Engineering
at Alberta

JSXGraph

Weichen Qiu

August 2020

Contents

Introduction

JSXGraph is a Javascript graphing library that can be used in conjunction with HTML's `<input>` widgets to produce interactive examples.

This documentation will cover the basics of the following:

1. HTML
2. CSS
3. Javascript
4. JSXGraph

Setup

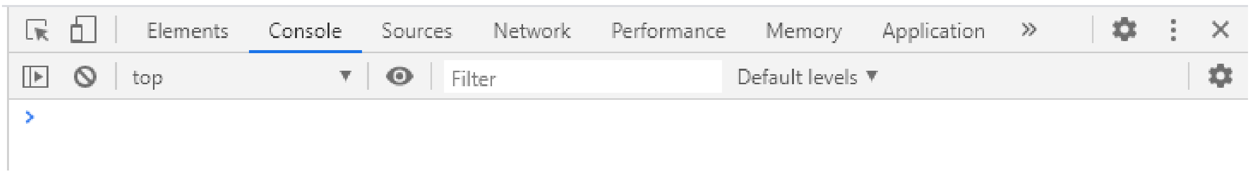
All code must be written in a file with a **.html** extension, this file can be opened by a text editor and/or a web browser.

Open the HTML file in a text editor to write the code and also have it open in the web browser to see the current graph.

Open the web browser's console to see errors in the code and for debugging purposes:

1. In web browser, hit **CTRL + SHIFT + I** or **right-click + INSPECT**
2. A window will open on the right, at the top bar of this window, select the **Console** tab

Javascript code can be run in the console, type code next to **>** and hit enter



Make changes to the code, hit **save** or **CTRL + S,** and refresh (**CTRL + R**) the web browser to see new changes.

HTML

HTML (Hypertext Markup Language) consists of all the elements that are visible on the page, such as sliders, text, and buttons, and makes up the structure of the web page.

Basic Page Structure

Type `html` then hit [TAB] to get the following basic structure for a HTMl file:

```
<!DOCTYPE html>
<html>
<head>
      <title></title>
</head>
<body>
</body>
```

```
</html>
```

1. `<head></head>` - All imported libraries must be declared inside head
2. `<body></body>` - All elements and text on the page **must** be inside the body tag

HTML Tags

HTML is a tag language with an opening tag: `<tag>` and a closing tag: `</tag>`. All text must be written in between the opening and closing tags. `<tag></tag>`

Most text editors will auto complete tag names by hitting [TAB] (ie. Type div and hit [TAB] gives <div></div>)

The most common tag is `<div></div>` or division, which groups text and elements into "containers".

Attributes

Each tag can have attributes in its opening tag, in between the tag's name and the ">" bracket.

```
<tag attr=" "></tag>
```

These attributes vary for different tags and are used to declare important information needed for that tag. For instance, the `<input>` tag has a type attribute that declares the type of input.

Common HTML Tags

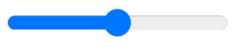
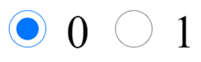
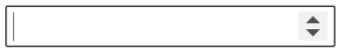
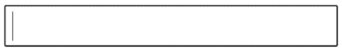
<div></div>	Division	<script type="text/javascript"></script>	Write Javascript code
<input type="">	Takes user inputs	<style type="text/css"></style>	Write CSS code
<br>	New line break	<table></table>	HTML table
<sub></sub>	Subscript	<strong></strong>	Bold
<sup></sup>	Superscript	<h1></h1>	Header Text

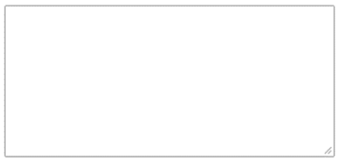
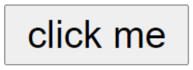
Input tag

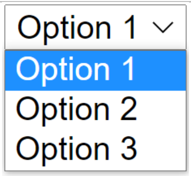
The input tag is used to interact with the user and will be used later to modify JSXGraphs.

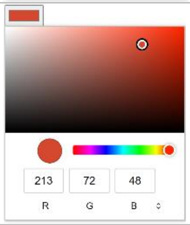
Full list of HTML Input types here

```
<input type=" ">
```

	Slider	<input type="**range**" min="0" max="10" value="5" step=".1">
	Radio Buttons **name="" must be the same*	<input type="**radio**" name="mc" value="0" **checked**> <label for="0">0</label> <input type="radio" name="mc" value="1"> <label for="1">1</label>
	Checkbox	<input type="**checkbox**">
	Number Input	<input type="**number**">
	Text Input	<input type="**text**">

	Text Box	`<textarea></textarea>`
	Button	`<input type="button" value="click me">`

	Dropdown Menu	`<select>`
		    `<option value="1">Option 1</option>`
		    `<option value="2">Option 2</option>`
		    `<option value="3">Option 3</option>`
		`</select>`

	Color Picker	`<input type="color">`

CSS

CSS (Cascading Style Sheets) handles all the design and styling of the page.

CSS can be written directly in the html file in 2 places:

- Inside the opening tag as an attribute: `<tag style=""></tag>`
    a. eg. `<div style="color:red"></div>`
- Between the style tags: `<style type="text/css"></style>`.
    a. Longer lines of code should be written between style tags.

