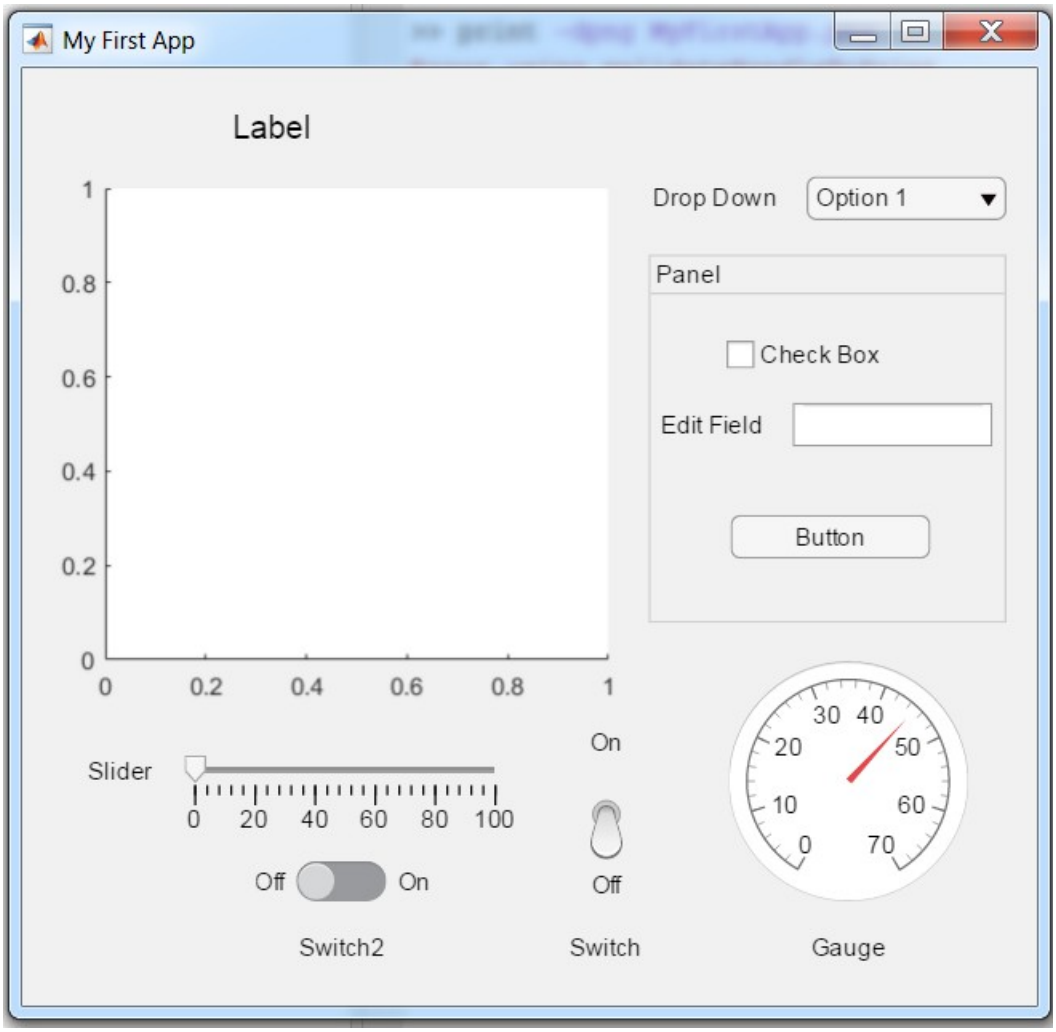
INTRODUCTION TO THE MATLAB APPLICATION DESIGNER

