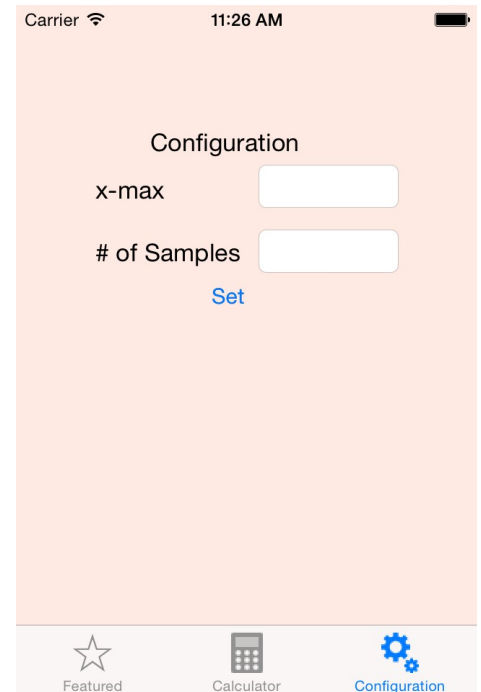
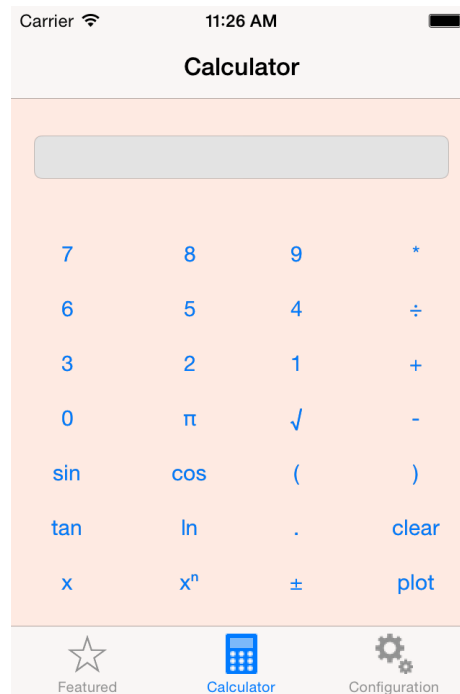
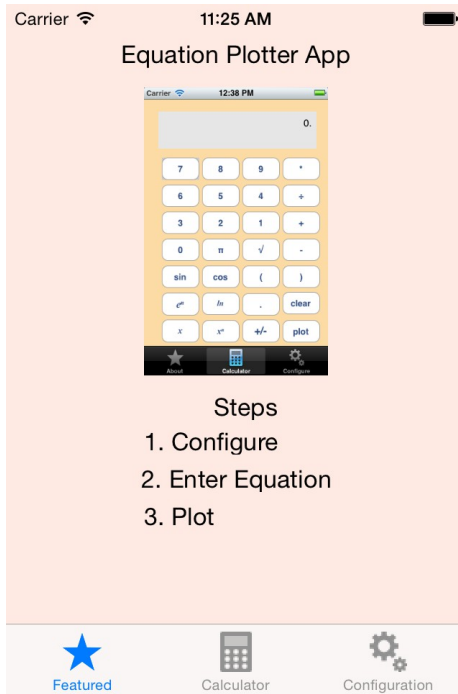


App 3

Multi-View Expression Calculator

Overview

App 3 consists of three separate *top-level* views as shown below:



1st View – Initial view or displayed when Tab Bar Button *Featured* is pressed

2nd View – Displayed when Tab Bar *Calculator* button is pressed

3rd View – Displayed when Tab Bar *Configuration* Button is pressed

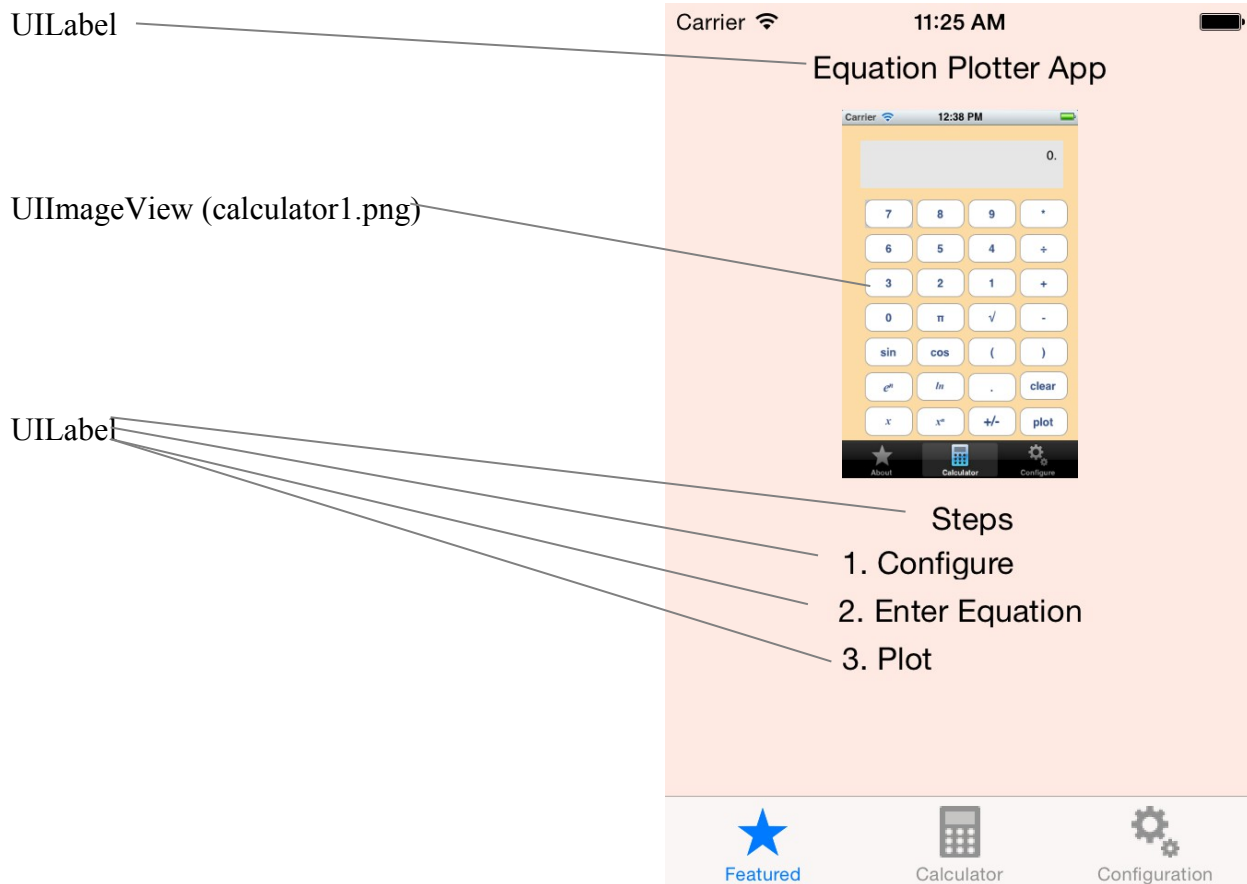
The UITabBar is created from the following images/titles:

Title	Image
Featured	fave.png
Calculator	calculator.png
Configuration	gear.png

The view associated with the *About* button should be displayed when the App starts. A description of each view follows:

Feature View

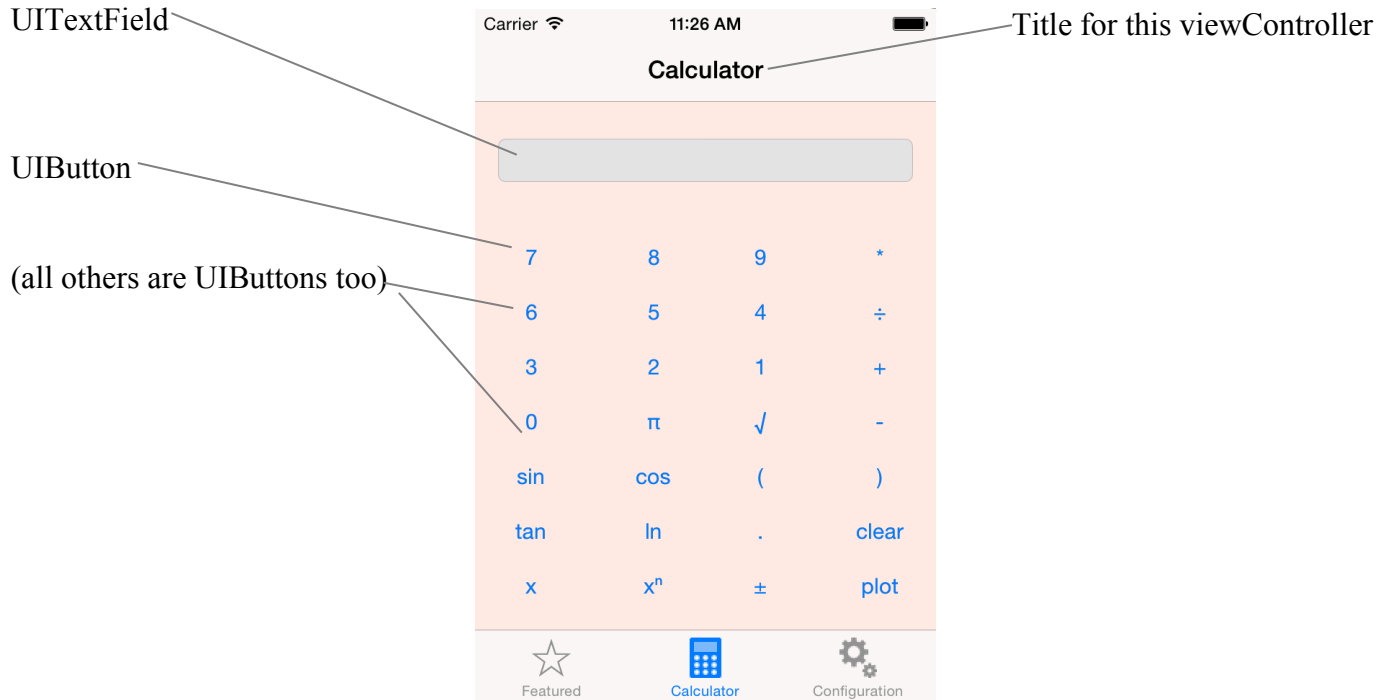
The view associated with the left most UITabBar button – *Featured* – is shown below:



The 3 instructions should be displayed using 3 UILabels when the App starts or the *Featured* UITabBar is pressed.

Calculator View

The *top-level* view associated with the middle UITabBar button – *Calculator* – is shown below:



The purpose for the calculator is to create expressions or equations. The user creates the equations by pressing the corresponding buttons on the calculator. The equations are displayed in the UITextField located at the top of the view. ***The equations are right justified - each keystroke is appended to the right of the current text.*** After the equation has been entered by the user, a plot of the equation can be generated by pressing the *plot* button. The ***default*** constraints for the plots are given below:

$x_{\text{max}} = 5$ (for symmetric plots – starting/ending x have same magnitude, different signs)
 $\text{no_samples} = 50$

These defaults can be changed by the *Configuration* View. Valid equations as entered by the user along with example plots are shown below:

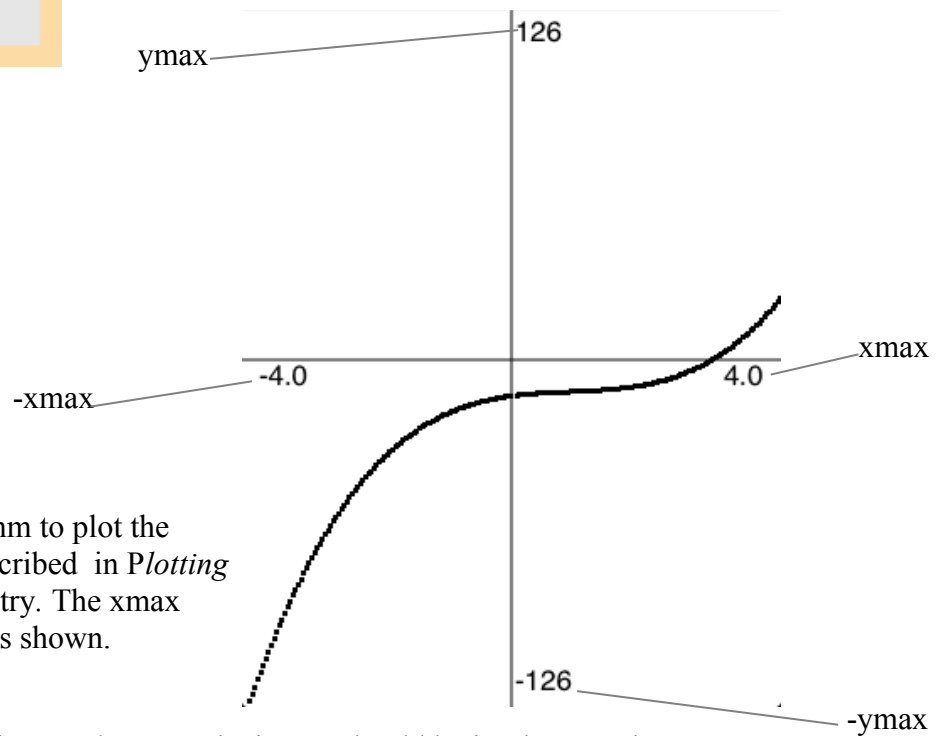
$$x^3 - 2.5x^2 + 3.1x - 13.3$$

Range:

$x_{\text{max}} = 4$

$\text{no_samples} = 200$

You will need to implement an algorithm to plot the equation. The plotting algorithm is described in *Plotting The Equation* section. Note the symmetry. The x_{max} y_{max} is displayed at each axis's edge as shown.



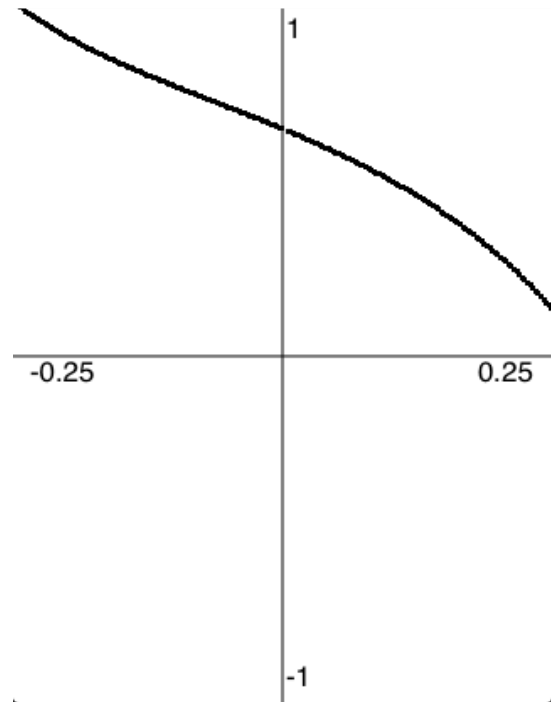
The plot crosshairs are provided as an image *-bg.png*. The image should be implemented as a *UIImageView* for the *UIView* for the *PlotController*. It can be implemented as a *UIImageView*.

$$\sin(x) + \cos(2x) - \tan(3x)$$

Range:

$$x_{\max} = 0.25$$

$$\text{no_samples} = 200$$

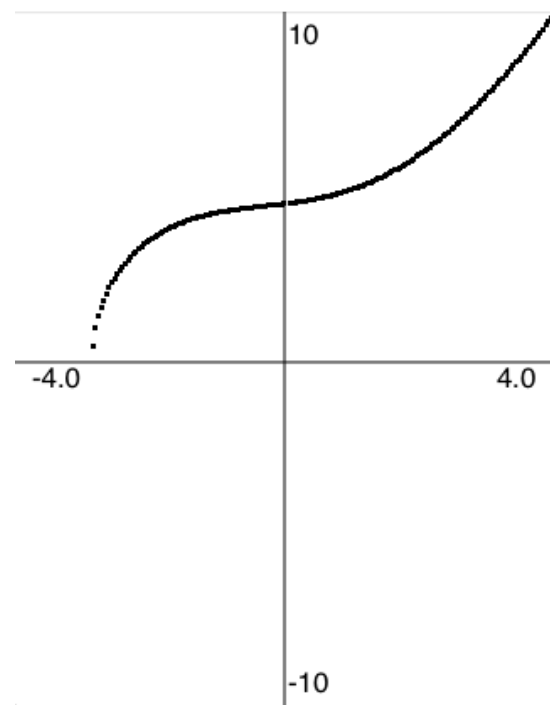


$$\sqrt{x^3 + x^2 + 2.5x + 23}$$

Range:

$$x_{\max} = 4$$

$$\text{no_samples} = 200$$

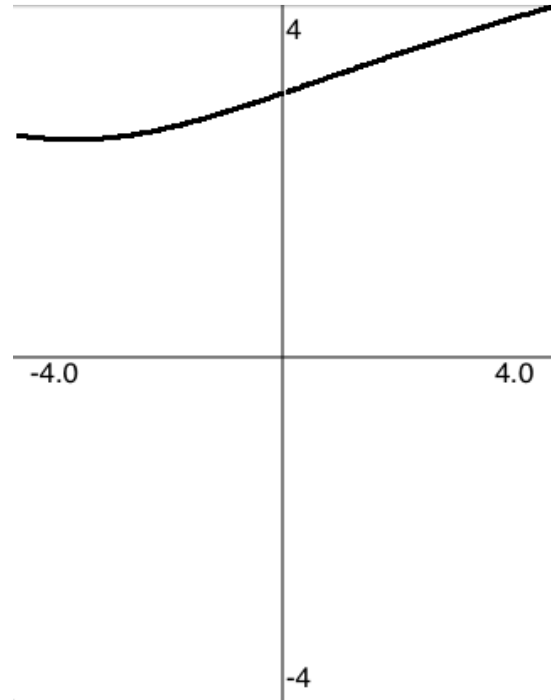


$$\ln(x^2 + 2\pi x + 23)$$

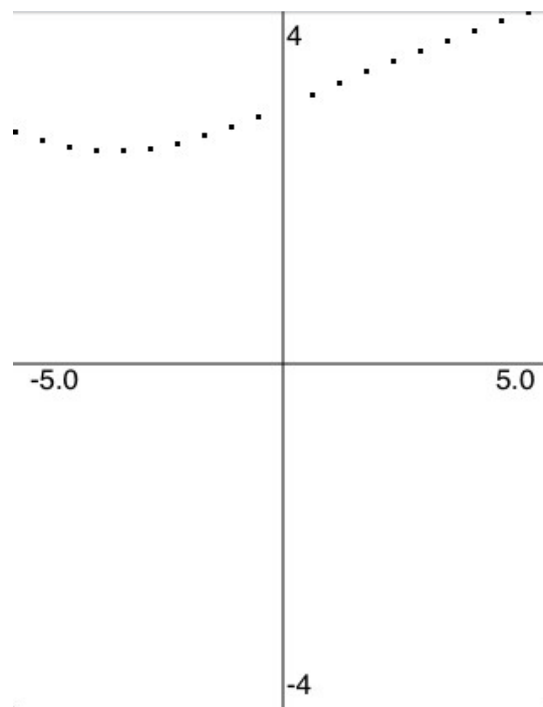
Range:

x_max 4

no_samples: 200



Changing the number of samples changes the sharpness of your plot. Suppose we use 50 for the no_samples for the above equation. The plot now appears as follows:



Each equation should be re-formatted into an appropriate equation that the GCMathParser class can parse – see the notes to view the types of equations and formatting of the equation required by the GCMathParser. There are countless other equations that can be created, but the above examples show you the 4 basic forms that can be created by this expression calculator. A description of all required keys is provided next.

Numeric Key Pad



The numeric key pad is numbered 0 thru 9. Pressing any of these keys will append the corresponding number to the *right* of the current display. Assume the user presses all numbers in an appropriate manner and order. **No validation need be done on the ordering of the numbers in your expressions .**

PI



Pressing π will result in the Unicode π symbol appended to the right of the display. The Unicode symbol must be displayed (e.g., **don't** display 3.1415926 for instance). You can select this character (and other unicode characters) from the Edit->Special Characters menu from XCode. You can select this one from Greek symbols. **No validation need be done on the ordering of π in your expressions.**

Square Root



Pressing the square root key results in the corresponding Unicode symbol appended to the right of the display. The Unicode symbol must be displayed (e.g., **don't** display **sqrt** for instance). An opening parenthesis should be displayed immediately to the right of the square root symbol. The user can then proceed to enter a valid expression. **Assume that the user appropriately adds the ending parenthesis. No validation need be done on the ordering of the square root symbol in your expressions.**

sin



Pressing the **sin** key results in the string **sin** appended to the right of the display. Purpose of key is to compute sine of a value. An opening parenthesis should be displayed immediately to the right of the sin symbol. The user can then proceed to enter a valid expression. **Assume that the user appropriately adds the ending parenthesis. No validation need be done on the ordering of *sin* in your expressions.**

cos



Pressing the **cos** key results in the string **cos** appended to the right of the display. Purpose of key is to compute cos of a value. An opening parenthesis should be displayed immediately to the right of the cos symbol. The user can then proceed to enter a valid expression. **Assume that the user appropriately adds the ending parenthesis. No validation need be done on the ordering of *cos* in your expressions.**

tan



Pressing the tan key results in the string **tan** appended to the right of the display. Purpose of key is to compute tangent of a value. An opening parenthesis should be displayed immediately to the right of the tan symbol. The user can then proceed to enter a valid expression. Assume that the user appropriately adds the ending parenthesis. **No validation need be done on the ordering of *tan* in your expressions.**

(



Pressing the (key results in an opening parenthesis (appended to the right of the display. **No validation need be done on the ordering or balance of opening parenthesis in your expressions.**

)



Pressing the) key results in a closing parenthesis) appended to the right of the display. **No validation need be done on the ordering or balance of closing parenthesis in your expressions. Assume user will end the closing parenthesis whenever appropriate.**

.