1. In order to identify which HTML element to style, the element is assigned an `id` or `class` attribute
    a. id can only be assigned to one element, and is unique to that one element
        i. `<div id="my_id"></div>`
    b. class can be assigned to a group or a class of elements
        i. `<div class="my_class"></div>`
2. In between the `<style type="text/css"></style>` specify the target element to style using its `class` or `id` name and code its style inside the {} brackets.
    a. For Id's, **#** (ie. Hashtag) is used:
        i.
```
<style type="text/css">
    #my_id{      }
</style>
```
    b. For class, **.** (ie. Dot) is used:
        i.
```
<style type="text/css">
    .my_class{  }
</style>
```

3. The styling options are declared inside the { } brackets,
   a. First type the styling attribute, then a colon [:], followed by the value and ended with a semicolon
      i. `attribute:value;`
   b.
```
<style type="text/css">
    #my_id{
        width: 20%;
    }
</style>
```
   c.
```
<style type="text/css">
    .my_class{
        background-color: blue;
    }
</style>
```
   d. Note: the <u>colon</u> in the middle and <u>semicolon</u> at the end are required
   e. Numeric Values can be expressed in 2 ways:

i.	**`px`** - pixels	eg. `20px`
ii.	**%** - percentage	eg. `20%`

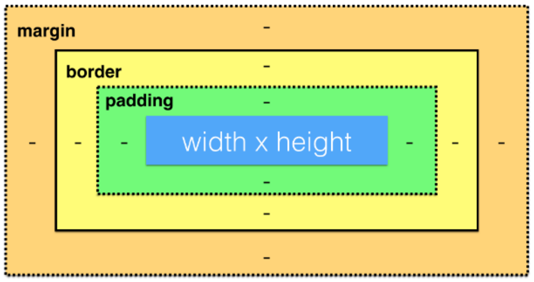
      Note. Percent should be used for measurements in width/horizontal direction to adjust for window size
   f. Color Values can be expressed in 4 ways:

i.	**#**_____- hex	eg. `#00CED1`
ii.	**`rgb`**`(,,)` - rgb ranging from 0 to 255	eg. `rgb(0,255,100)`
iii.	**`rgba`**`(,,,)` - rgb with opacity from 0 to 1	eg. `rgba(0,255,100,0.5)`
iv.	***`colorName`*** – primary color names	eg. `orange`

Common CSS Attributes

font-size: 16px;	Font size	opacity: 50%;	Transparency/opacity
font-family: “arial”;	Font type	width: 20%;	Width
font-weight: 600;	Thickness of font	height: 200px;	Height
color: white;	Font color	background-color: orange;	Background color
text-align: center;	Align text	padding: 1%;	Padding
border: 1px solid black	1 pixel black border	margin: 1%;	Margin

Spacing



Padding

Padding is the space inside the element, between the text content and the border.

- `padding: 1%;` is 1% of space or padding around all sides
- `padding-top: 2%;` is 2% padding at the top. The same applies for bottom, left, right.

Border

Border is the edge of the element.

- `border: 1px solid black;` is a 1 pixel thick solid black border
    a. Format is `border: width type color;`
- `border-radius: 25px;` rounds the corners by 25px

Margin

Margin is the space outside the element and is used to add space between different elements.

- `margin: 1%;` adds 1% space outside the border
- `margin-top: 2%;` adds 2% space above the element. The same applies for bottom, left, right.

How to center in CSS

There are many ways to center elements depending on each situation, 3 ways are as follows:

In example below, center the div with id: "centered"

```
<div id=”outerDiv”>
     <div id=”centered”></div>
</div>
```

```
1. #outerDiv { text-align:center; }
2. #centered { margin-left: auto; margin-right:auto; }
3. #outerDiv { display: flex; justify-content:center;
               align-items:center; }
```

   a. `justify-content` centers horizontally
   b. `align-items` centers vertically

Lining up elements

Flexboxes can be used to line up multiple elements in a row or column

1. Enclose all the elements in an outer div and assign it a class or id

```
a. <div class="flex-box">
        <div>element 1</div>
        <div>element 2</div>
   </div>
```

2. In css, set the `display` to `inline-flex` and set the `flex-direction` to either row or column

```
a. <style type="text/css">
       .flex-block{
            display: inline-flex;
```

```
                flex-direction: ______;
        }
</style>
```

i. Set flex-direction to `row` to line up in a row
ii. Set flex-direction to `column` to line up in a column

Javascript

Javascript handles all the interactivity, math, and calculations behind the graphs.

Javascript must be written inside <`script`> tags:

```
<script type="text/javascript"></script>
```

In most text editors, typing `script`, then [TAB} will return the above.

Notes on Javascript

- Javascript is not indent-sensitive
- Javascript is a very flexible language and is lenient on syntax:
  a. ; is not required to end a line of code, but can be used
  b. `var` is recommended to proceed variable declaration but not required
     - `var my_var = value`
- Use `console.log( )` to print text, variables, elements, arrays, dictionaries, graphs to console
- Use // for single-line comments and /* */ for multi-line comments
- Javascript is a bracket language
  a. If statements
     - `if ( condition ){ … } else { … }`
  a. For loops
     - `for ( var i=0 ; i<10 ; i++ ){    …    }`
  b. Functions:
     - `function my_function ( inputs ) {     …     }`

getElementById

To get an element from HTMl into Javascript, first assign the tag an **id**, not a class:

```
<div id=”div_element”>Hello there</div>
<input id=”input_element” type=”range” value=2>
```

Use `document.getElementById(“id”)` to get the element by its id:

```
<script type="text/javascript">
      var element_1 = document.getElementById("div_element")
      var element_2 = document.getElementById("input_element")
</script>
```

Here, document refers to the webpage and `getElementById` will get the element with the specified id in the document.