EXERCISES

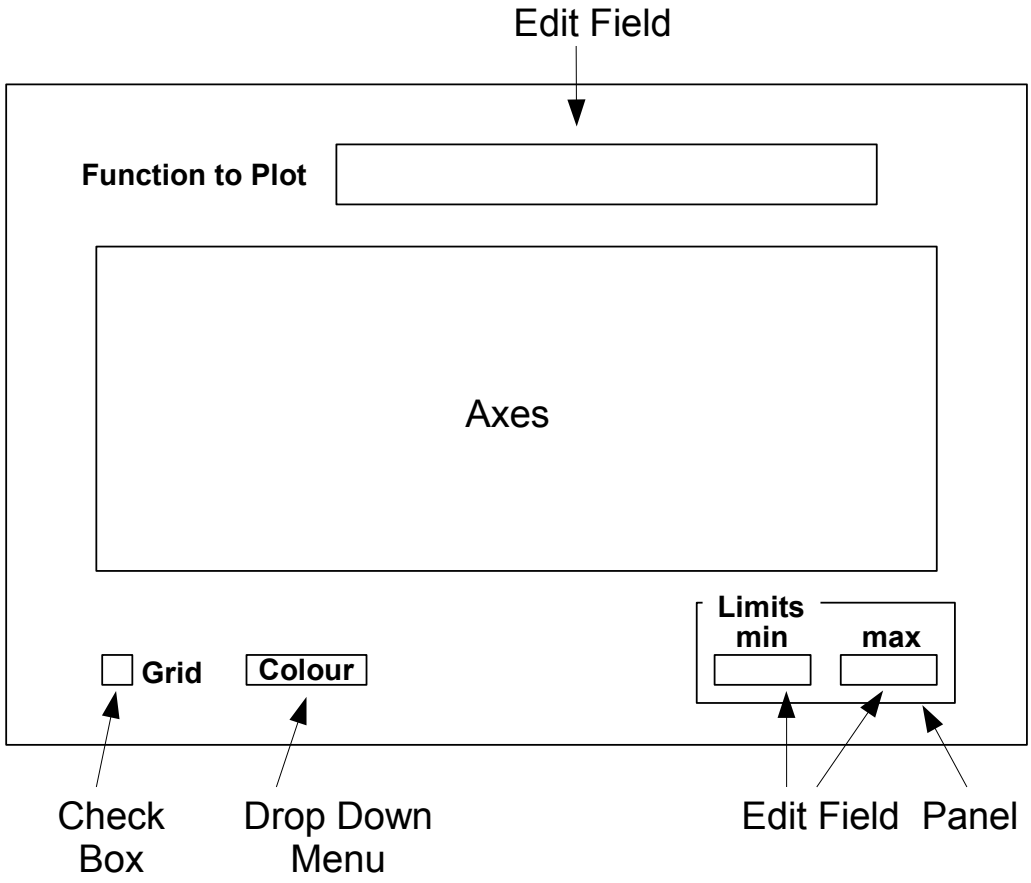


Eric Peasley, Department of Engineering Science, University of Oxford
version 4.6, 2018

MATLAB Application Exercises

In these exercises you will be building your own MATLAB App using the MATLAB App Designer.

Before starting any design, it is a good idea to make a rough sketch of the what you want the app to look like.



The purpose of the app we are building today is to plot the MATLAB expression entered into the Edit Field at the top of the figure. The graph will be plotted between the two values in the Edit Fields in the bottom right Panel. The Check Box will toggle the grid on and off. The Drop Down Menu will be used select the colour of the graph.

The first exercise will produce the minimum required to get the app working. The following exercises will each add more controls and extra functionality.

Exercise 1 (The Basic Program)

In this exercise you will create your app and start laying out the graphical objects and controls. You will also automatically generate the MATLAB program and edit it to produce a basic function plotter.

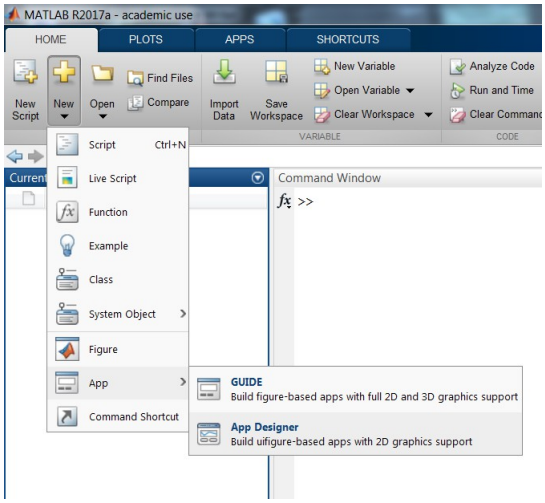
Start the App Designer

Select the **HOME** icon bar at the top of the MATLAB window.

R2017a and **R2017b**
Then in the New menu, under the yellow plus sign, select **App** then **App Designer**.

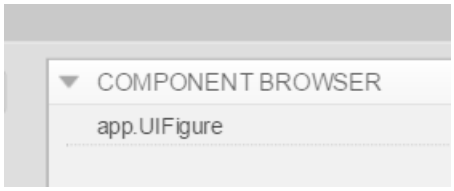
R2018a and **R2018b**
Then in the New menu, under the yellow plus sign, select **App**.

You can also open it by enter **appdesigner** in the command window.



The Component Browser

On the right of the App Designer window you will find the Component Browser as show on the right. At the start, the design only has a single figure show as **app.UIFigure**.



Click on **app.UIFigure** in the browser, to select the figure, then click in the grey region in the browser below **app.UIFigure** to deselect the figure.

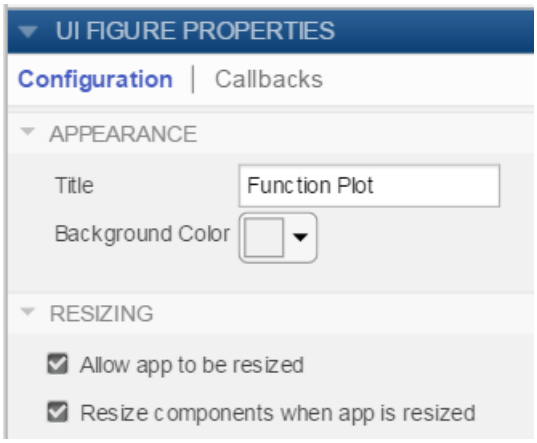
The grey rectangle in the centre of the window is the figure. Notice that this has a blue edge when selected. You can also select the figure by clicking on the rectangle.

The Property Editor

The **Property Editor** is below Component Browser.

Select the figure to observe the figure properties. Change the title to **Function Plot** as shown on the right.

Notice that in the browser the figure has changed name to **app.FunctionPlotUIFigure**.