Pressing the decimal point key results in a decimal point (period) appended to the right of the display. **No validation need be done on the ordering of the decimal point symbol. Assume user will add the decimal point whenever appropriate.**

ln



Pressing the ln key results in the string ln appended to the right of the display. Purpose of key is to compute natural logarithm of a value. An opening parenthesis should be displayed immediately to the right of the ln symbol. The user can then proceed to enter a valid expression. Assume that the user appropriately adds the ending parenthesis. **No validation need be done on the ordering of *ln* in your expressions.**

x



Pressing the x button results in the letter x appended to the right of the display. This key can be pressed in any order in any expression as often as desired always resulting in a valid expression. **No error checking need be performed on this key – assume it is used properly.**

Exponentiation



Pressing the exponentiation key allows the previously entered x to be raised to an exponential power. For simplicity, the following rules apply to exponentiation for our App:

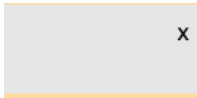
1. Only **one** integer – 0 thru 9 – can be used for exponents.
2. Exponents only apply to previously entered x – no other functions/values can be raised to a power.
3. No fractional exponents
4. No exponents > 9 used Immediately
5. The entered exponent must be displayed as a raised superscript – use appropriate unicode character

You do not have to perform error validation on these rules – assume use will enter the exponents correctly.

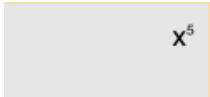
For our application the x value has to be raised to an integer power – although in theory any floating point values could be raised.

An example of exponentiation is outlined as follows:

1. First the letter x is pressed as shown below:

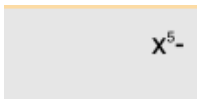


2. Next the exponentiation key is pressed – nothing is displayed – but the key is in exponentiation mode. The next integer will be displayed as a superscript above the x (you can use Edit->Special Characters from Xcode and choose *Digits*).
3. Suppose the number 5 is pressed – it would be displayed as follows:



Since only one exponent is used in our app, the next operand will be displayed in normal mode, no longer as an exponent.

4. Suppose the user now presses a -. The display will appear as follows:



The user continues building the expression using single integer exponents as desired for the previously entered x.

clear



Pressing the clear button reset the text field. All displayed text is removed and the display should appear as



shown below:

The Operators + - * /

The 4 basic operators



Are provided on our calculator. Pressing the multiply, add, or subtraction operator **in a valid manner** results in the corresponding character appended to the right of the screen. Pressing divide in a **valid manner** results in the / character displayed (**use this character instead of the Unicode character shown on the button**).

Assume the user enters the operators in a valid way in order to form a valid GCMathParser expression. No error checking need be performed.

positive/negative



Pressing the positive/negative key adds a – sign. **For simplicity, assume that this button is used only at the beginning of the expression (1st character). If pressed a second time, the – sign is removed.**

plot

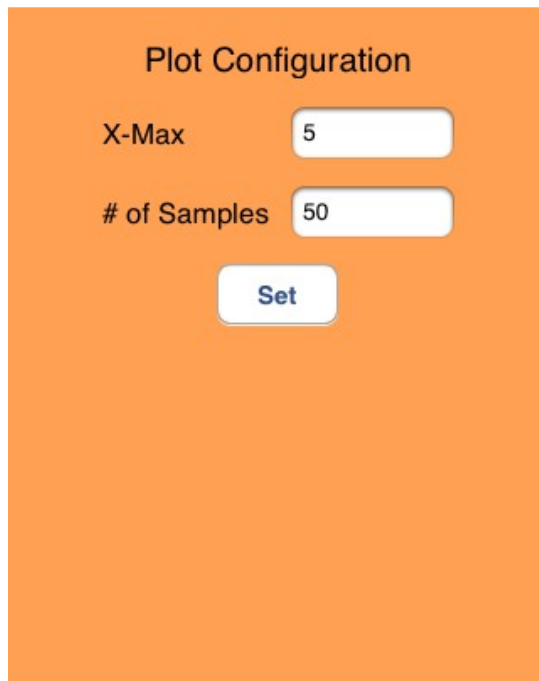


Pressing the plot key will perform the following operations:

1. Generate a Plot of the equation previously entered.
2. **Display an appropriate UIAlertView if no equation has been entered** (i.e., the UITextField is blank)
3. The Configuration **previously** entered will be used to plot the equation. If no Configuration has been entered, the default Configuration values will be used – xmax = 5, # samples = 50.

Configuration View

Initially pressing the Configure UIButton on the UITabBar results in the following default display:



The image shows a mobile app screen titled "Plot Configuration" with an orange background. It contains two text input fields: "X-Max" with the value "5" and "# of Samples" with the value "50". Below these fields is a white button with the text "Set" in blue.

Default x_max is 5

Because of symmetry, the X-Max field sets the start/end of x-axis. The number of samples effect how many points will be generated between the x_max and -x_max. For the default case, the -x_max is -5 and the x_max is 5. ***Assume for our test purposes that the largest possible value for X-Max is 100, and the largest number of samples is 10000.*** No data validation need be performed on these fields – assume user enters appropriate values for these fields.

Pressing either of the UITextFields results in a numeric keyboard displayed.
Pressing the Set button must remove the keyboard and display the following message:



The image shows the same "Plot Configuration" screen, but with updated values: "X-Max" is now "100" and "# of Samples" is now "10000". The "Set" button is still present. Below the button, the text "Configuration Set Successfully" is displayed. A thin grey line points from the text in the paragraph above to this message.

Plotting the Equation

The first step in plotting a given equation is to first generate a set of points – a set of y values for a set of x values. Create a single point by computing a corresponding y-value for each possible x-value between the starting x and ending x position (i.e., between -x_max and x_max). The y-value is computed based on the equation of the polynomial – just plug in a value for x and get the corresponding y. The x-values should be evenly distributed between the -x_max and x_max positions. For the default case example, there will be 50 points *evenly* distributed between -5 and 5.

You will need to compute an offset between each x-value equal to:

$$\text{offset} = (x_max - (-x_max)) / \text{number_of_samples}$$

Next, you will need to compute the maximum x value and the maximum y value computed from the equation. For this example, the maximum x value would be 5. You will need to iterate thru each computed y value to determine the maximum y values.

These points must be evenly *distributed* (i.e., plotted) between the -x_max/x_max values along the x-axis, as well as -y_max / y_max along the y-axis. In other words, the plotted points must be *scaled* to cover the majority of the iPhone's UIView. The width and height of the UIView can be given as:

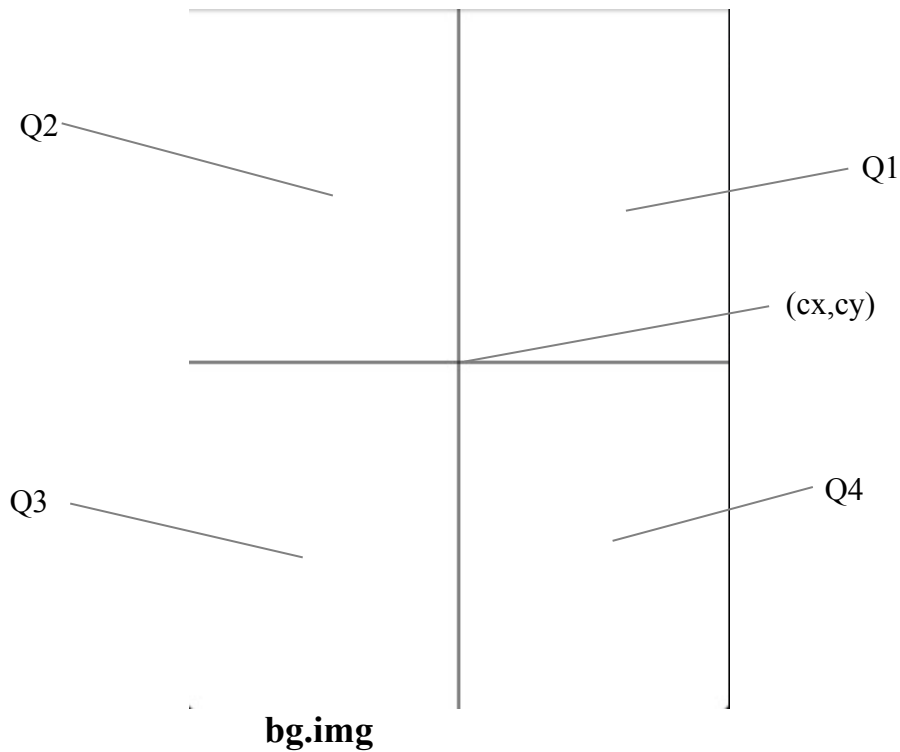
```
var width = self.view.frame.size.width  
var height = self.view.frame.size.height
```