Once the element is assigned a variable, information can be retrieved from the elements:

- For div elements, **.innerHTML** is used to get the text inside the div
  ```
  a. console.log(element_1.innerHTML) // prints "Hello there"
  ```
- For input elements, **.value** is used to get numerical numbers, in most cases
  ```
  a. console.log(element_2.value)      // prints 2
  ```

Likewise, the innerHTML and values can be changed using Javascript:

```
element_1.innerHTML = "new text"
element_2.value = 5
```

addEventListener

Event Listeners are used to executed a function when an "event" happens, such as when a user moves the slider.

```
document.getElementById("my_id").addEventListener('event',function(){
   //runs this code when event happens to element with id of my_id  })
```

Here, the element with id of my_id is retrieved using `document.getElementById("my_id")` and an event listener is added to it using `addEventListener` which will execute code in `function(){}` when the event occurs.

Event Types

input	Gets an input (recommended for sliders)	mousedown	When the mouse button is pressed
click	Clicked element	mouseup	mouse button released
change	When changes	mouseover	Mouse is over element
mouseenter	Mouse enters an element	mouseout	Mouse leaves element
mouseleave	Mouse leaves an element	mousemove	Mouse moves

Full list of events here.

JSXGraph

JSXGraph documentation here

Import Library

To import JSXGraph, add the following in between the `<head></head>` tags:

```
<link rel="stylesheet" type="text/css" href="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraph.css" />
<script type="text/javascript" src="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraphcore.js"></script>
```

Board

The JSXGraph Board is where the graph is drawn. Add the following in between the `<body></body>` tags:

```
<div id="box" class="jxgbox" style="width:500px; height:500px;"></div>
```

Change the width and height here to resize the graph.

Inside the `<script>` tag, initialize the board:

```
var board = JXG.JSXGraph.initBoard('box', {boundingbox: [-5, 5, 5, -5],
axis:true,showCopyright: false});
```

- boundingbox is the x and y range, which is specified by the top left and bottom right corners
    a. boundingbox:[top_left_x, top_left_y, bottom_right_x, bottom_right_y]
- `showNavigation:false` can be used to turn off the controls at the bottom right

Board.create

Board.create is used to add shapes and graphs to the board.

Function Graph

Graph of $y = \sin(5x) + \cos(2x)$

```
var y = board.create('functiongraph', [function (x) { return
Math.sin(5*x)+Math.cos(2*x) }], {strokeColor:'#6183B6',strokeWidth:3 });

// restrict function between 2 values (-1,1)
var y = board.create('functiongraph', [function (x) { return
Math.sin(5*x)+Math.cos(2*x) },-1,1] );

// derivative of a graph object
var y2 = board.create('derivative', [y])

// derivative of a mathematical function

JXG.Math.Numerics.D(x**2)
```

Integral

Integral of the above y function: $y = \sin(5x) + \cos(2x)$

```
var y_integral = board.create('integral',[[-1, 1], y],{color:'#C3E4ED'} );
```

Point

Point 1 at (0,0) is not draggable. Point 2 at (1,1) is draggable

```
var point_1 = board.create('point',[0, 0],{ fixed:true})
var point_2 = board.create('point',[1, 1])
```

Segment

Line segment from point_1 at (0,0) to point_2 at (1,1)

```
var seg = board.create('segment', [point_1, point_2],{
strokeColor:'red'});
```

Arrow

Arrow from point_1 to point_2

```
var arrow = board.create('arrow', [point_1, point_2], {strokeColor:'red',
strokeWidth:3,fixed:true});
```

Arc

Arc centered at point_1, that starts at point_2 and ends at point_3

```
var arc = board.create('arc', [point_1, point_2, point_3],
{strokeColor:'red'});
```

Line

Solid line (line) and dashed line (dashed) that go to infinite and pass through coordinates (0,0) and (1,1)

```
var line = board.create('line',[[0,0],[1,1]],{color:'black',strokeWidth:2
})

var dashed = board.create('line',[[0,0],[1,1]],{color:'black',dash:2,
strokeWidth:2 })
```

Curve

```
X = [1,2,3,4,5]
Y = [9,8,6,5,1]
var y = board.create('curve', [X,Y],{strokeColor:'blue',strokeWidth:2});
```

Legend

Legend at position (100,100) with labels of ‘f(x)’ and ‘P(x)’ where f(x) has blue line and P(x) has orange line.

```
var legend = board.create('legend', [100,100], {labels:['f(x)','P(x)'],
colors: ['blue','orange'], strokeWidth:4} );
```

