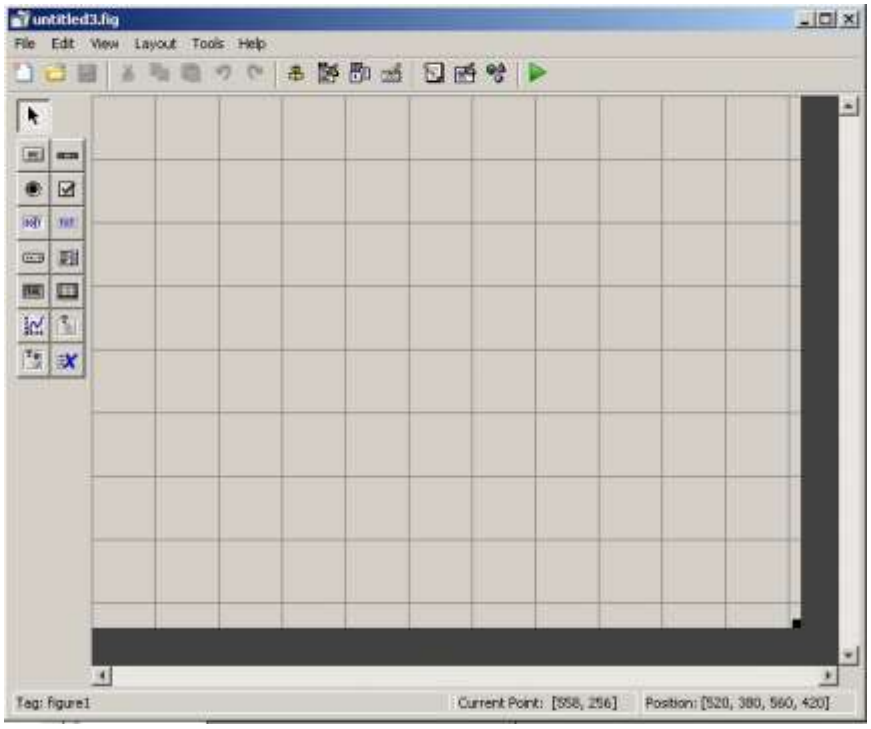
# Exercise 8

## Graphical User Interface (GUI)

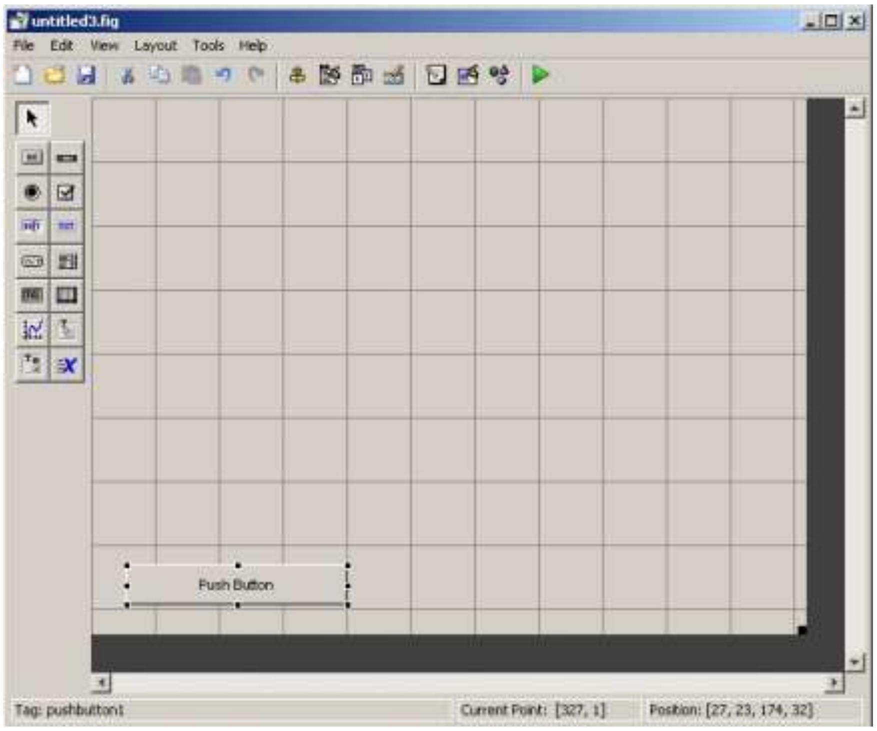
We will turn our focus to creating and using GUIs, which will be required for you in order to pass your projects in this course. A nice tool in Matlab is the GUIDE application. Start it up by typing in the command prompt:

```
>> guide        % Open the GUI Design Envrionment.
```

