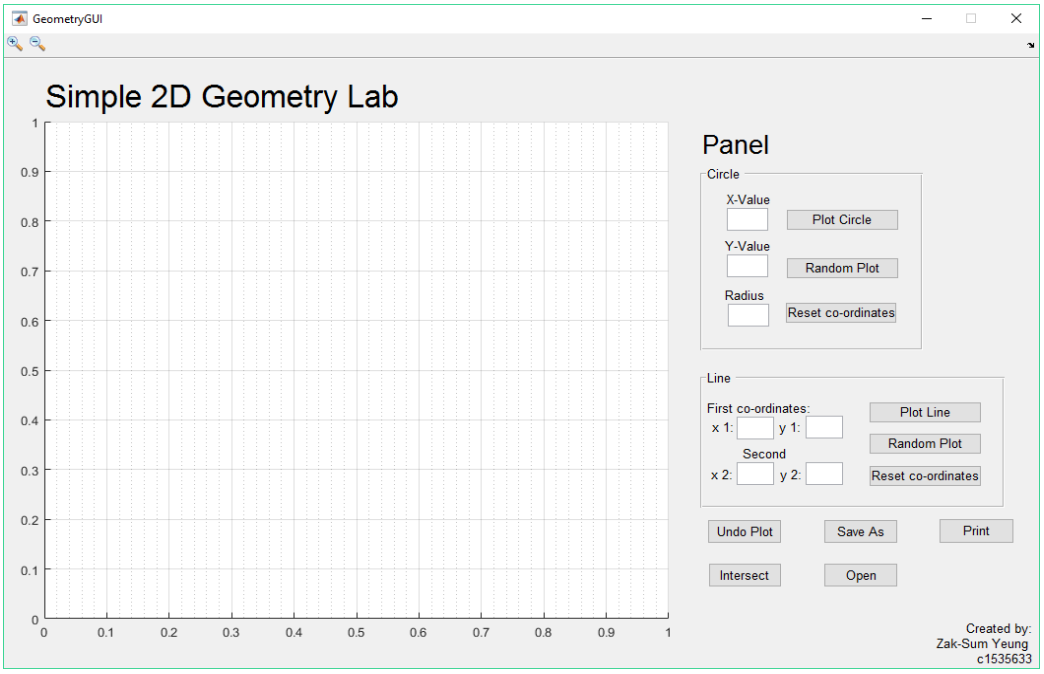
MATLAB Simple 2D Geometry Lab

This program is designed to allow the user such as school pupils to learn simple geometry and simple intersection points. The code that I have implemented enables users to input their desired ‘x’ or ‘y’ co-ordinates for a Line or the ‘x’ and ‘y’ co-ordinates for the centre of a circle as well as the length of the radius. Once the shapes have been plotted, the user would be able to find the intersection points of between ‘lines and lines’, ‘circles and lines’ or ‘circles and circles’, displayed by a ‘red point’ for each intersection.

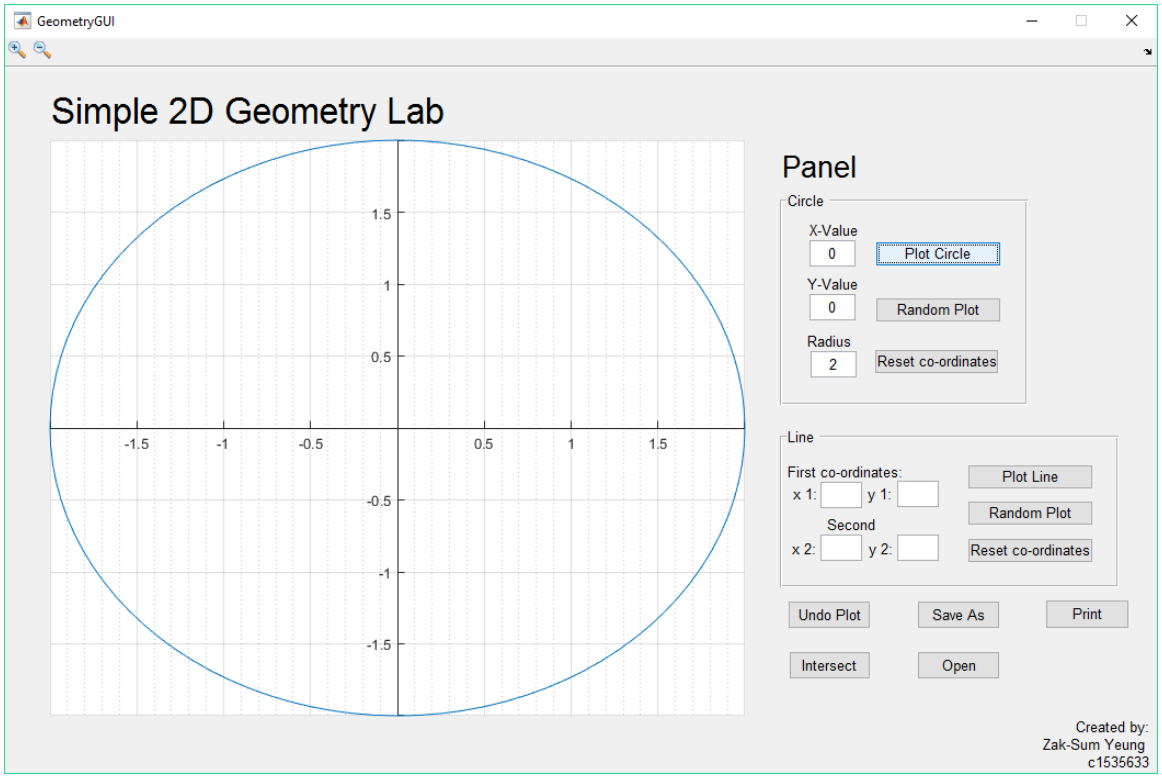
For the user to plot their shapes and lines, I created a Graphical User Interface (GUI) so that the users can view the shapes and the intersections that have been plotted. My GUI design is a simple design as suggested in the ‘title’ for school pupils at any level as well as teachers to use. I have inserted an Axes called ‘axes1’ where all the plotted lines and shapes would appear, to show the shapes, there are several panels for each shape or line, all containing textboxes, labels and buttons directing the users to plot their desired geometry.