Setting the Figure Size

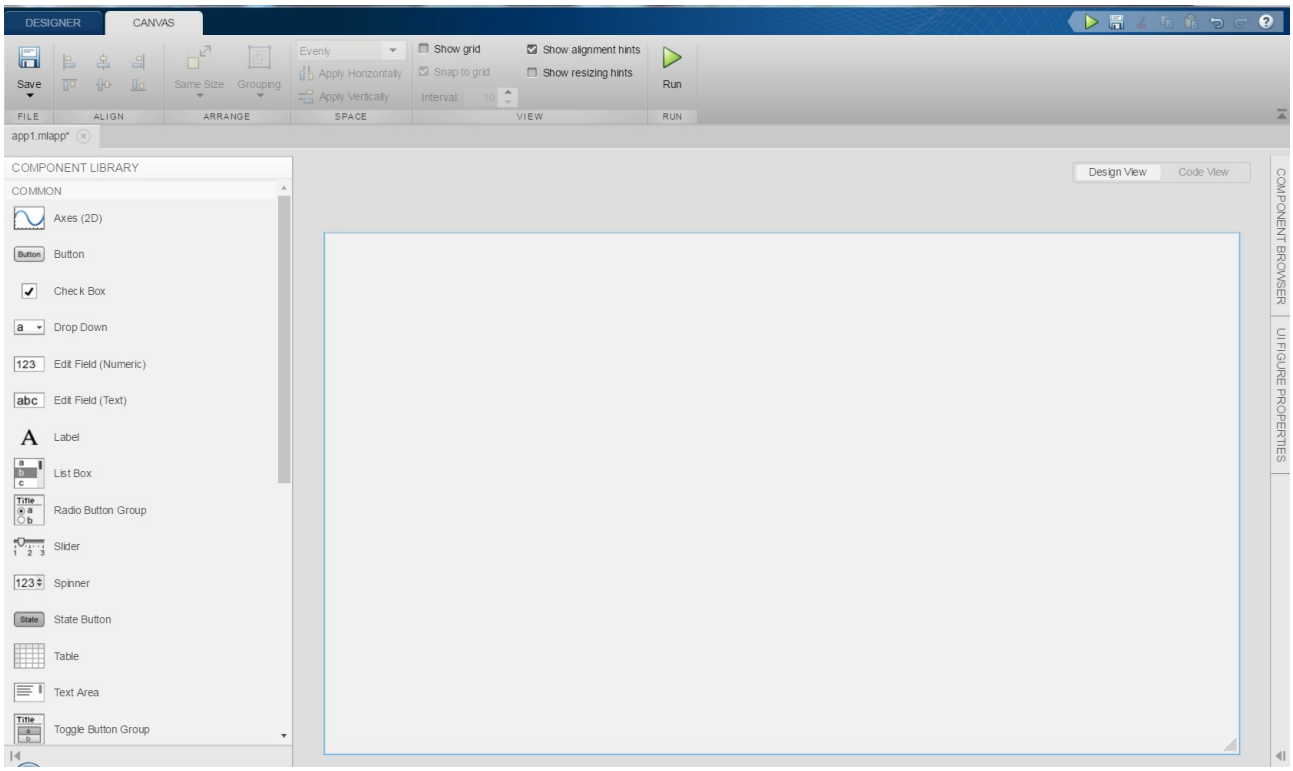
In the bottom left and right hand corners of the designer window are arrows that allow you to collapse and expand the panels on the left and right hand sides. Try these out now.

Collapse the panels on the right, but leave the **Component Library** open on the left.

Maximise the designer window.

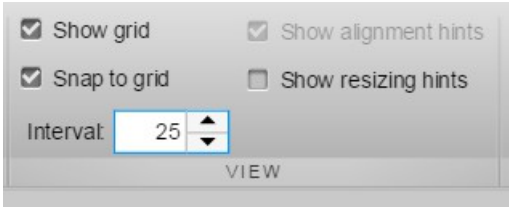
Drag the bottom right hand corner on the figure so that the figure occupies the majority of the space between **Component Library** and the right hand side.

The designer window should now look something like this.

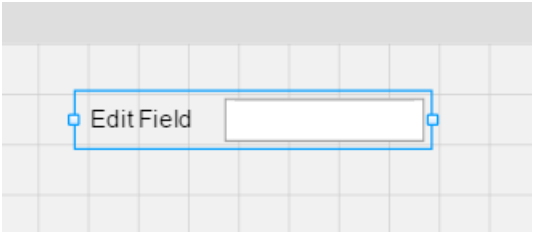


Placing the Components on the Figure

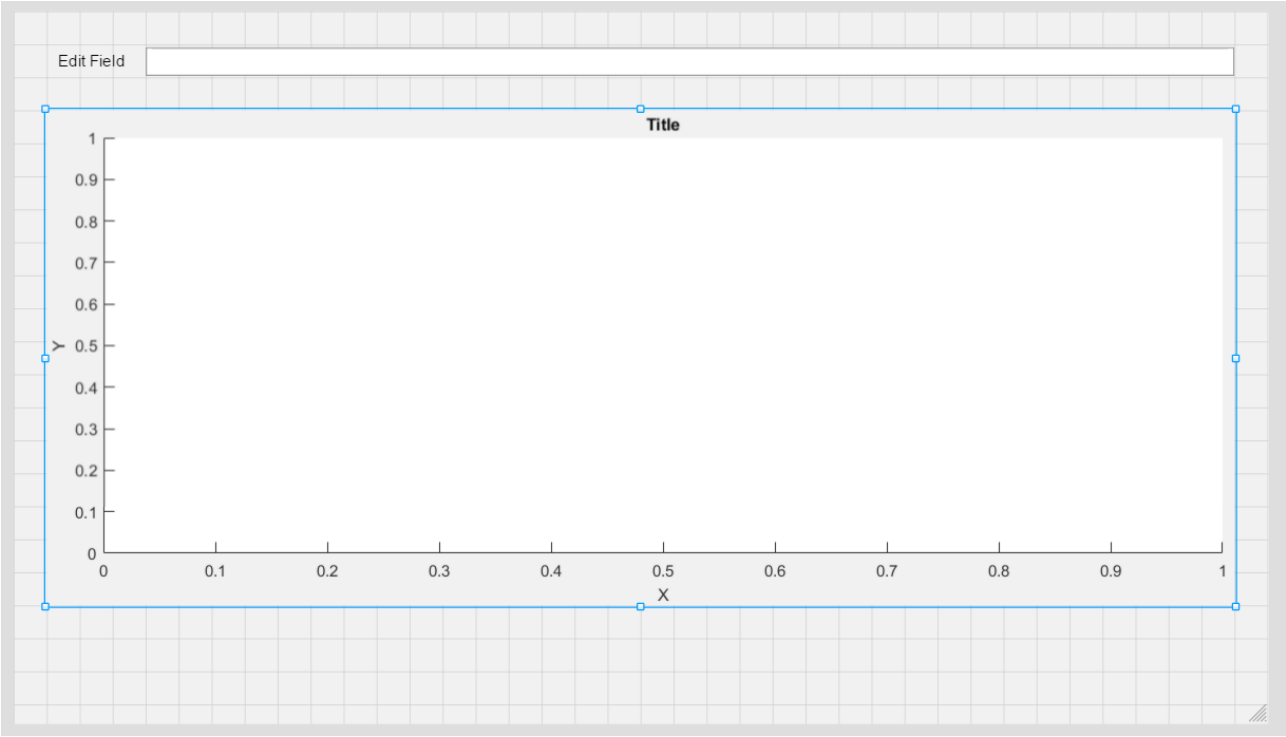
You should be on the **CANVAS** tab at the top of the window. On the **VIEW** region of the icon bar, enable **Show grid** and set the **Interval** to **25**, as shown on the right.