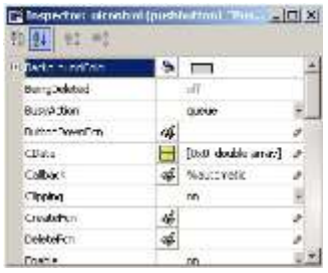
Select “blank GUI” in the options You should see something like this:


untitled3.fig
File Edit View Layout Tools Help
Tag: figure1
Current Point: [558, 256]
Position: [520, 380, 560, 420]

From this point, you can design your GUI to contain any of the standard items that you find in the left part of the window. Lets start by putting a regular push button in the GUI. This should be your result:



If you double click the newly created push button, its properties will be brought up in a new window called the “property inspector”:



| | |
|---|---|
| BackgroundColor | |
| BeingDeleted | off |
| BusyAction | queue |
| ButtonDownFcn | |
| CData | [0x0 double array] |
| Callback | %automatic |
| Clipping | on |
| CreateFcn | |
| DeleteFcn | |
| Enable | on |

Find the property “String”, and change it from “Push Button”, to say “Click Me!”. This will put this text on the button in the GUI. Let’s try to run this GUI, which you do by clicking the small icon on the top of GUIDE, that looks like a play button:

This will ask you to save the project as something, and you could pick something instructive, and not the name “untitled3.fig” (which is what is choosen here).

The GUI will be quite useless, as nothing happens when you push the button. Lets fix that now. Go back to GUIDE, and right click the button. Choose “View Callbacks -> Callback”. This will automatically open an m-file called “untitled3.m”, and you will inside this m-file see the following code part:

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

This function will be called every time the button is clicked. Right now the function is empty, which is why nothing happens when you click the button. Enter the line:

```
disp('Hello World, from my GUI');
```

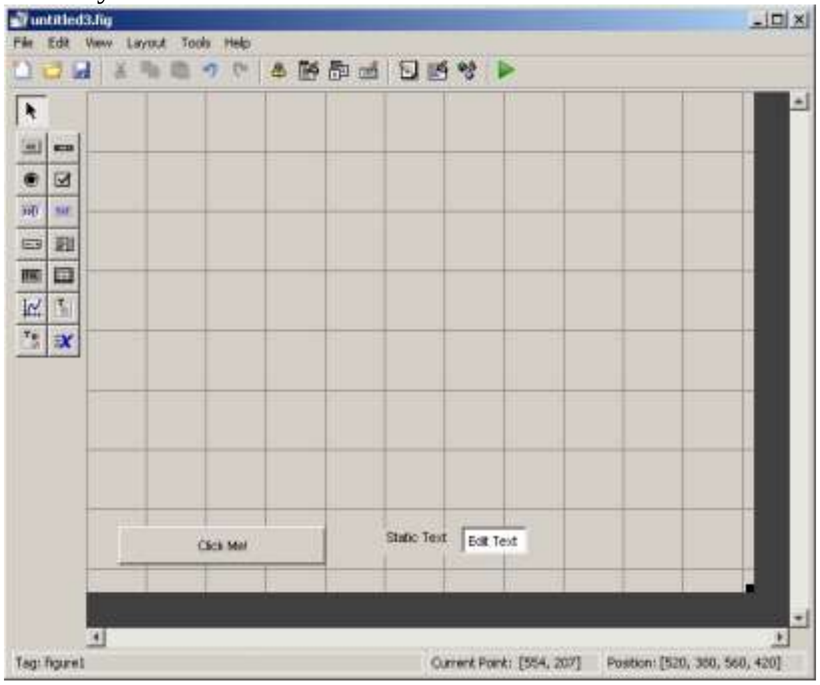
To be your entire function, save the m-file, then run your GUI again. Every time you click the button, you should now see the text being echoed in the command window. There are predefined dialog windows that are useful when we want to display messages to the user.  Go back to the "pushbutton1_Callback" and add the line:

```
msgbox('A little message for you','silly dialog');
```