As these are the basics functionalities of this program, I have also implemented several extra functionalities of my own. These include, clearing any input co-ordinates with a simple ‘Reset co-ordinates’ button, allowing the options for the users to ‘Print’ out their plotted shapes or lines including the intersection points with a simple ‘Print’ button by exporting the GUI into a ‘pdf’ file thus giving the user to print out their figures. Another extra functionality that I have included and comes in handy is enabling to plot ‘Random’ points of any shapes, ‘Circles’ or ‘Lines’, instead of plotting each co-ordinate for each shape which may take longer than expected for beginners trying to learn how ‘plotting’ works. Therefore, having this functionality would increase the speed of plotting shapes and line, but also, it could be used as a Demo by teachers showing the pupils the Aim of this program. The last two extra functionalities that I have also implemented and proves usefulness for the program to run is the ‘Save’ and ‘Open’ functions which allows the user Save their previous work and open it back up in the future.

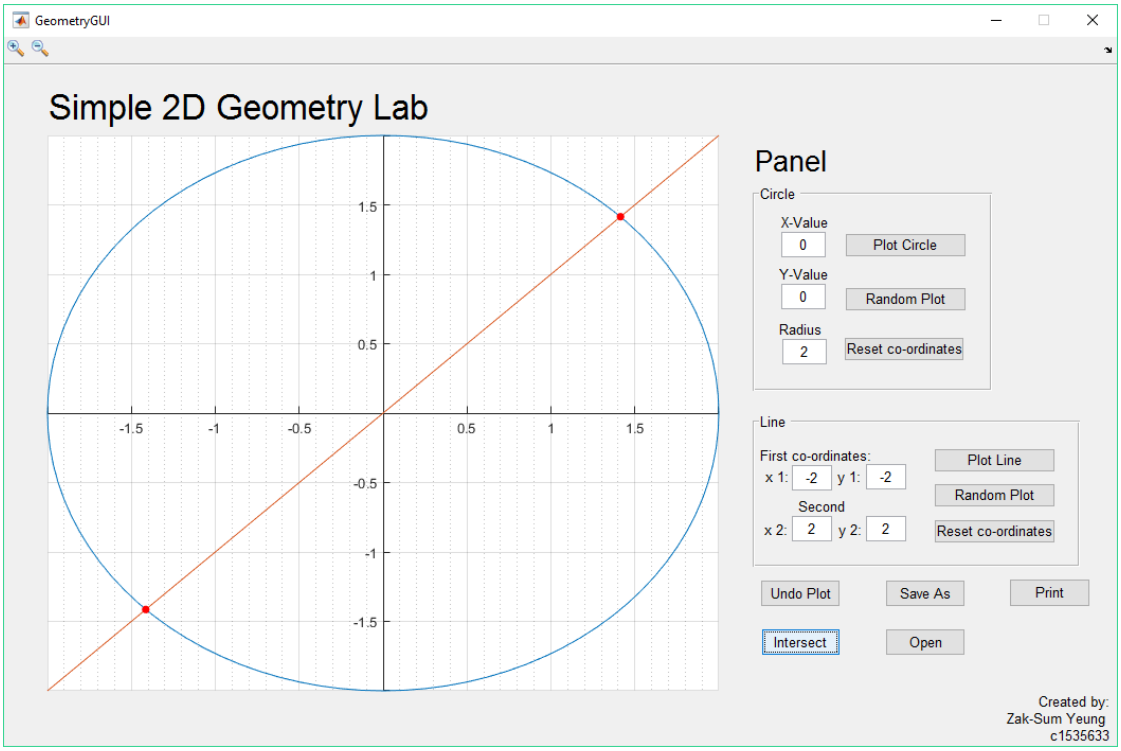
Below is an example of the finished GUI:

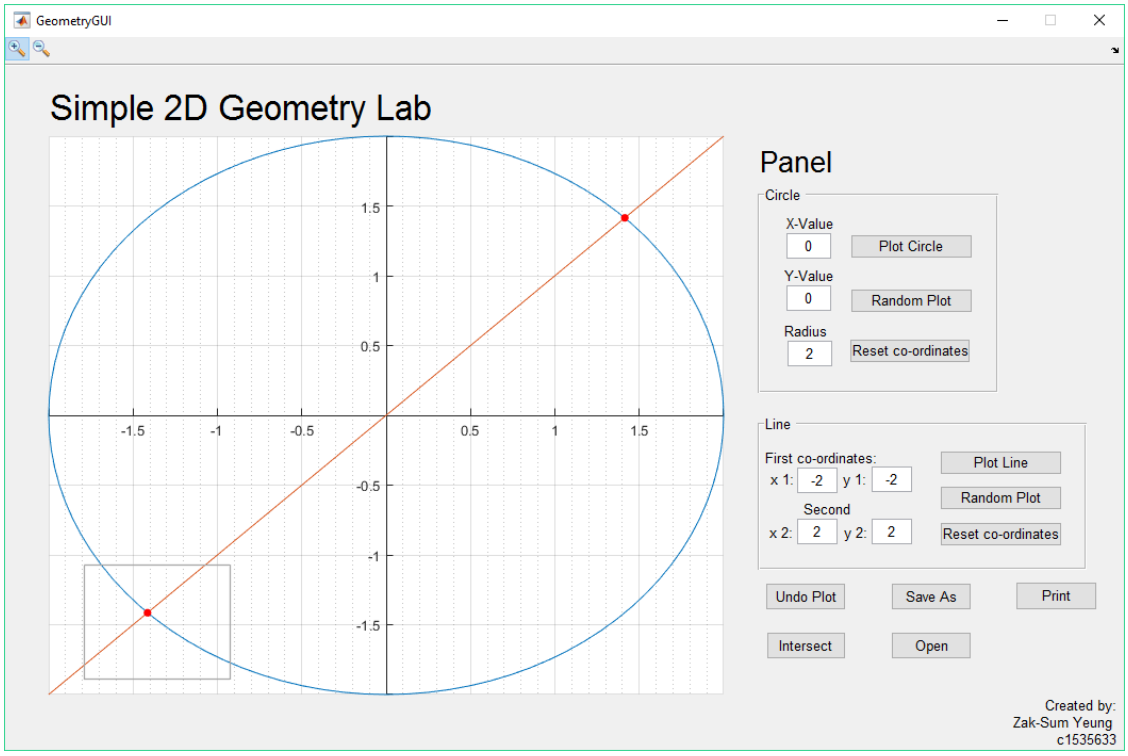


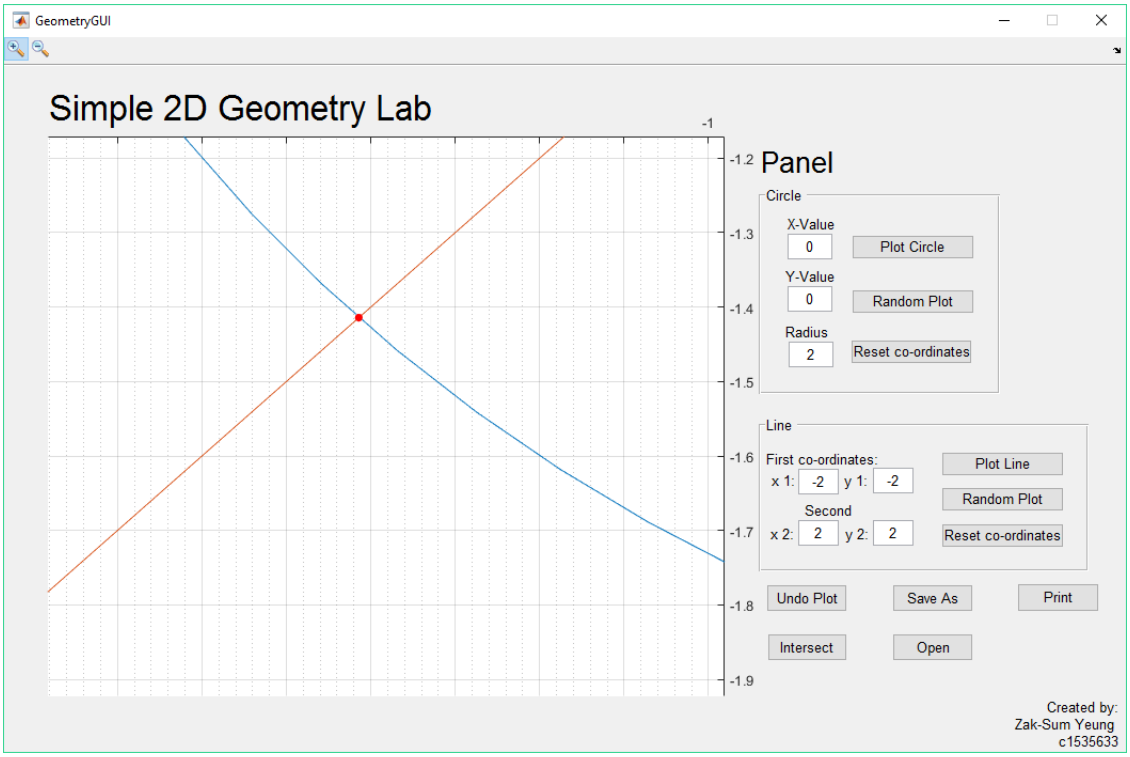
As mentioned above, I have included a Basic feature where the program allows the user to plot Circles and Lines. For Circle, I have included three simple 'edit boxes' which gives the user the choice to input their 'x co-ordinate', 'y co-ordinate' of the centre of the circle and the radius, which can plot a circle on 'axes1'. There are labels attached to each 'edit box' which can direct the user to the correct places thus avoiding confusions on which values have been entered. For lines, I have made four 'edit boxes' in total, one set of co-ordinates for the first point of a line and another set for the last point of a line. Both sets of co-ordinates have been labelled to every 'edit box' giving users simple instructions to where they should input their values. Both Lines and Circles Panels allow multiple inputs to 'edit boxes' giving users a choice to create more lines and circles on the same axes. To Intersect lines and circles, there is a 'Intersection' button which when clicked, it would plot 'red circles' wherever an intersection is located which could either be between 'lines and lines', 'circles and lines' or 'circles and circles'. The last basic functionality of this program is the 'Undo Plot' button, where when clicked, it would clear the last plotted line/circle/intersection point depending on what the user did last to their program. In the Axes 'axes1', you can see that the it displays a lined grid on the graph with the origin at the bottom left as no points have been plotted. However, when the user decides to plot a 'circle' or a 'line', the axes would automatically move to show the whole shape from start to end. Below is an example where the user inputted '0', '0', '2' for the 'x', 'y' centre values of a Circle and the length of the 'Radius' respectively:



Another feature that you may have noticed is the 'Zoom in' and 'Zoom out' tools at the top left hand side of the GUI. These are extra functionalities that are included, however they are in the GUI by default, meaning that I did not program these to work with my code. Although, as this is a GUI for plotting points and finding intersection points, I allowed the use of 'Zoom in' and 'Zoom out' enabling easier access if there was a large amount of intersections at a similar point. Below is an example of the default MATLAB 'Zoom in' and 'Zoom out' functionality:


GeometryGUI
Simple 2D Geometry Lab
Panel
Circle
X-Value
0
Plot Circle
Y-Value
0
Random Plot
Radius
2
Reset co-ordinates
Line
First co-ordinates:
x 1: -2 y 1: -2
Second
x 2: 2 y 2: 2
Plot Line
Random Plot
Reset co-ordinates
Undo Plot
Save As
Print
Intersect
Open
Created by:
Zak-Sum Yeung
c1535633


GeometryGUI
Simple 2D Geometry Lab
Panel
Circle
X-Value
0
Plot Circle
Y-Value
0
Random Plot
Radius
2
Reset co-ordinates
Line
First co-ordinates:
x 1: -2 y 1: -2
Second
x 2: 2 y 2: 2
Plot Line
Random Plot
Reset co-ordinates
Undo Plot
Save As
Print
Intersect
Open
Created by:
Zak-Sum Yeung
c1535633



Algorithmic Description:

Basic Features:

In the GeometryGUI output function, I created 2 empty handles, 'handles.lines' and 'handles.circle' where it would store values when called in from other functions. I have also included error dialogue boxes to each 'edit boxes' such as 'x_input', 'y_input', 'r_input', 'x1_input', 'x2_input', 'y1_input' and 'y2_input' where if the user inputs a string character into a non-valid string 'edit box', an error dialogue box would pop up informing the user, 'Input must be a number' and changing the input to '0'.

Function 'btn_Circle' allows the user to plot the centre co-ordinates of a circle as well as the radius. When the user inputs a number into a 'string' edit box, it gets stored into a variable where the variables 'x', 'y', 'r' would link to the MATLAB file 'circle.m' allowing the inputs to be calculated and passing it back to the GUI 'axes1' to draw the Circle specified by the user.

In the function 'btn_Line', it has a similar concept to the function 'btn_Circle', where whenever the user inputs the values into the 'edit boxes', 'x1_input', 'y1_input', 'x2_input' and 'y2_input', it gets the values and passes it through the MATLAB code 'lines.m', where it stores the input value and handles the values as a matrix and plots the input number onto 'axes1'.

The aim of btn_clear was to clear all values and any input matrix values when the button is pressed. However, there was a problem when inputting my values, where I wasn't able to clear the saved values, therefore I decided that I should make an undo button to undo the users previous selection/input.

btn_Intersect consists of 3 main for loops where it goes through all elements and finds the intersection. The first loop compares each handles matrix to make sure that they do not intersect

with each other, if they do intersect with each other, then it would plot a red circle. I have used 'circles_imp_int_2d' which was provided in the Geometric toolbox by the lecturer. This function takes the radius and centre 'x' and y' co-ordinates and returns the number of intersections which are plotted onto the axis.

I have also used the 'linesegintersect' function provided in the Geometric Tool Box by the lecturer which calculates the intersection between lines. It returns the number of intersections found between the lines and the intersection points.

Source code:

GeometryGUI.m

```
function varargout = GeometryGUI(varargin)
% GEOMETRYGUI MATLAB code for GeometryGUI.fig
%      GEOMETRYGUI, by itself, creates a new GEOMETRYGUI or raises the
existing
%      singleton*.
%
%      H = GEOMETRYGUI returns the handle to a new GEOMETRYGUI or the
handle to
%      the existing singleton*.
%
%      GEOMETRYGUI('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in GEOMETRYGUI.M with the given input
arguments.
%
%      GEOMETRYGUI('Property','Value',...) creates a new GEOMETRYGUI or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before GeometryGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to GeometryGUI_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 GeometryGUI

% Last Modified by GUIDE v2.5 09-Dec-2016 20:33:25

% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GeometryGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @GeometryGUI_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 GeometryGUI is made visible.
function GeometryGUI_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 GeometryGUI (see VARARGIN)

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

% Update handles structure
guidata(hObject, handles);

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

clear all; clc;
% --- Outputs from this function are returned to the command line.
function varargout = GeometryGUI_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;
handles.lines = [];
handles.circle = [];
guidata(hObject, handles);


function x_input_Callback(hObject, eventdata, handles)
% hObject    handle to x_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 x_input as text
%        str2double(get(hObject,'String')) returns contents of x_input as a
double
x_input = str2double(get(hObject, 'String'));
if isnan(x_input)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% --- Executes during object creation, after setting all properties.
function x_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to x_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 y_input_Callback(hObject, eventdata, handles)
% hObject    handle to y_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 y_input as text
%        str2double(get(hObject,'String')) returns contents of y_input as a
double
y_input = str2double(get(hObject, 'String'));
if isnan(y_input)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% --- Executes during object creation, after setting all properties.
function y_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to y_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 r_input_Callback(hObject, eventdata, handles)
% hObject    handle to r_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 r_input as text
%        str2double(get(hObject,'String')) returns contents of r_input as a
double
r_input = str2double(get(hObject, 'String'));
if isnan(r_input)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% --- Executes during object creation, after setting all properties.
function r_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to r_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 btn_Circle.
function btn_Circle_Callback(hObject, eventdata, handles)
% hObject    handle to btn_Circle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axis(handles.axes1);
x=str2double(get(handles.x_input,'String'));
y=str2double(get(handles.y_input,'String'));
r=str2double(get(handles.r_input,'String'));
circle(x,y,r);
handles.circle = [x,y,r; handles.circle];
        if isnan(handles.circle)
            errordlg('No Circles have been plotted');
        elseif isnan(x)
            errordlg('No x co-ordinate has been plotted');
        elseif isnan(y)
            errordlg('No y co-ordinate has been plotted');
        elseif isnan(r)
            errordlg('No Radius has been plotted');
        end
guidata(hObject, handles);
%disp(handles.circle);



function x1_input_Callback(hObject, eventdata, handles)
% hObject    handle to x1_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 x1_input as text
%        str2double(get(hObject,'String')) returns contents of x1_input as
a double
x1_input = str2double(get(hObject, 'String'));
if isnan(x1_input)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% --- Executes during object creation, after setting all properties.
function x1_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to x1_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 x2_input_Callback(hObject, eventdata, handles)
% hObject    handle to x2_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 x2_input as text
%        str2double(get(hObject,'String')) returns contents of x2_input as
a double
x2_input = str2double(get(hObject, 'String'));
if isnan(x2_input)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% --- Executes during object creation, after setting all properties.
function x2_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to x2_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 y1_input_Callback(hObject, eventdata, handles)
% hObject    handle to y1_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 y1_input as text
%        str2double(get(hObject,'String')) returns contents of y1_input as
a double
y1_input = str2double(get(hObject, 'String'));
if isnan(y1_input)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% --- Executes during object creation, after setting all properties.
function y1_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to y1_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 y2_input_Callback(hObject, eventdata, handles)
% hObject    handle to y2_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 y2_input as text
%        str2double(get(hObject,'String')) returns contents of y2_input as
a double
y2_input = str2double(get(hObject, 'String'));
if isnan(y2_input)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% --- Executes during object creation, after setting all properties.
function y2_input_CreateFcn(hObject, eventdata, handles)
% hObject    handle to y2_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 btn_Line.
function btn_Line_Callback(hObject, eventdata, handles)
% hObject    handle to btn_Line (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axis(handles.axes1);
% global x1
% global y1
x1=str2double(get(handles.x1_input,'String'));
y1=str2double(get(handles.y1_input,'String'));
x2=str2double(get(handles.x2_input,'String'));
y2=str2double(get(handles.y2_input,'String'));
lines(x1,y1,x2,y2);
handles.lines = [x1,y1,x2,y2; handles.lines];
disp(handles.lines);
guidata(hObject, handles);
        if isnan(handles.lines)
            errordlg('No Lines have been plotted');
        elseif isnan(x1)
            errordlg('No First x co-ordinate has been plotted');
        elseif isnan(y1)
            errordlg('No First y co-ordinate has been plotted');
        elseif isnan(x2)
            errordlg('No Second x co-ordinate has been plotted');
        elseif isnan(y2)
            errordlg('No Second y co-ordinate has been plotted');
        end
```

```
% --- Executes on button press in btn_clear.
function btn_clear_Callback(hObject, eventdata, handles)
% hObject    handle to btn_clear (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%
resetaxis = get(gca, 'children');
delete(resetaxis(1));


% --- Executes on button press in btn_Intersect.
function btn_Intersect_Callback(hObject, eventdata, handles)
 for m =1: size(handles.circle)
     center = handles.circle(m,1:2);
     radius = handles.circle(m,3);
    for i = m+1: size(handles.circle)
        center2 = handles.circle(i,1:2);
        radius2 = handles.circle(i,3);
        [ num_int, p ] = circles_imp_int_2d ( radius, center, radius2,
center2 )
        hold on;
        plot(p(1,:),p(2,:),'or','MarkerSize',5,'MarkerFaceColor','r');
    end
 end

for n =1: size(handles.lines)
    x1 = handles.lines(n,1:2);
    y1 = handles.lines(n,3:4);
   for j = n+1: size(handles.lines)
    x2 = handles.lines(j,1:2);
    y2 = handles.lines(j,3:4);
    Z = lineseg intersect(x1, y1, x2, y2);
    hold on;
    if Z > 0
    plot(Z(1,:),Z(2,:),'or','MarkerSize',5,'MarkerFaceColor','r');
    end
   end
end
  for o=1: size(handles.lines)
      x_1 = handles.lines(o,1:2);
      y_1 = handles.lines(o,3:4);
      [f,g,x0,y0] = line_exp2par_2d(x_1,y_1);
      for z=1: size(handles.circle)
          center = handles.circle(z,1:2);
          radius = handles.circle(z,3);
          [num_int ,v ] = circle_imp_line_par_int_2d(radius,center, x0,y0,
f,g);
          hold on;
            if num_int > 1
            preX1=handles.lines(o,1);
            preY1=handles.lines(o,2);
            preX2=handles.lines(o,3);
            preY2=handles.lines(o,4);

            minXPnt=min(preX1,preX2);
            minYPnt=min(preY1,preY2);
            maxXPnt=max(preX1,preX2);
            maxYPnt=max(preY1,preY2);
            hold on;
            if v >= minXPnt & v >= minYPnt & v <= maxXPnt & v <= maxYPnt
```

```
                plot(v(1,:),
v(2,:),'or','MarkerSize',5,'MarkerFaceColor','r');
            end
            if minXPnt <= v(1,1) & v(1,1) <= maxXPnt & minYPnt <= v(2,1) &
v(2,1) <= maxYPnt

plot(v(1,1),v(2,1),'or','MarkerSize',5,'MarkerFaceColor','r');
            elseif minXPnt <= v(1,2) & v(1,2) <= maxXPnt & minYPnt <=
v(2,2) & v(2,2) <= maxYPnt
                plot(v(1,2),
v(2,2),'or','MarkerSize',5,'MarkerFaceColor','r');
            end
            end
      end
  end

% --- Executes on button press in btn_reset_coord.
function btn_reset_coord_Callback(hObject, eventdata, handles)
% hObject    handle to btn_reset_coord (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.x1_input,'String','');
set(handles.x2_input,'String','');
set(handles.y1_input,'String','');
set(handles.y2_input,'String','');


% --- Executes on button press in btn_reset_circ.
function btn_reset_circ_Callback(hObject, eventdata, handles)
% hObject    handle to btn_reset_circ (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.x_input,'String','');
set(handles.y_input,'String','');
set(handles.r_input,'String','');


% --- Executes on button press in btn_Rand_Lines.
function btn_Rand_Lines_Callback(hObject, eventdata, handles)
% hObject    handle to btn_Rand_Lines (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
rx1 = round(rand(1)*500)
rx2 = round(rand(1)*500)
ry1 = round(rand(1)*500)
ry2 = round(rand(1)*500)
set(handles.x1_input,'String',rx1);
set(handles.x2_input,'String',rx2);
set(handles.y1_input,'String',ry1);
set(handles.y2_input,'String',ry2);


% --- Executes on button press in btn_Rand_Circle.
function btn_Rand_Circle_Callback(hObject, eventdata, handles)
% hObject    handle to btn_Rand_Circle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
rx = round(rand(1)*500)
ry = round(rand(1)*500)
rr = round(rand(1)*500)
```

```
set(handles.x_input,'String',rx);
set(handles.y_input,'String',ry);
set(handles.r_input,'String',rr);


% --- Executes on button press in btn_Save.
function btn_Save_Callback(hObject, eventdata, handles)
% hObject    handle to btn_Save (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
input = ('Please enter your filename: ')
title = 'Save As';
filename = inputdlg(input,title,[1 25]);
saveas(gcf, filename{1}, 'fig');

% --- Executes on button press in btn_Open.
function btn_Open_Callback(hObject, eventdata, handles)
% hObject    handle to btn_Open (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
input = ('Please enter your file name: ')
title = 'Open New (.fig)';
filename = inputdlg(input,title,[1 25]);
close(gcf);
openfig(strcat(filename{1},'.fig'), 'reuse');


% --- Executes on button press in btn_Print.
function btn_Print_Callback(hObject, eventdata, handles)
% hObject    handle to btn_Print (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    s = get(handles.axes1, 'Parent')
    print(s)
```