Circle

Circle with center at (0,0) and radius of 3

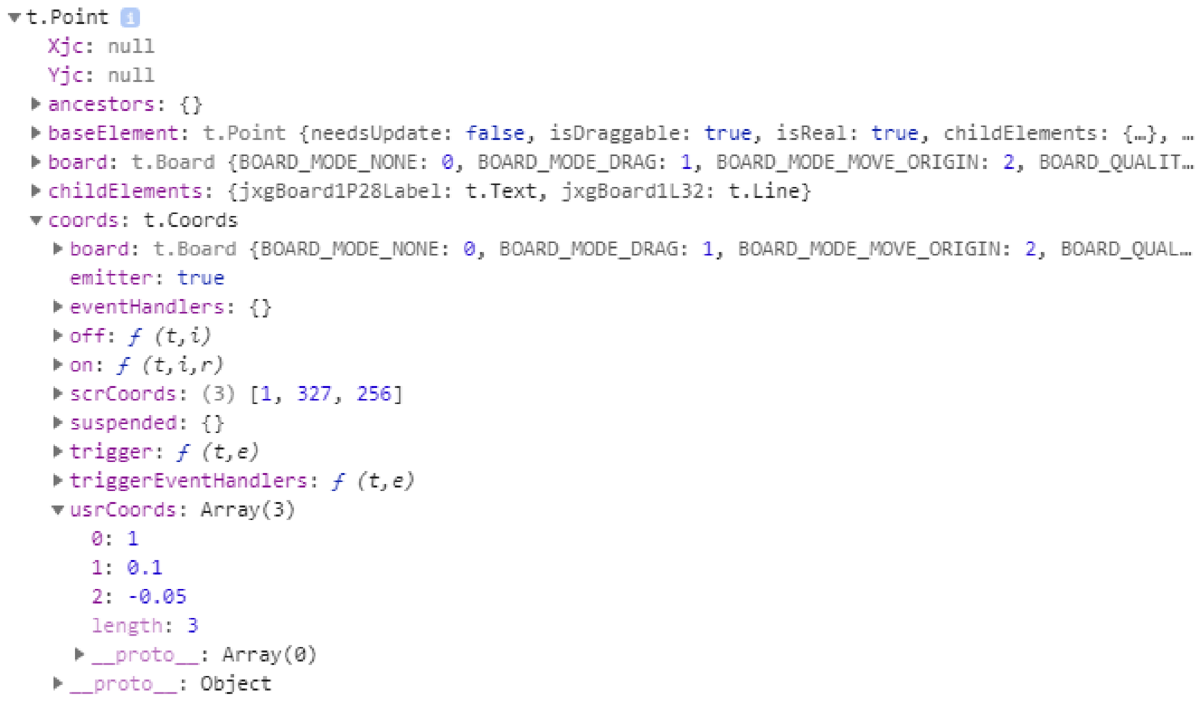
```
var circle = board.create('circle', [ [0,0] ,3],
{strokeColor:'black',strokeWidth:2,fixed:true})
```

Modify

The positions, style, and data of the above graphs and shapes can be modified by calling its variable. Then console.log the element to see all its attributes. Use *variable.attribute* to get an attribute.

After modifications to the graph, the board must be **updated**, simply write the line

```
board.update()
```



```
▼t.Point ℹ
   Xjc: null
   Yjc: null
  ▶ancestors: {}
  ▶baseElement: t.Point {needsUpdate: false, isDraggable: true, isReal: true, childElements: {…}, …
  ▶board: t.Board {BOARD_MODE_NONE: 0, BOARD_MODE_DRAG: 1, BOARD_MODE_MOVE_ORIGIN: 2, BOARD_QUALIT…
  ▶childElements: {jxgBoard1P28Label: t.Text, jxgBoard1L32: t.Line}
  ▼coords: t.Coords
    ▶board: t.Board {BOARD_MODE_NONE: 0, BOARD_MODE_DRAG: 1, BOARD_MODE_MOVE_ORIGIN: 2, BOARD_QUAL…
     emitter: true
    ▶eventHandlers: {}
    ▶off: ƒ (t,i)
    ▶on: ƒ (t,i,r)
    ▶scrCoords: (3) [1, 327, 256]
    ▶suspended: {}
    ▶trigger: ƒ (t,e)
    ▶triggerEventHandlers: ƒ (t,e)
    ▼usrCoords: Array(3)
       0: 1
       1: 0.1
       2: -0.05
       length: 3
      ▶__proto__: Array(0)
    ▶__proto__: Object
```

For example, to get the usrCoords attribute, use:

```
point_1.coords.usrCoords
```

Points can be moved to a new location using `.moveTo()`:

```
point_1.moveTo([2,2])   // moves point_1 to (2,2)
```

Common Attributes

Attributes are specifies in between curly brackets: { }, and separated by commas

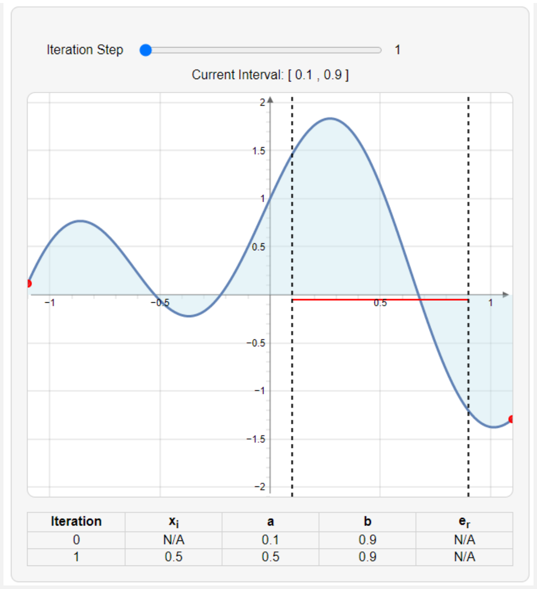
strokeColor: ‘red’ strokeColor: ‘#8B0000‘	Color of graph	fixed: true	Not draggable
strokeWidth: 3	Width of graph	dash:2	Dash line
visible: false	Turn off	Color: ‘red’	color