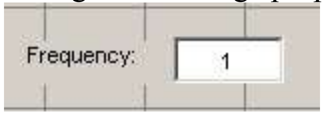
Several other useful built-in dialogs like this one exist. Try adding these lines to your callback function(it will cause a lot of dialogs of different kinds to pop up during button click):

```
warndlg('Watch up!','Warning !')
helpdlg('Click to get help !','Help')
questdlg('Is Matlab fun, ?','Question')
errordlg('Don't !','Error !')
```

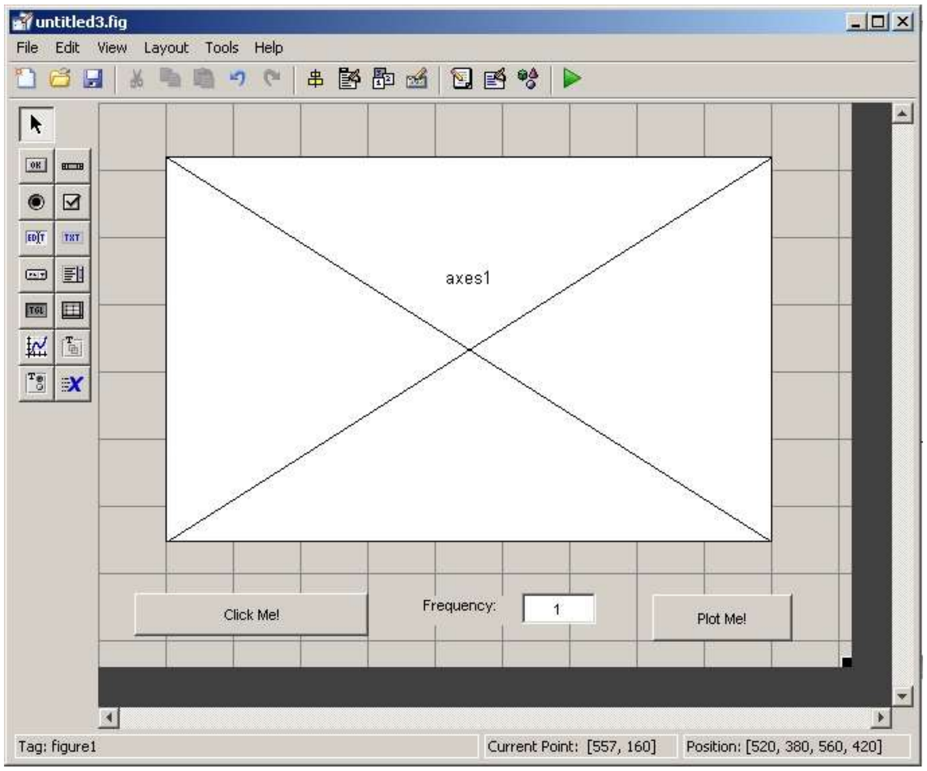
Next we will add some text boxes. There are two kinds "static text" and "edit text". Add one of each, so that your GUI looks like this:


untitled3.fig
File Edit View Layout Tools Help
Click Me!
Static Text
Edit Text
Tag: figure1
Current Point: [554, 207]
Position: [520, 380, 560, 420]