Lines.m

```
function l = lines(x1,y1,x2,y2)
hold on

x=[x1,x2];
y=[y1,y2];

l = plot(x,y);

hold off
```

Circle.m

```
function h = circle(x,y,r)
hold on

th = 0:pi/50:2*pi;

xunit = r * cos(th) + x;

yunit = r * sin(th) + y;
```

```
h = plot(xunit, yunit);

hold off
```

circle_imp_line_par_int_2d.m

```
function [ num_int, p ] = circle_imp_line_par_int_2d ( r, center, x0, y0,
f, g )

%% CIRCLE_IMP_LINE_PAR_INT_2D: ( implicit circle, parametric line )
intersection in 2D.
%
%  Discussion:
%
%    An implicit circle in 2D satisfies the equation:
%
%      ( X - CENTER(1) )**2 + ( Y - CENTER(2) )**2 = R**2
%
%    The parametric form of a line in 2D is:
%
%      X = X0 + F * T
%      Y = Y0 + G * T
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    31 January 2005
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, real R, the radius of the circle.
%
%    Input, real CENTER(2), the center of the circle.
%
%    Input, real F, G, X0, Y0, the parametric parameters of
%    the line.
%
%    Output, integer NUM_INT, the number of intersecting points found.
%    NUM_INT will be 0, 1 or 2.
%
%    Output, real P(2,NUM_INT), the intersecting points.
%
  dim_num = 2;
  p = [];

  root = r * r * ( f * f + g * g ) - ( f * ( center(2) - y0 ) ...
    - g * ( center(1) - x0 ) ).^2;

  if ( root < 0.0 )
```

```
    num_int = 0;

  elseif ( root == 0.0 )

    num_int = 1;

    t = ( f * ( center(1) - x0 ) + g * ( center(2) - y0 ) ) / ( f * f + g *
g );
    p(1,1) = x0 + f * t;
    p(2,1) = y0 + g * t;

  elseif ( 0.0 < root )

    num_int = 2;

    t = ( ( f * ( center(1) - x0 ) + g * ( center(2) - y0 ) ) ...
      - sqrt ( root ) ) / ( f * f + g * g );

    p(1,1) = x0 + f * t;
    p(2,1) = y0 + g * t;

    t = ( ( f * ( center(1) - x0 ) + g * ( center(2) - y0 ) ) ...
      + sqrt ( root ) ) / ( f * f + g * g );

    p(1,2) = x0 + f * t;
    p(2,2) = y0 + g * t;

  end

  return
end
```