A simple example to control a point with a slider is as follows:

```
<!DOCTYPE html>
<html>
<head>
      <title></title>
      <!-- import JSXGraph -->
      <link rel="stylesheet" type="text/css" href="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraph.css" />
      <script type="text/javascript" src="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraphcore.js"></script>
</head>
<body>
      <!-- Graph Board -->
      <div id="box" class="jxgbox" style="width:500px; height:500px;
border: 1px solid rgb(210,210,210);"></div>
      <!-- Slider -->
      <input type="range" min=-5 max=5 value=0 id="slider">

      <script type="text/javascript">
            // initialize the board with y and y range of [-5,5]
            var board = JXG.JSXGraph.initBoard('box', {boundingbox: [-5,
5, 5, -5], axis:true,showCopyright: false});
            // create a fixed point at (0,0)
            var point_1 = board.create('point',[0, 0],{fixed:true})
            // add event listener to slider
            document.getElementById("slider").addEventListener("input",
                  function(){
                        // on input event, get slider's value
                        var value = document.getElementById("slider").value
                        // move point's x to slider value
                        point_1.moveTo([value,0])
                        // update board
                        board.update()
                  })
      </script>
</body>
</html>
```

Examples

Bisection Method



Iteration	x_i	a	b	e_r
0	N/A	0.1	0.9	N/A
1	0.5	0.5	0.9	N/A

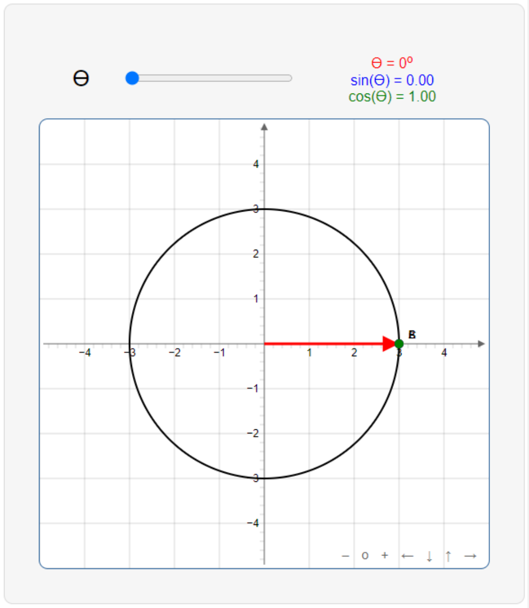
```
<!DOCTYPE html>
<html>
<head>
      <title></title>
      <link rel="stylesheet" type="text/css" href="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraph.css" />
      <script type="text/javascript" src="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraphcore.js"></script>
</head>
<body>
      <div class="container">
            <div class="block">
                  <label style="font-size: 16px; padding-right:
3%;">Iteration Step</label>
                  <!-- Slider -->
                  <input id="slider" type="range" min="1" max="12"
value="1" style="width: 50%;">
                  <div id="slider_value" style="padding-left: 2%;">1</div>

            </div>
            <div style="margin: 2%; margin-top: 0;">
                        Current Interval:
                        [ <p id="a" style="display: inline;">0.1</p> , <p
id="b" style="display: inline;">0.9</p> ]
```

```
            </div>
            <!-- GRAPH -->
            <div id="box" class="jxgbox" style="width:600px; height:500px;
border: 1px solid rgb(210,210,210);"></div>

            <!-- TABLE -->
            <br>
            <table id="dataTable">
                  <tr>
                        <td><strong>Iteration</strong></td>
                        <td><strong>x<sub>i</sub></strong></td>
                        <td><strong>a</strong></td>
                        <td><strong>b</strong></td>
                        <td><strong>e<sub>r</sub></strong></td>
                  </tr>
                  <tr>
                        <td>0</td>
                        <td>N/A</td>
                        <td>0.1</td>
                        <td>0.9</td>
                        <td>N/A</td>
                  </tr>
                  <tr>
                        <td>1</td>
                        <td>0.5</td>
                        <td>0.5</td>
                        <td>0.9</td>
                        <td>N/A</td>
                  </tr>
            </table>

      </div>

      <script type="text/javascript">

      var board = JXG.JSXGraph.initBoard('box', {boundingbox: [-1.1, 2.1,
1.1, -2.1], axis:true,showCopyright: false, showNavigation:false});

      var y = board.create('functiongraph', [function (x) { return
Math.sin(5*x)+Math.cos(2*x) }],
{strokeColor:'#6183B6',strokeWidth:3,baseLeft:false});

      var y_integral = board.create('integral', [[-10, 10], y],
{color:'#C3E4ED'} );
      y_integral.curveLeft.isDraggable = false;
      y_integral.curveRight.isDraggable = false;
      ///////////////////VARIABLEs////////////
      // where to draw red horizontal line
      var h_line_pos = -0.05;
      // a and b values
      var a = 0.1;
      var b = 0.9;
      var old_a = 0.5;
      var old_b = 0.9;

      var old_iteration = 1
      ///////////////////////

      // draw the horizontal red line
      var lowerPoint = board.create('point',[a,
h_line_pos],{visible:false,fixed:true})
      var upperPoint = board.create('point',[b,
h_line_pos],{visible:false,fixed:true})
      var seg = board.create('segment', [lowerPoint,
upperPoint],{fixed:true,strokeColor:'red'});

      var h_line_dict = {1:[lowerPoint,upperPoint,seg]} // dictionary to
track horizontal red lines

      // lower range line
      var lower =
board.create('line',[[0.1,0],[0.1,1]],{color:'black',dash:2,
strokeWidth:2,fixed:true})
      // upper range line
      var upper =
board.create('line',[[0.9,0],[0.9,1]],{color:'black',dash:2,
strokeWidth:2,fixed:true})

      // list of er values
      er = [0.28571,-0.16667,0.07692,0.03704,-
0.01887,0.00935,0.00465,0.00232,-0.00116,0.00058,0.00029]

      // Bisection method /////////////////////
      function f(x){
            return Math.sin(5*x)+Math.cos(2*x)
      }

      function bisection(a, b){
            // Step 2: Calculate the value of the root in iteration i as
x_i=(a_i+b_i)/2. Check which of the following applies:
            var x_i = (a+b)/2
            var er = 0 // error
            // 1. If f(x_i)=0, then the root has been found, the value of