Change the "String" property of the text boxes to be "Frequency:" and "1" respectively:



Also, add a new push button with “String” set to “Draw Me”, and a big “axes object” so that your GUI looks like this:


untitled3.fig
File Edit View Layout Tools Help
axes1
Click Me!
Frequency:
1
Plot Me!
Tag: figure1
Current Point: [557, 160]
Position: [520, 380, 560, 420]

Now, what we are aiming for is to have a nice plot of a sinus wave in the newly created axes object. The sinus should have the frequency indicated in the edit box (1 rad/sec by default), and it should be drawn when we click the button “Plot Me!”

In order to do this, we must use handles to the two objects of interest:

1) the text box containing the value given by the user, and
2) the axes object, where we will draw the frequency curve

We can do this most easily by setting tags on these objects(this will become clear soon).
Now, double click the axes object in GUIDE, and in the property inspector, find the property called “Tag” (it has the tag “axes1” by default) change its value to “plotAreaForSinus”. Find the tag property for the text box as well (by default its called “edit1”) and change it to “freqNumber”.

After this, right click on the “Plot Me!” button, and select “View Callbacks -> Callback”.

Change the callback function so that it looks like this:

```
function pushbutton2_Callback(hObject, eventdata, handles)
%First, we get handles to the two objects of interest:
hAxes = handles.plotAreaForSinus;  % the axes to draw in
hTxt  = handles.freqNumber;        % the text box that has frequency

freqString = get(hTxt,'String');   % get the string value in the box
freqValue = str2num(freqString );  % convert the string to numerical
t = linspace(0,2*pi);              % interval for t
plot(hAxes, sin(freqValue*t));     % plot in the specified axes "hAxes"
```

When you run the GUI, after saving the m-file, you will now be able to put any value you like into the text box, and the corresponding frequency will be plotted.

## More on callback functions

The way that this works, is through the argument “handles” that is sent to the callback function. We can get any handles to any objects in our GUI through that one, if we know the “tag” of the object in question. We simply use the construction:

```
handles.someTag;
```

to get access to any and all handles. We did this to get a handle to both the axes and the text box in the example above. Once we have a handle to an object, we can use “get” and “set” with it. We can also use the “plot” function with a handle, which tells it specifically where to draw.

Callback functions are very useful, and are a recurring theme in many programming languages and environments. You have already had experience in the callback function for handling keyboard presses in previous exercises.

The final example we will dive into is the powerful topic of handling mouse clicks. Like the keyboard presses, this is done with a callback as well

### Mouse input, “ButtonDownFcn” vs “ginput”

We learnt fairly early on about the matlab function “ginput”. It is a very easy to use function, but it is limited. We have seen that with GUIDE we can click arbitrarily on any of the defined buttons, and get them to react by executing callback functions. We can do this for the general figure axes as well.

In GUIDE:
1. Create a new blank GUI, name it “tmp” and add an axes to it.
2. Double click it, bring up the properties of it. Find the properties “YLimMode” and “XLimMode”, set them both to “manual” (this prevents axis rescaling)
3. Right-click the axes object, select callbacks, and then click “ButtonDownFcn”. This will bring you to the callback function that will execute at any time you click the axis when the application is running.

On my system, this looks like this:

```
% --- Executes on mouse press over axes background.
function axes1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Change this function to the following:

```
function axes1_ButtonDownFcn(hObject, eventdata, handles)
pos=get(gca,'CurrentPoint');
x = pos(1,1);
```

```
y = pos(1,2);
disp(['you clicked on point:  (' num2str(x) ', ' num2str(y) ')' ])
```

This opens up for a lot of powerful interactive applications in Matlab. Lets try to keep track of several clicked positions. This requires us first to define variables that will store all previous clicks. The easiest way to do this is by defining some global variables. We have to make sure the variables are empty when we start the application, so open the “`tmp_OpeningFcn`” first. The “`OpeningFcn`” is the place where we put initialization code, and there we can define (and empty) the globals as follows:

```
% at the end of tmp_OpeningFcn:
global pX pY;
pX = [];
pY = [];
```

After that, we can go back to the callback function and modify it to the following:

```
function axes1_ButtonDownFcn(hObject, eventdata, handles)
global pX pY;

pos=get(gca,'CurrentPoint');
pX = [pX pos(1,1)];
pY = [pY pos(1,2)]   % echos all the y coordinates clicked
```

This will just echo the y coordinates of the clicked points back to the prompt. Naturally, we are now itching to do something more fun with the gathered clicks, but if we are not careful we will run into trouble.
Lets say we would like to interactively plot lines between each clicked point. Our first attempt at doing this could look as follows:

```
function axes1_ButtonDownFcn(hObject, eventdata, handles)
global pX pY;

pos=get(gca,'CurrentPoint');
pX = [pX pos(1,1)];
pY = [pY pos(1,2)]   % echos all the y coordinates clicked

plot(pX,pY);
```

(notice that above snippet still echos all the y coordinates back to the prompt). This will fail because of how the function “plot” works. “plot” is a high level function, meaning it likes to fiddle and change settings around as it pleases. One of the annoying things it does by default is to change all the callback functions of any axis where it is drawing. Therefore, the above snippet of code will only run once. As soon as “plot” is called, it makes sure that there is no more callbacks associated with the axis. We can change this behavior of plot through its many configurable options, or we can use a lower level function instead. Lower level here means… it keeps its fingers off the settings we have put on our axes. The function we can use for our purposes is called “line”, and it works very similar to “plot” for this application:

```
function axes1_ButtonDownFcn(hObject, eventdata, handles)
global pX pY;

pos=get(gca,'CurrentPoint');
pX = [pX pos(1,1)];
pY = [pY pos(1,2)]   % echos all the y coordinates clicked
```

```
line(pX,pY);
```

The above snippet works fairly well, but we can do a more effecient solution. We can create a graphics object, and choose to only update its data instead of creating new objects on each callback. We can do that by replacing our previous globals (pX, and pY) with a handle hLine. In the opening function:

```
% at the end of tmp_OpeningFcn:
global hLine;
hLine = [];
```

After which we could do a callback as follows:

```
function axes1_ButtonDownFcn(hObject, eventdata, handles)
global hLine;

pos=get(gca,'CurrentPoint');
x = pos(1,1);
y = pos(1,2);