Drag an **Edit Field (Text)** from the **Component Library**, to the top of the figure, approximately one square from the top. Then drag the little blue squares to make it bigger.



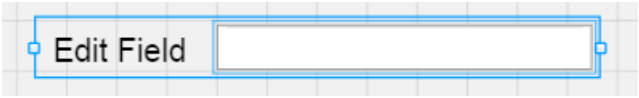
Drag an **Axes** onto the figure below the Edit Field. Resize the **Axes** so that it looks something like this.



Component Properties

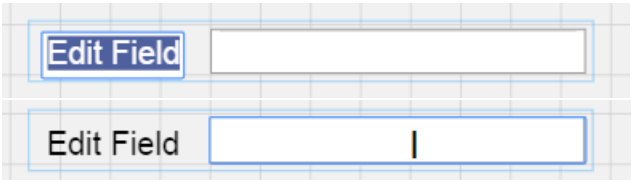
Expand the panel on the right hand side so that you can see the **Component Properties** window. Then click on the **Edit Field** at the top of the figure.

- If you click once on the **Edit Field**, you select the whole edit field including the label.
  The properties of both are shown in the **Component Properties** window.



- If you then Click on the **Label** or the **Edit Field** box, you only get the properties of the **Label** or the **Edit Field**.

- If you double click on the **Label** or in the **Edit Field** box, you can edit the **Label** text or the text in the **Edit Field** box.



Change the properties of the **Edit Field** so that :

1. The font of both the **Label** and the **Edit Field** are 16 point, Arial, Bold.

2. The **Label** text is **Function to Plot** and the text in the **Edit Field** box is **sin(x).**
   You may need to adjust the size of the **Edit Field** again.

3. The text in the **Edit Field** box is centred.

Select the **Axes** and in the **Component properties** window:

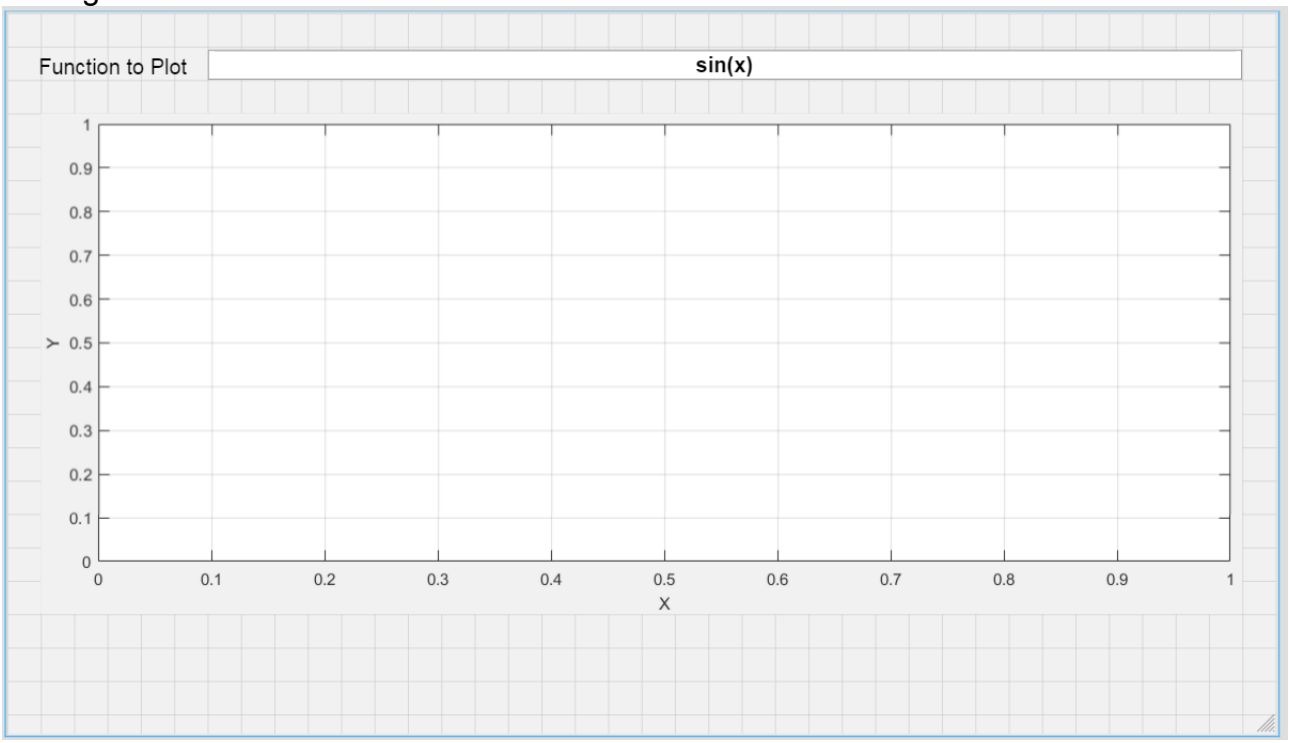
1. Empty the **Title** box.