the error er=0. Exit.
            if (f(x_i) == 0){
                  er = 0
            }
            // 2. If f(x_i)f(a_i)<0, then for the next iteration, x_{i+1}
is bracketed between a_i and x_i. The value of
\varepsilon_r=\frac{x_{i+1}-x_{i}}{x_{i+1}}.
            if (f(x_i)*f(a) < 0){
                  x_i_1 = (x_i + a)/2
                  // er = (x_i_1 - x_i)/x_i_1
                  return [a, x_i, x_i, er]
            }
            // 3. If f(x_i)f(b_i)<0, then for the next iteration, x_{i+1}
is bracketed between x_i and b_i. The value of
\varepsilon_r=\frac{x_{i+1}-x_{i}}{x_{i+1}}.
            if (f(x_i)*f(b) < 0){
                  x_i_1 = (x_i + b)/2
                  // er = (x_i_1 - x_i)/x_i_1
                  return [x_i,b, x_i, er]
            }
      }

document.getElementById("slider").addEventListener('input',function(){
            var iteration =
parseInt(document.getElementById("slider").value)
            document.getElementById('slider_value').innerHTML = iteration

            if (iteration > old_iteration){
                  // adding to figure
                  for (var i=old_iteration+1;i<=old_iteration+(iteration-
old_iteration);i++){
                        // generate new range of a and b
                        new_ab =
bisection(parseFloat(old_a),parseFloat(old_b))
                        // update a and b values
                        a = new_ab[0]
                        b = new_ab[1]
                        x_i = new_ab[2]
                        // add new row to table with current a and b values
                        newRow =
document.getElementById("dataTable").insertRow()
                        newRow.insertCell(0).innerHTML = i            //
cell 1: iteration
```

```
                        newRow.insertCell(1).innerHTML =
Number(x_i.toFixed(5))
                        newRow.insertCell(2).innerHTML =
Number(a.toFixed(5))
                        newRow.insertCell(3).innerHTML =
Number(b.toFixed(5))
                        newRow.insertCell(4).innerHTML = Number(er[i-
2].toFixed(5))


                        // update current a and b values with the previous
a and b value
                        document.getElementById("a").innerHTML =
Number(old_a.toFixed(5))
                        document.getElementById("b").innerHTML =
Number(old_b.toFixed(5))

                        // update range lines with previous a and b values
                        lower.point1.coords.usrCoords[1] = old_a
                        lower.point2.coords.usrCoords[1] = old_a
                        upper.point1.coords.usrCoords[1] = old_b
                        upper.point2.coords.usrCoords[1] = old_b

                        // draw the horizontal red line
                        var lowerPoint = board.create('point',[old_a, -
i*0.05],{visible:false,fixed:true})
                        var upperPoint = board.create('point',[old_b, -
i*0.05],{visible:false,fixed:true})
                        var h_line = board.create('segment', [lowerPoint,
upperPoint],{fixed:true,strokeColor:'red'});
                        // add points and line to dict
                        h_line_dict[i] = [lowerPoint, upperPoint, h_line]

                        // update old_a and old_b
                        old_a = new_ab[0]
                        old_b = new_ab[1]
                  }

            }else{
                  // removing from figure
                  for (var i=old_iteration;i>old_iteration+(iteration-
old_iteration);i--){
                        // remove last row
                        document.getElementById("dataTable").deleteRow(-1)
                        // remove red line and its endpoints of next
iteration
```

```
                       board.removeObject(h_line_dict[i])

                       board.update()
                 }
                 // update range lines
                 lower.point1.coords.usrCoords[1] =
h_line_dict[iteration][0].coords.usrCoords[1]
                 lower.point2.coords.usrCoords[1] =
h_line_dict[iteration][0].coords.usrCoords[1]
                 upper.point1.coords.usrCoords[1] =
h_line_dict[iteration][1].coords.usrCoords[1]
                 upper.point2.coords.usrCoords[1] =
h_line_dict[iteration][1].coords.usrCoords[1]
                 board.update()

                 // update a and b values
                 old_a = h_line_dict[iteration+1][0].coords.usrCoords[1]
                 old_b = h_line_dict[iteration+1][1].coords.usrCoords[1]

                 // update current a and b values
                 document.getElementById("a").innerHTML =
Number(h_line_dict[iteration][0].coords.usrCoords[1].toFixed(5))
                 document.getElementById("b").innerHTML =
Number(h_line_dict[iteration][1].coords.usrCoords[1].toFixed(5))

           }
           old_iteration = iteration

           board.update()
       })

     </script>

</body>

<style type="text/css">
      .container{
           margin: 2%;
           padding: 1%;
           background-color: rgb(246,246,246);
           border-radius: 10px;
           font-family: 'helvetica','calibri';
           width: max-content;
           border: 1px solid rgb(210,210,210);
           text-align: center;
      }
      .block{
           display: inline-flex;
           align-items: center;
           width: 100%;
           margin: 2%;
           margin-bottom: 0;
           padding: 2%;
      }
     table{
           width: 100%;
           text-align: center;
           border-collapse: collapse;

      }
     td{
           border: 1px solid rgb(210,210,210);
           width: 20%

      }
</style>
</html>
```