if isempty(hLine) %this happens on the first click
    hLine = line(x,y);
else
    %get the previous clicks:
    pX = get(hLine,'XData');
    pY = get(hLine,'YData');
    % append new click (x,y):
    set(hLine,'XData',[pX x], 'YData',[pY y]);
end
```

In above solutions, we have used the global workspace, which we know is usually a dangerous and ugly way to work. Unfortunately, due to short-comings in how Matlab is designed, this is the best way to solve problems like this. The *officially* recommended way of solving these tasks (by Mathworks) is to use a function called “guidata”. This makes for very very ugly solutions, that have their own sources of bugs and difficult to read code. Using the “guidata” function will also make for less efficient programs, when the data amount is large. However, if you really wish to avoid the global workspace, when developing these kinds of applications, you can take a look at “guidata” (you have been warned).

## Tasks for Exercise 8

Solutions that do not fill the following requirements EXACTLY, PRECISELYAND TO THE LETTER will not be considered:

• Send your solutions by email to:

stefan.karlsson@hh.se

subject: Matlab, Exercise X, YourNames

• Send all the files that are requested, no more and no less, in one single zip file per exercise, with NO sub-folders in the zip file. All the files, for all the tasks should be bundled into one zip file.

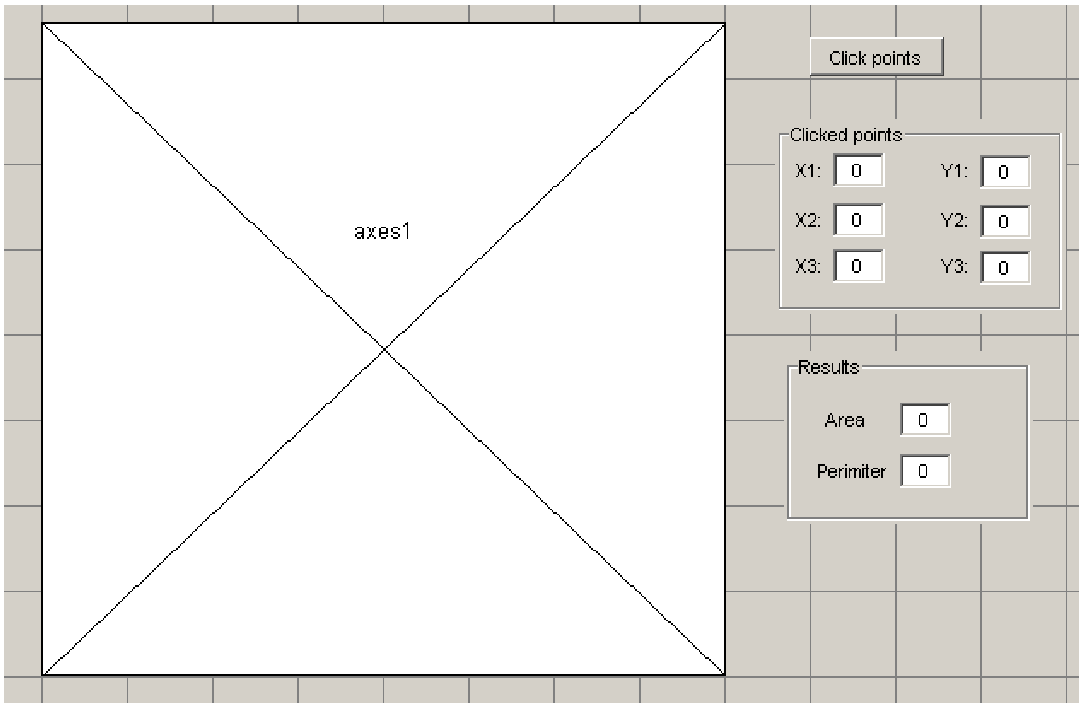
• Put the Names of the authors, in remarks, at the top of every m-file.

• Send the solutions within 2 weeks of every exercise session. That is, you have a two week deadline to hand it in.

• You will get 2 chances to send it in to me correctly.

## Task 1

Make an application using GUIDE, call it “e8_1”. It should have the following GUI:



It should have the same functionality as task 3, of exercise 1, which I remind you of here (you can look in the original pdf as well):

------ from previous pdf (E1) -----

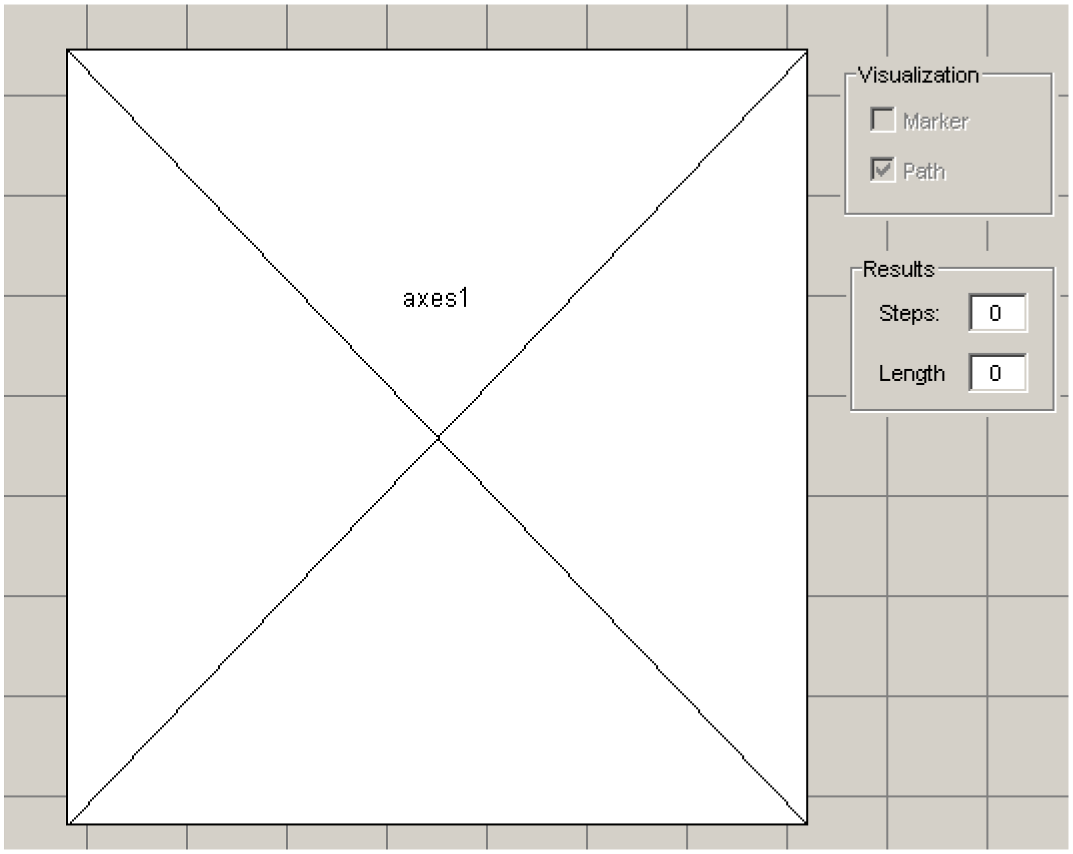
1. Let the user click 3 times in a figure, defining a triangle.
2. The area and perimeter of the triangle clicked should be displayed in a dialog box;