2. MATLAB release R2018a and older
   Under **Appearance** turn on the box and the grid.

   MATLAB release R2018b
   Under **Grids** tick both **XGrid** and **YGrid**

In the **Component Browser** window, double click on the **Axes** and change the name to **MainUIAxes**.

The figure should now look like this.



Saving and Viewing the MATLAB code.

Save the app as **FunctionPlot.mlapp**.

We have finished the layout of the app for now. The next step is to look at the code.

Above the top right of the figure is a box as show on the right.
Click on **Code View**.

At first sight, the code can look quite intimidating. However, all code in the program so far is not editable. You cannot change it even if you wanted to. You can create a perfectly functional app without understanding this code.

Click on the **Run** icon on the icon bar.

The program creates the figure and all the components. You can type things into the **Function to Plot Edit Field**, but it does not do anything as yet.

Creating a Call Back Function

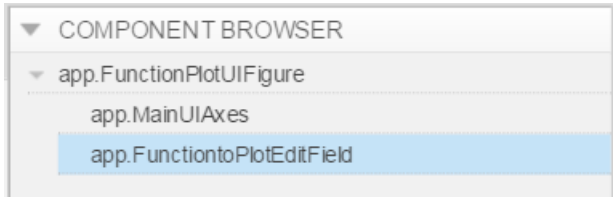
We want something to happen when we type something into the **Edit Field**. For that we need a callback function that will execute after the text in the field has been changed.

There are two Edit Field callback functions available.

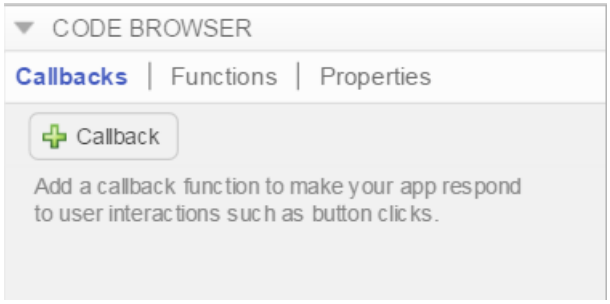
Value Changing Function	Executes for every key press inside the Edit Field.
Value Changed Function	Executes when enter is pressed in the Edit Field or you click outside the Edit Field after entering text.

We want a Value **Changed** Function.

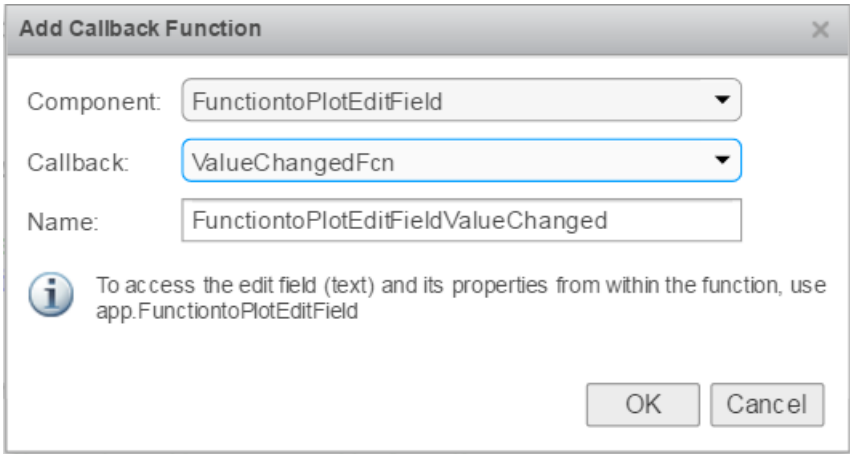
In the **Component Browser** on the right, select the **Edit Field**.



In the **Code Browser** on the left, select **Callbacks** and click on the **+ Callback** button.



Make sure that the Value **Changed** Function is selected and then hit the **OK** button.



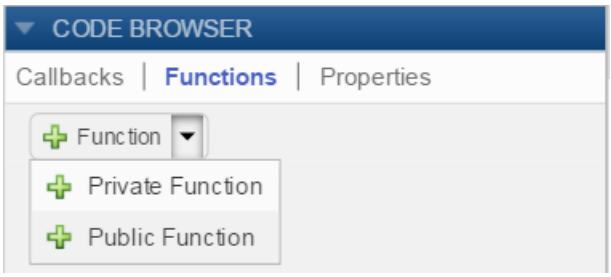
You should now be able to see the callback function in the code.

```
method   (Access = private)

    % Value changed function: FunctiontoPlotEditField
    function FunctiontoPlotEditFieldValueChanged(app, event)
            value = app.FunctiontoPlotEditField.Value;

    end
end
```

Notice that we have for the first time a section of code that is not greyed. We can edit the code in the white box in the function. However, before we do that, we are going to create our own function in the code that we can use in the callback function.

Adding Your Own Functions

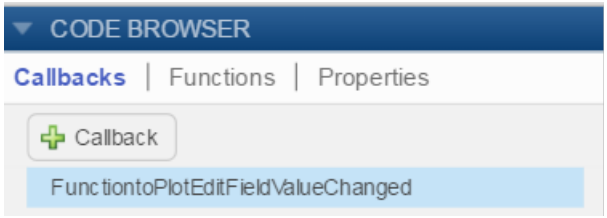
In the **Code Browser** on the left, select **Functions** and click on the down arrow next to **+ Function** and in the menu that appears, select **+ Public Function**.