and the width/height of the UIImageView is shown below:

```
var widthImage = imageView.frame.size.width;  
var heightImage = imageView.frame.size.height;
```

The width/height will be different depending on which iPhone it is running on. (These equations will also be correct for any Ipad as well). All plot points consist of an x coordinate and a y coordinate which should be computed with respect to the plot origin – the intersection of the x-axis and y-axis as shown above. This origin coordinates (cx,cy) is given below in terms of their location in the UIView.

The plot must be displayed correctly for all iPhone simulators.



The locations for the x_max/y_max and the centerpoint of the image (cx, cy) are given below:

```
var x_plot_max = widthImage/2
var y_plot_max = heightImage/2

var cx = widthImage/2
var cy = originY + heightImage/2
```

originX and originY are the upper left coordinate of the UIImageView.

There are 4 sets of scaling equations based on the quadrant that the generated point resides in. The equations that convert the generated x/y values into values that can be plotted onto the iPhone's UIView are given below (x_max is the largest possible x value while y_max is the largest computed y value, $points[i]$ is a set of computed y values for each value of x)

Quadrant 1 (Q1): ($x > 0, y > 0$)

```
x_plot = cx + (fabs(x)/x_max)*x_plot_max;
y_plot = cy - (fabs(points[i])/y_max)*y_plot_max
```

Quadrant 2 (Q2): ($x < 0, y > 0$)

```
x_plot = cx - (fabs(x)/x_max)*x_plot_max;
y_plot = cy - (fabs(points[i])/y_max)*y_plot_max
```

Quadrant 3 (Q3): ($x < 0, y < 0$)

```
x_plot = cx - (fabs(x)/x_max)*x_plot_max;  
y_plot = cy + (fabs(points[i])/y_max)*y_plot_max
```

Quadrant 4 (Q4): ($x > 0, y < 0$)

```
x_plot = cx + (fabs(x)/x_max)*x_plot_max;  
y_plot = cy + (fabs(points[i])/y_max)*y_plot_max
```

Displaying a Point

The best way to plot a point on the iPhone's UIView is to use the Core Graphics, which we have not covered yet.

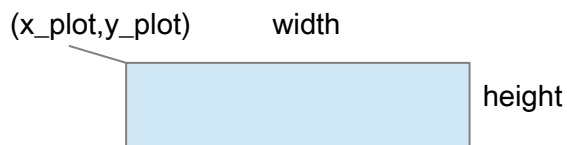
As an alternative, we can display the point on the iPhone's UIView as a UILabel. First, we would need to programmatically create the UILabel as shown below:

```
var label = UILabel()  
label.frame = CGRectMake(CGFloat(x_plot), CGFloat(y_plot), 2.0, 2.0)  
label.backgroundColor = UIColor.redColor();
```



The above point will be created by a UILabel of 2x2 pixels – you can experiment with different sizes such as 1x1 pixel.

CGRectMake creates the width/height of the UILabel and sets its position as shown below:



A width/height set to 2 works well for this App, but you can experiment with different values if you like.

Next, the background color for the UILabel should be set to *red* as shown above:

And finally, the label is added *dynamically* to the UIView of PlotController by the following:

```
self.view.addSubview(label)
```

Testing

Be sure to test your App as follows:

1. Zip up your entire project.
2. Unzip your project on a different machine than you developed on (preferably one in Room 214 or in the library).
3. Run your app on that machine. Make sure it works as it does on your original machine.

What you Submit

You should name your Xcode project **EquationPlotter**. Zip up the entire Xcode project. Submit the zip file via Blackboard.

Grading

It's important to provide the core functionality of the App as specified in this document. **You will not be graded on how identical your App is to the example.** Feel free to use different backgrounds, different options, and different layouts if you like.