circles_imp_int_2d.m

```
function [ num_int, p ] = circles_imp_int_2d ( r1, center1, r2, center2 )

%% CIRCLES_IMP_INT_2D: finds the intersection of two implicit circles in
2D.
%
%  Discussion:
%
%    Two circles can intersect in 0, 1, 2 or infinitely many points.
%
%    The 0 and 2 intersection cases are numerically robust; the 1 and
%    infinite intersection cases are numerically fragile.  The routine
%    uses a tolerance to try to detect the 1 and infinite cases.
%
%    An implicit circle in 2D satisfies the equation:
%
%      ( X - CENTER(1) )**2 + ( Y - CENTER(2) )**2 = R**2
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
```

```
%  Modified:
%
%    24 February 2005
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, real R1, the radius of the first circle.
%
%    Input, real CENTER1(2), the center of the first circle.
%
%    Input, real R2, the radius of the second circle.
%
%    Input, real CENTER2(2), the center of the second circle.
%
%    Output, integer NUM_INT, the number of intersecting points found.
%    NUM_INT will be 0, 1, 2 or 3.  3 indicates that there are an infinite
%    number of intersection points.
%
%    Output, real P(2,2), if NUM_INT is 1 or 2,
%    the coordinates of the intersecting points.
%
  dim_num = 2;

  tol = r8_epsilon ( );

  p(1:dim_num,1:2) = 0.0;;
%
%  Take care of the case in which the circles have the same center.
%
  t1 = ( abs ( center1(1) - center2(1) ) + abs ( center1(2) -
center2(2) ) ) / 2.0;

  t2 = ( abs ( center1(1) ) + abs ( center2(1) ) + abs ( center1(2) ) + abs
( center2(2) ) + 1.0 ) / 5.0;

  if ( t1 <= tol * t2 )

    t1 = abs ( r1 - r2 );
    t2 = ( abs ( r1 ) + abs ( r2 ) + 1.0 ) / 3.0;

    if ( t1 <= tol * t2 )
      num_int = 3;
    else
      num_int = 0;
    end

    return

  end

  distsq = sum ( ( center1(1:2) - center2(1:2) ).^2 );

  root = 2.0 * ( r1 * r1 + r2 * r2 ) * distsq - distsq * distsq ...
    - ( r1 - r2 ) * ( r1 - r2 ) * ( r1 + r2 ) * ( r1 + r2 );
```

```
  if ( root < -tol )
    num_int = 0;
    return
  end

  sc1 = ( distsq - ( r2 * r2 - r1 * r1 ) ) / distsq;

  if ( root < tol )
    num_int = 1;
    p(1:dim_num,1) = center1(1:dim_num)' ...
      + 0.5 * sc1 * ( center2(1:dim_num) - center1(1:dim_num) )';
    return
  end

  sc2 = sqrt ( root ) / distsq;

  num_int = 2;

  p(1,1) = center1(1) + 0.5 * sc1 * ( center2(1) - center1(1) ) ...
                      - 0.5 * sc2 * ( center2(2) - center1(2) );
  p(2,1) = center1(2) + 0.5 * sc1 * ( center2(2) - center1(2) ) ...
                      + 0.5 * sc2 * ( center2(1) - center1(1) );

  p(1,2) = center1(1) + 0.5 * sc1 * ( center2(1) - center1(1) ) ...
                      + 0.5 * sc2 * ( center2(2) - center1(2) );
  p(2,2) = center1(2) + 0.5 * sc1 * ( center2(2) - center1(2) ) ...
                      - 0.5 * sc2 * ( center2(1) - center1(1) );

  return
end
```

line_exp2par_2d.m

```
function [ f, g, x0, y0 ] = line_exp2par_2d ( p1, p2 )

%% LINE_EXP2PAR_2D converts a line from explicit to parametric form in 2D.
%
%  Discussion:
%
%    The explicit form of a line in 2D is:
%
%      ( P1, P2 ) = ( (X1,Y1), (X2,Y2) ).
%
%    The parametric form of a line in 2D is:
%
%      X = X0 + F * T
%      Y = Y0 + G * T
%
%    We normalize by choosing F*F+G*G=1 and 0 <= F.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
```

```
%
%    20 July 2006
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, real P1(2), P2(2), two points on the line.
%
%    Output, real F, G, X0, Y0, the parametric parameters
%    of the line.
%
  x0 = p1(1);
  y0 = p1(2);

  f = p2(1) - p1(1);
  g = p2(2) - p1(2);

  norm = sqrt ( f * f + g * g );

  if ( norm ~= 0.0 )
    f = f / norm;
    g = g / norm;
  end

  if ( f < 0.0 )
    f = -f;
    g = -g;
  end

  return
end
```

line_par2imp_2d.m

```
function [ a, b, c ] = line_par2imp_2d ( f, g, x0, y0 )

%% LINE_PAR2IMP_2D converts a parametric line to implicit form in 2D.
%
%  Discussion:
%
%    The parametric form of a line in 2D is:
%
%      X = X0 + F * T
%      Y = Y0 + G * T
%
%    We normalize by choosing F*F+G*G=1 and 0 <= F.
%
%    The implicit form of a line in 2D is:
%
%      A * X + B * Y + C = 0
%
%  Licensing:
%
```

```
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    20 July 2006
%
%  Author:
%
%    John Burkardt
%
%  Reference:
%
%    Adrian Bowyer and John Woodwark,
%    A Programmer's Geometry,
%    Butterworths, 1983.
%
%  Parameters:
%
%    Input, real F, G, X0, Y0, the parametric line parameters.
%
%    Output, real A, B, C, the implicit line parameters.
%
  a = -g;
  b = f;
  c = g * x0 - f * y0;

  return
end
```