The following gets inserted into the code

```
methods (Access = public)

    function results = func(app)

    end

end
```

In the Code Browser, click on **Callbacks** and select the function **FunctiontoPlotEditFieldValueChanged**



Cut the following from the callback function:

```
  value = app.FunctiontoPlotEditField.Value;
```

In the Code Browser, click on **Functions** and select the function **func(app,...)**

Paste the line of code, that you cut out earlier, into the function func.

```
  function results = func(app)
     value = app.FunctiontoPlotEditField.Value;
  end
```

This line of code takes the text written in the Edit Field and puts it into the variable ***value***. This will contain the MATLAB expression that we want to plot.

We will assume that the expression typed in will use ***x*** as the independent variable.

To plot a graph, we need to

- Produce a vector containing values of ***x***.
- Read in the text from the edit field
- Evaluate the expression in the text for every value in ***x*** and put the result into a variable ***y***.
- Plot ***x y***.

Rename **func** to **replot** and make the following changes:

```
methods (Access = public)

    function replot(app)
        % Function to replot the graph

        % Define the limits of x
        minx = -5;
        maxx = 5;

        %Produce the x values
        x = linspace(minx,maxx,1000);

        % Read in the string from the Edit Field
        value = app.FunctiontoPlotEditField.Value;

        % Evaluate the string as a MATLAB expression
        y = eval(value);

        %Plot the graph
        plot(app.MainUIAxes,x,y)
    end
end
```

Notice that when you plot in the app, you need to specify which axes the plot is going go into.

Use the **Code Browser** to go back the the callback function **FunctiontoPlotEditFieldValueChanged** and add the following into the function.

```
  replot(app)
```

Save and run the app.

Try entering you own MATLAB expressions into the edit field on the app and see if it plots.

The Startup Function

If you restart the app, you will find that the default expression at start time is not plotted. You can change this by calling **replot** in the figure startup function.

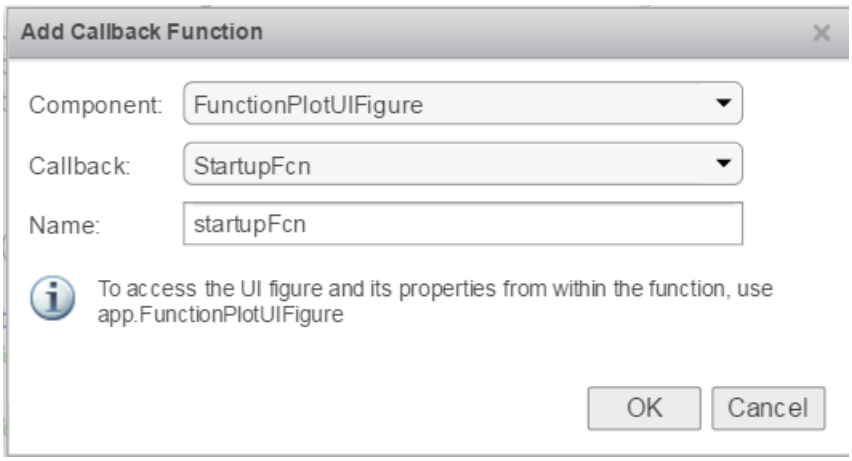
First we need to create a figure startup function.

In the **Component Browser**, select the figure called **app.FunctionPlotUIFigure**.

In the **Code Browser** on the left, select **Callbacks** and click on the **+ Callback** button.

Make sure that the selections are as shown on the right.

Then click on OK.



Then call the replot function in the Startup function.

```
replot(app)
```

Save the app.

Run the app. You should find that **sin(x)** is plotted straight away.

Exercise 2 (Expression Error Recovery)

Enter an erroneous expression into the function to be plotted.
For example, x^2 will not work because a dot is needed after the x.

At the moment the program crashes. What would be better is to detect the error and report what is going wrong without crashing the program.

In the function **FunctiontoPlotEditFieldValueChanged**, replace

```
replot(app)
```

with

```
try
  replot(app);
catch err
  errordlg(err.message,'Expression Error');
end
```

Save the app.

Run the app.

Enter the expression again to see the error dialogue. The program should not crash this time.

Exercise 3 (The Grid Check Box)

In this exercise, you will add a Check Box to toggle the grid on and off.

Editing the Design

Return to the **Design View** by clicking in the box above and to the right of the code.

Drag a **Check Box** onto the figure
about one square below the main axes
so that the left hand side is approximately in line with the left hand edge of main axes.

The **Check Box** properties will be shown in the **Component Properties** window.
Change the following properties of the Check Box.

1. **Selected** (**Value** in R2018b) should be ticked.
2. Change the **Text** to **Grid**
3. Change the font to 16 point, Arial, Bold.

Editing the Code

We can determine the state of the Check Box by looking at it's value.

- If the Check Box is ticked, the value will be one.
- If the Check Box is not ticked, the value will be zero.

Return to the **Code View**.

Select the **Check Box** in the **Component Browser**.

In the **Code Browser**, select **Callbacks** and click on the **+ Callback** button.

Select a **Value Changed Function**.

In the callback function for the Check Box, add the following code.

```
%Read in the status of the Check Box
Status = app.CheckBox.Value;

if (Status)  % the Check Box is ticked
    %Turn on both grids
    app.MainUIAxes.XGrid = 'on';
    app.MainUIAxes.YGrid = 'on';
else
    %Turn off both grids
    app.MainUIAxes.XGrid = 'off';
    app.MainUIAxes.YGrid = 'off';
end
```

Run the app and test the Check Box.

Exercise 4 (The Colour Drop Down Menu)

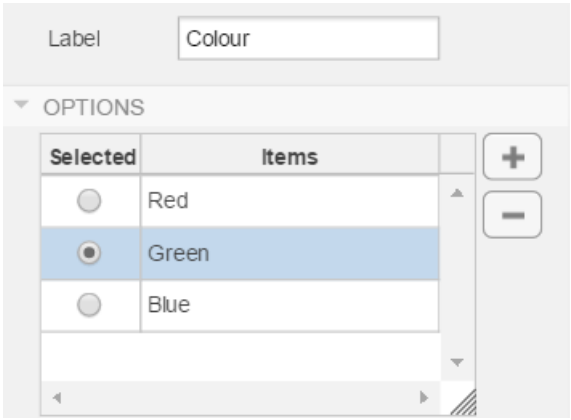
In this exercise you will add a Drop Down menu to change the colour of the line.