Trigonmetry Circle

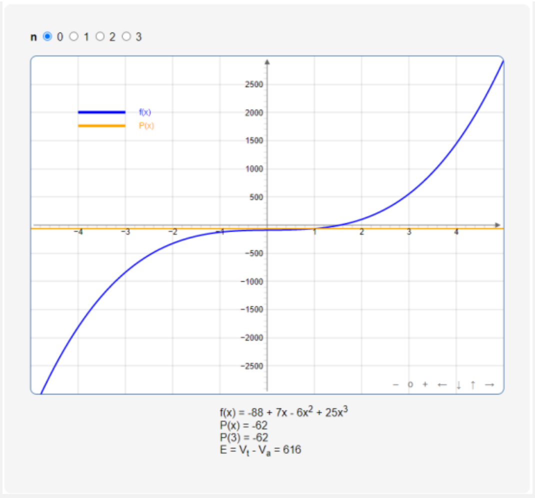


```
<!DOCTYPE html>
<html>
<head>
      <title></title>
      <!-- the link and script tags import the JSXGraph library, must
always have this here -->
      <link rel="stylesheet" type="text/css" href="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraph.css" />
      <script type="text/javascript" src="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraphcore.js"></script>
</head>
<body>

      <!-- This outer most div is the gray box and everything is contained
inside it -->
      <div style="margin: 2%; padding: 2%; background-color:
rgb(246,246,246);border-radius: 10px;font-family:
'helvetica','calibri';width: max-content;border: 1px solid
rgb(220,220,220);">
            <!-- this div is the theta, slider and text above the graph --
>
            <div style="display: inline-flex; align-items: center;width:
100%;text-align: center;" >
                  <!-- Theta label -->
                  <label style="font-size: 25px;flex:1;">&#1012;</label>
                  <!-- Slider -->
                  <input id="slider" type="range" style="flex: 2;" min="0"
max="360" value="0">
                  <!-- results text -->
                  <div id="results" style="margin:3%;flex: 2">
                        <div style="color:red">&#1012; =
0<sup>o</sup></div>
                        <div style="color:blue">sin(&#1012;) = 0.00</div>
                        <div style="color:green">cos(&#1012;) = 1.00</div>
                  </div>
            </div>

            <!-- GRAPH -->
            <div id="box" class="jxgbox" style="width:500px;
height:500px;"></div>

      </div>

      <!-- All javascript code inside this tag belo -->
      <script type="text/javascript">
            console.log("debug using console.log()")
            //// 1. Initialize an empty JSXGraph board/
            // first initialize a blank board
            var board = JXG.JSXGraph.initBoard('box', {boundingbox: [-5,
5, 5, -5], axis:true,showCopyright: false});
```

```
            /////2. plot the shapes onto the board
            // 1. outer circle, center at (0,0) and radius of 3
            outerCircle_center = [0,0]

      board.create('circle',[outerCircle_center,3],{strokeColor:'black',st
rokeWidth:2,fixed:true})
            // 2. arrow
            var arrow_endpoint = board.create('point',[3,0])
            var arrow = board.create('arrow', [[0,0],
arrow_endpoint],{strokeColor:'red',strokeWidth:3,fixed:true});
            // 3. vertical dashed line
            var verticalPoint = board.create('point',[3,
0],{color:'blue',fixed:true})
            var verticalLine = board.create('segment', [arrow_endpoint,
verticalPoint],{fixed:true,dash:2,strokeColor:'blue'});
            // 4. horizontal dashed line
            var horizontalPoint =
board.create('point',[3,0],{color:'green',fixed:true})
            var horizontalLine = board.create('segment', [arrow_endpoint,
horizontalPoint],{fixed:true,dash:2,strokeColor:'green'});
            // 5. inner arc
            const arc_center =
board.create('point',[0,0],{fixed:true,visible:false})
            const arc_start =
board.create('point',[1,0],{fixed:true,visible:false})
            var arc_angle =
board.create('point',[1,0],{fixed:true,visible:false})
            var arc = board.create('arc', [arc_center, arc_start,
arc_angle],{strokeColor:'red'});

            ////// Link the slider to the graph

             // EventListener listen to changes of slider and runs
function() in between {} on change

document.getElementById("slider").addEventListener('input',function(){
                  // MATH: circle is x^2+y^2=9
                  var angle = document.getElementById("slider").value
                  // update x and y value of endpoint, x=3cos(angle),y
=3sin(angle)

      arrow_endpoint.moveTo([3*Math.cos(angle*(Math.PI/180)),3*Math.sin(an
gle*(Math.PI/180))])
```

```
                 // update vertical line

      verticalPoint.moveTo([arrow_endpoint.coords.usrCoords[1],0])
                 // update horizontal line

      horizontalPoint.moveTo([0,arrow_endpoint.coords.usrCoords[2]])
                 // update arc

      arc_angle.moveTo([Math.sqrt(1.5)*Math.cos(angle*(Math.PI/180)),Math.
sqrt(1.5)*Math.sin(angle*(Math.PI/180))])
                 board.update() // update JSXGraph

                 // update the text
                 document.getElementById("results").innerHTML = `
            <div style="color:red">&#1012; = `+angle+`<sup>o</sup></div>
           <div style="color:blue">sin(&#1012;) =
`+Math.sin(angle*(Math.PI/180)).toFixed(2)+`</div>
           <div style="color:green">cos(&#1012;) =
`+Math.cos(angle*(Math.PI/180)).toFixed(2)+`</div>`
            })

      </script>

</body>
</html>
```