lines_imp_int_2d.m

```
function [ ival, p ] = lines_imp_int_2d ( a1, b1, c1, a2, b2, c2 )

%% LINES_IMP_INT_2D determines where two implicit lines intersect in 2D.
%
%  Discussion:
%
%    The implicit form of a line in 2D is:
%
%      A * X + B * Y + C = 0
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    12 May 2005
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, real A1, B1, C1, define the first line.
%    At least one of A1 and B1 must be nonzero.
%
%    Input, real A2, B2, C2, define the second line.
```

```
%    At least one of A2 and B2 must be nonzero.
%
%    Output, integer IVAL, reports on the intersection.
%
%    -1, both A1 and B1 were zero.
%    -2, both A2 and B2 were zero.
%     0, no intersection, the lines are parallel.
%     1, one intersection point, returned in X, Y.
%     2, infinitely many intersections, the lines are identical.
%
%    Output, real P(2), if IVAL = 1, then P is
%    the intersection point.  if IVAL = 2, then P is one of the
%    points of intersection.  Otherwise, P = [].
%
  dim_num = 2;
%
%  Refuse to handle degenerate lines.
%
  if ( a1 == 0.0 & b1 == 0.0 )
    ival = -1;
    p = [];
    return
  elseif ( a2 == 0.0 & b2 == 0.0 )
    ival = -2;
    p = [];
    return
  end
%
%  Set up and solve a linear system.
%
  a(1,1) = a1;
  a(1,2) = b1;
  a(1,3) = -c1;

  a(2,1) = a2;
  a(2,2) = b2;
  a(2,3) = -c2;

  [ a, info ] = r8mat_solve ( 2, 1, a );
%
%  If the inverse exists, then the lines intersect at the solution point.
%
  if ( info == 0 )

    ival = 1;
    p(1:dim_num) = a(1:dim_num,3);
%
%  If the inverse does not exist, then the lines are parallel
%  or coincident.  Check for parallelism by seeing if the
%  C entries are in the same ratio as the A or B entries.
%
  else

    ival = 0;
    p = [];

    if ( a1 == 0.0 )
      if ( b2 * c1 == c2 * b1 )
        ival = 2;
        p(1:dim_num) = [ 0.0, - c1 / b1 ];
```

```
      end
    else
      if ( a2 * c1 == c2 * a1 )
        ival = 2;
        if ( abs ( a1 ) < abs ( b1 ) )
          p(1:dim_num) = [ 0.0, - c1 / b1 ];
        else
          p(1:dim_num) = [ - c1 / a1, 0.0 ];
        end
      end
    end

  end

  return
end
```

lineseglintersect.m

```
function P = lineseglintersect(P1, P2, P3, P4)
P1 = P1';
P2 = P2';
P3 = P3';
P4 = P4';
% We assume P1, P2, P3 and P4 are column vectors
P1P2 = P2 - P1;
P3P4 = P4 - P3;
[a1 b1 c1] = line_par2imp_2d(P1P2(1), P1P2(2), P1(1), P1(2));
[a2 b2 c2] = line_par2imp_2d(P3P4(1), P3P4(2), P3(1), P3(2));
[ival, P] = lines_imp_int_2d (a1, b1, c1, a2, b2, c2);
P = P'; % convert from row vector to column vector
if ival ~= 1
P = [];
else
if dot(P - P1, P2 - P) < 0 || dot(P - P3, P4 - P) < 0
P = [];
end
end
end
```

r8_epsilon.m

```
function value = r8_epsilon ( dummy )

%**********************************************************************
***80
%
%% R8_EPSILON returns the R8 roundoff unit.
%
%  Discussion:
%
%    The roundoff unit is a number R which is a power of 2 with the
%    property that, to the precision of the computer's arithmetic,
%      1 < 1 + R
%    but
```

```
%      1 = ( 1 + R / 2 )
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    22 August 2004
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, double precision DUMMY, a dummy value.
%
%    Output, double precision VALUE, the roundoff unit.
%
  value = eps;

  return
end
```

r8mat_solve.m

```
function [ a, info ] = r8mat_solve ( n, nrhs, a )

%% R8MAT_SOLVE uses Gauss-Jordan elimination to solve an N by N linear
system.
%
%  Licensing:
%
%    This code is distributed under the GNU LGPL license.
%
%  Modified:
%
%    31 January 2005
%
%  Author:
%
%    John Burkardt
%
%  Parameters:
%
%    Input, integer N, the order of the matrix.
%
%    Input, integer NRHS, the number of right hand sides.  NRHS
%    must be at least 0.
%
%    Input, real A(N,N+NRHS), contains in rows and
%    columns 1 to N the coefficient matrix, and in columns N+1 through
%    N+NRHS, the right hand sides.
%
%    Output, real A(N,N+NRHS), the coefficient matrix
%    area has been destroyed, while the right hand sides have
```

```
%    been overwritten with the corresponding solutions.
%
%    Output, integer INFO, singularity flag.
%    0, the matrix was not singular, the solutions were computed;
%    J, factorization failed on step J, and the solutions could not
%    be computed.
%
  info = 0;

  for j = 1 : n
%
%  Choose a pivot row IPIVOT.
%
    ipivot = j;
    apivot = a(j,j);

    for i = j+1 : n
      if ( abs ( apivot ) < abs ( a(i,j) ) )
        apivot = a(i,j);
        ipivot = i;
      end
    end

    if ( apivot == 0.0 )
      info = j;
      return;
    end
%
%  Interchange.
%
    temp                = a(ipivot,1:n+nrhs);
    a(ipivot,1:n+nrhs) = a(j,     1:n+nrhs);
    a(j,     1:n+nrhs) = temp;
%
%  A(J,J) becomes 1.
%
    a(j,j) = 1.0;
    a(j,j+1:n+nrhs) = a(j,j+1:n+nrhs) / apivot;
%
%  A(I,J) becomes 0.
%
    for i = 1 : n

      if ( i ~= j )

        factor = a(i,j);
        a(i,j) = 0.0;
        a(i,j+1:n+nrhs) = a(i,j+1:n+nrhs) - factor * a(j,j+1:n+nrhs);

      end

    end

  end

  return
end
```