Drag a **Drop Down** onto the figure to the right of the grid Check Box.

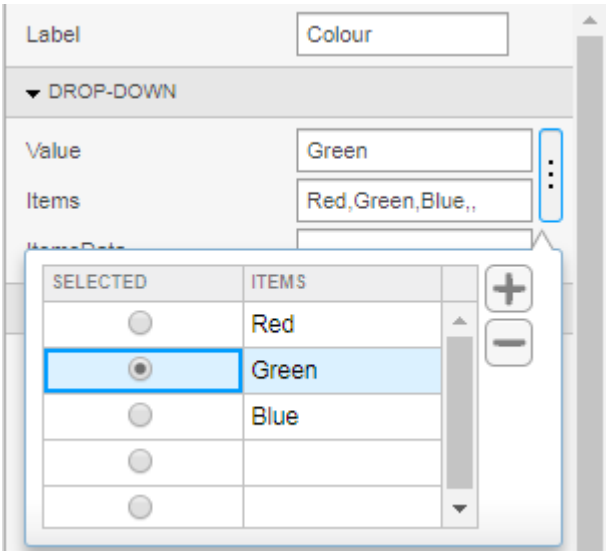
Change the following properties of the Drop Down:

1. Change the **Label** to **Colour**.
2. Change it so there are only three options.
3. Option1 to Red
4. Option2 to Green
5. Option3 to Blue
6. Set Green as the default colour at start up.
7. Change the font to 16 point, Arial, Bold.

R2017a - R2018a



R2018b



Edit the function **replot**. Replace the plot command:

```
%Plot the graph
 plot(app.MainUIAxes,x,y)
```

with the following code.

```
%Get the colour from the Drop Down menu
col = app.ColourDropDown.Value;

%Plot the graph
 plot(app.MainUIAxes,x,y,col)
```

Save and run the app. Select a different colour using the pop-up menu. **Don't worry if the colour does not change straight away.** At this stage the colour will only change after a new expression has been entered. So enter a new expression.

Question 1

The reason that the colour only changes when you enter a new expression is that **replot** only runs at the start of the program and in the callback function of the **Edit Field**. It does not run when you select a new colour with the Drop Down menu.

What can you do to force the graph to plot when you select a new colour?

The answer is on the next page.

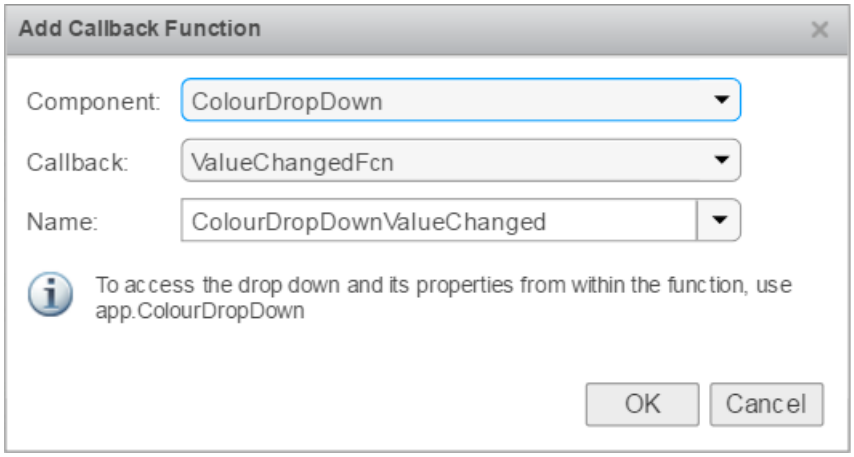
Answer to Question 1

You call the function **replot** in the Drop Down menu callback function.

Select **app.ColourDropDown** in the **Component Browser**.

In the **Code Browser**, select **Callbacks** and click on the **+ Callback** button.

The selections should be as shown on the right.
Hit the **OK** button.



Edit the callback function **ColourDropDownValueChanged**.
Remove all the code in the function and replace with **`replot(app);`**

```
function ColourDropDownValueChanged(app, event)
    replot(app);
end
```

Exercise 5 (X Axis Limits)

In this exercise you are going to add a panel with two Edit Fields that specify the limits of the X axis.

Drag a **Panel** onto the figure. Don't worry about the size or position of the **Panel** yet. Just drag it to the centre of the figure.
Change the following properties of the Panel.

1. Set **Title** to **Limits**.

2. Change the font to 16 point, Arial, Bold.

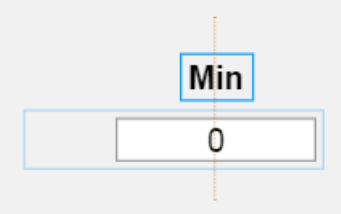
Drag a **Edit Field (Numeric)** into the Panel.
Select the **Label** of the **Edit Field**, then:

1. Set **Text** to **Min**.

2. Centre the text both vertically and horizontally.

3. Change the font to 16 point, Arial, Bold.

Drag the label above the **Edit Field** box and adjust the position so that you see an orange centre line through both the **Label** and the box.