Taylor Series


n [✓ 0] 1 2 3

$f(x) = -88 + 7x - 6x^2 + 25x^3$
$P(x) = -62$
$P(3) = -62$
$E = V_t - V_a = 616$

```
<!DOCTYPE html>
<html>
<head>
```

```
      <title></title>
      <link rel="stylesheet" type="text/css" href="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraph.css" />
      <script type="text/javascript" src="https://jsxgraph.uni-
bayreuth.de/distrib/jsxgraphcore.js"></script>
</head>
<body>

      <div style="margin: 2%; padding: 2%; background-color:
rgb(245,245,245);border-radius: 10px;font-family:
'helvetica','calibri';width: max-content;">
            <!-- SELECTION BUTTONS -->
            <strong>n </strong>
            <input type="radio" id="0" name="taylor" value="0" checked>
                  <label for="0">0</label>
            <input type="radio" id="1" name="taylor" value="1">
                  <label for="1">1</label>
            <input type="radio" id="2" name="taylor" value="2">
                  <label for="2">2</label>
            <input type="radio" id="3" name="taylor" value="3">
                  <label for="3">3</label>
            <br><br>

            <!-- GRAPH -->
            <div id="box" class="jxgbox" style="width:700px;
height:500px;"></div>

            <!-- RESULTS TEXT -->
            <div id="results" style="margin:2%;margin-left: 40%;">
                  <div>f(x) = -88 + 7x - 6x<sup>2</sup> +
25x<sup>3</sup></div>
                  <div>P(x) = -62</div>
                  <div>P(3) = -62</div>
                  <div>E = V<sub>t</sub> - V<sub>a</sub> = 616</div>
            </div>
      </div>

      <script type="text/javascript">
            // first initialize a blank board
             var board = JXG.JSXGraph.initBoard('box', {boundingbox: [-5,
3000, 5, -3000], axis:true,showCopyright: false});

             // generate x data
             x_data = []
             main_plot = []
             estimate_0 = []
             estimate_1 = []
             estimate_2 = []

             for (var x=-5;x<5;x+=0.1){
                  x_data.push(x);
                  main_plot.push(25*(x**3)-6*(x**2)+7*x-88)
                  estimate_0.push(-62)
                  estimate_1.push(-132+70*x)
                  estimate_2.push(-63-68*x+69*(x**2))
             }
             // plot the curves
             board.create('curve',
[x_data,main_plot],{strokeColor:'blue',strokeWidth:2});
             var graph = board.create('curve',
[x_data,estimate_0],{strokeColor:'orange',strokeWidth:2});

             // create the legend
             board.create('legend', [-4, 2000], {labels:['f(x)','P(x)'],
colors: ['blue','orange'], strokeWidth:4} );

             // listen to changes of radio button

document.getElementById("0").addEventListener('click',function(){
                  graph.dataY = estimate_0
                  board.update()
                  document.getElementById("results").innerHTML = `
                                                             <div>f(x) = -88 +
7x - 6x<sup>2</sup> + 25x<sup>3</sup></div>
                                                             <div>P(x) = -
62</div>
                                                             <div>P(3) = -
62</div>
                                                             <div>E =
V<sub>t</sub> - V<sub>a</sub> = 616</div>`
             })

document.getElementById("1").addEventListener('click',function(){
                  graph.dataY = estimate_1
                  board.update()
                  document.getElementById("results").innerHTML = `
                                                             <div>f(x) = -88 +
7x - 6x<sup>2</sup> + 25x<sup>3</sup></div>
                                                             <div>P(x) = -132
+ 70x</div>
                                                             <div>P(3) =
78</div>
```

```
                                                                <div>E = 
V<sub>t</sub> - V<sub>a</sub> = 476</div>`
             })

document.getElementById("2").addEventListener('click',function(){
                  graph.dataY = estimate_2
                  board.update()
                  document.getElementById("results").innerHTML = `
                                                                <div>f(x) = -88 + 
7x - 6x<sup>2</sup> + 25x<sup>3</sup></div>
                                                                <div>P(x) = -63 - 
68x + 69x<sup>2</sup></div>
                                                                <div>P(3) = 
354</div>
                                                                <div>E = 
V<sub>t</sub> - V<sub>a</sub> = 200</div>`
             })

document.getElementById("3").addEventListener('click',function(){
                  graph.dataY = main_plot
                  board.update()
                  document.getElementById("results").innerHTML = `
                                                                <div>f(x) = -88 + 
7x - 6x<sup>2</sup> + 25x<sup>3</sup></div>
                                                                <div>P(x) = -88 + 
7x - 6x<sup>2</sup> + 25x<sup>3</sup></div>
                                                                <div>P(3) = 
554</div>
                                                                <div>E = 
V<sub>t</sub> - V<sub>a</sub> = 0</div>`
             })
      </script>
</body>
</html>
```

References

[1] "w3schools," [Online]. Available: https://www.w3schools.com/. [Accessed 18 August 2020].

[2] "JSXGraph," [Online]. Available: https://jsxgraph.uni-bayreuth.de/wiki/index.php/Documentation. [Accessed 18 August 2020].