3. The triangle should be drawn in a figure.

-------------------------------------------------------------------

However, your application will NOT bring up a new figure, nor a dialog box. It will use the GUI instead. When the user clicks the button “click points”, the application allows you to click three times in the axes, the triangle will be displayed in the same axes. In the uipanel: “Clicked points”, the x and y coordinates of the clicked points should be displayed. In the uipanel: “Results”, the area and perimeter of the clicked triangle is displayed. Use “ginput” to solve this task.

## Task 2 (optional for grade 4)

Make an application using GUIDE, call it “e8_2”. It should have the following GUI:



The workings of this application is exemplified in the attached video file, and p-code is made available for you to run.

The application will track the positions you have clicked in the axes, and draw lines between them. The application will automatically update the uipanel “Results”, with both the number of times you clicked (“steps”) as well as the length of path your clicks define.

See next page for hints.

Hints:

- It is impossible to solve this using “ginput”, you need to use a callback function.
- Using global variables to keep track of the points clicked is the easiest (but ugly) way to solve this. In the function “`e82_OpeningFcn`”, you can put the lines:

  ```
  global pX pY hPath;
  pX    = [];
  pY    = [];
  hPath = [];
  ```

  Where hPath will be used for the graphics object handle for the line drawing, and pX and pY will be used to store the x and y coordinates clicked. Don’t forget to declare the global variable in each function you use them in after this.

- Don’t forget to set the XlimMode and YLimMode to “manual” for the axis
- The check boxes in uipanel “Visualizations”, should be deactivated on start (property “enable” set to off). Let them be activated (“enable” set to “on”) after the first mouse click.
- You need to code in 3 callback functions for this task: the axis (buttonDownFcn), and the two tickboxes