Select the box of the **Edit Field**, then:

1. Centre the **Value**.

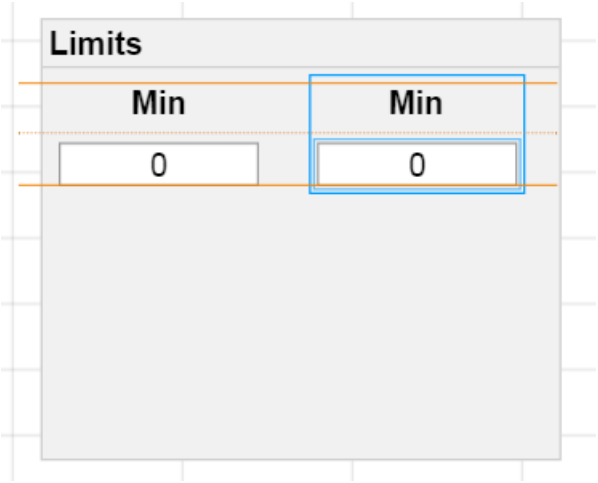
2. Change the font to 16 point, Arial, Bold.

Click on the **Edit Field** away from the both the **Label** and the box, so that both are selected.

Then drag the entire **Edit Field** to the top left corner of the panel.

Right click on the **Edit Field** and select **Duplicate**.

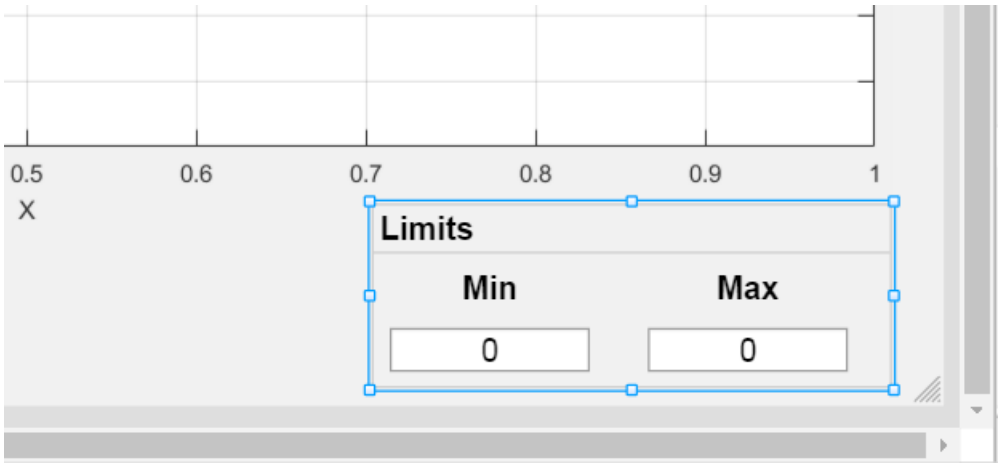
Drag the new Edit Field to the right and adjust until you see orange lines to show that both **Edit Fields** are aligned.



Change the **Text** of the new **Edit Field** to **Max**.

Click on the **Panel** and drag the bottom edge of the panel up, to just below the Edit Fields.

Drag the **Panel** to the bottom right corner of the **Figure**.



If there is not sufficient room for the Panel, resize the **MainUIAxes** to make extra space available.

Change the Value of the **Min** to **-5** and the **Max** to **5**.

In the function **replot**, change

```
% Define the limits of x
minx = -5;
maxx = 5;
```

to

```
% Define the limits of x
minx = app.MinEditField.Value;
maxx = app.MaxEditField.Value;
```

Add a callback function for each **Edit Field** to run **replot** when the value has been changed.

Save and run the app.

Exercise 6 (Using the App in other MATLAB Programs)

Make sure that the App has been saved, then quit the App Designer.

In the MATLAB Command Window enter

```
>>  FunctionPlot
```

to run the app.

The app is an object. You can access the app using the same method used to access a graphical object.

```
>>  myapp = FunctionPlot
>>  myapp.MainUIAxes
```

This means that you can use the app in your own programs.
You should have a MATLAB script called UseApp. Open the script to see what it does and then run the script.

UseApp.m

```
%UseApp
%This program shows how you can get your own programs to use the
App.

%Launch the App.  fp is the app object
fp = FunctionPlot;

% Change the Function to Plot Edit Field
fp.FunctiontoPlotEditField.Value = '4*x.^3 - 3*x';

%Change the Limits of the graph
fp.MaxEditField.Value = 1;
fp.MinEditField.Value = -1;

%Change the Colours available
fp.ColourDropDown.Items = { 'Red' 'Green' 'Blue' 'Magenta' 'Cyan'
};

%Change the Colour of the Graph
fp.ColourDropDown.Value = 'Cyan';


%Replot the graph using the public function in the App.
fp.replot();
```

Exercise 7 (Packaging a MATLAB App)

You may have noticed the **APPS** tab at the top of MATLAB. You can package up your App so the other users can install the App onto the **APPS** tab.

Right click on **FunctionPlot.mlapp** in the MATLAB **Current Folder** window and select **Open** to open the App in the **App Designer**.

Click on the **Designer** tab at the top of the window.

R2017a and R2017b	**R2018a and R2018b**
Click on the **Package App** Icon	Click of the **Share** Icon Then select **MATLAB App**

Fill in whatever details you want, e.g. **Author Name**

Hit the **Package** button on the right. Wait for it to say **Packaging Complete**.

Close the **Package App** window.

You will see a number a new files have appeared.

Function Plot.prj	The project file that contains the information that you have entered in to the Package App window. Clicking on this file will reopen the project in the Package App window.
Function Plot.mlappinstall	Is the file to install the app into MATLAB. This is the file that you send to users that want to use the app.

To install the App into MATLAB, double click file **Function Plot.mlappinstall**.

The installed App will not run while you are in the this folder. Right click in the **Current Folder** window and select **New Folder**. Then double click on the **New Folder**.

To run the app, click on the down arrow on the right of the **Apps** banner at the top of MATLAB.

Under **MY APPS**, click on **Function Plot**.

To remove the app, right click on the app and select **Uninstall**.