Two gnomonics libraries for Javascript

by Steve Lelievre

This document was revised on 2021-08-07.

These libraries are intended for use in webpages – whether server-hosted or local HTML files – that use Javascript as a programming language. The library *gMath.js* is for performing trigonometry using degrees instead of Javascript's usual radians. The library *gPlot.js* is for drawing using Scalable Vector Graphics (SVG). It can be used to create lines, polygons, circles, circular arcs, ellipses, elliptical arcs, text, and curved text. There are facilities for printing and saving a drawing to file.

gMath.js

At the time of writing, this library contains most of the trig functions from the *gmath* add-on for Excel, but none of the other Excel *gmath* functions. To use *gMath.js*, precede your own script with

```
<script src='http://www.gnomoni.ca/gMath.js'></script>
```

Function	Purpose
rads(x)	Degrees to radians
degrees(x)	Radians to degrees
sin(x)	Sine
cos(x)	Cosine
tan(x)	Tangent
asin(x)	Arcsine
acos(x)	Arccosine
atan(x)	Arctangent
atan2(y, x)	Quadrant arctangent
abs(x)	Mathematical absolute
floor(x)	Mathematical floor
ceil(x)	Mathematical ceiling
round(x)	Rounds a number to the nearest integer

gPlot.js

Overview

This library wraps some of the standard HTML and Javascript features relating to use of SVG for drawing. My first aim was to provide a simplified and fairly minimal SVG drawing library for my own use. I then generalized it slightly, with the idea of making it suitable for other dialists to use as well.

To use the *gPlot.js* library, precede your own script with

```
<script src='http://www.gnomoni.ca/gPlot.js'></script>
```

If you need advanced SVG features, you should use any of the more sophisticated libraries that are readily found by searching the web.

The gPlot.js library simply adds elements to an SVG area but it does not rely exclusive access to the area. Hence, the area can contain other elements, created by means of HTML tags.

There are 17 procedures in gPlot.js. It is important to note that, unlike gMath.js, these are not called directly as functions. Instead, they are ‘methods’ of a Javascript object. If you are not familiar with the concept of methods, please check the web for information about Encapsulation as it relates to Object Oriented Programming e.g. see https://youtu.be/pTB0EiLXUC8. As well, review my sample program (available at http://gnomoni.ca/gPlotDemo.htm or distributed with this document).

Declaring the SVG area

Your page’s HTML should declare an SVG element looking something like this:

```
<svg id = ‘svg’ width = ‘278mm’ height = ‘210mm’></svg>
           1                2                3
```

Box 1 contains the element identifier, which must be given as a parameter to Plot.

The SVG width and height define the screen area that will be used to show the drawing.[1]

Boxes 2 and 3 must be the same, and are units of measurement: in (inches), cm or mm. They define the unit of measurement that will be used for the SVG diagram on screen as well as the units applied to a drawing saved to a file. If you do not supply a unit, the drawing will be done using pixels, and a drawing saved to file will be sized at 96 pixels per inch.

SVG normally has the top left corner as the origin of the coordinate system, with the *y*-axis increasing down the page. When you use the gPlot.js library, *y* increases up the page. As well, the origin can be positioned by the user – by default it will be in the center of the drawing area; for drawings such as vertical dials it may be more convenient to place the origin at the middle top.

Parameter passing

Javascript does not allow parameter passing by name, only by position. Hence, parameters can only be left off if they come the end of the list. In practice, the simplest way around this is to choose an actual value for all intermediate parameters. Alternatively, give the special value *undefined* (not in quotes) to make Javascript use the default value for any parameter that is not at the end of the parameter list.

Default Appearance

There are built-in initial values for things like line and text color, font size, line thickness and so on.

To change these settings use *paintAppearance* or *textAppearance* as applicable. Be aware that *gPlot.js* does no parameter validation – pay attention to case and, in particular, the use of camelCase.

As well as updating the saved settings, you have the option to bypass the current settings whenever you use create SVG objects or text. This is done by supplying an appearance parameter to the applicable method.

Clipping

Any parts of your drawing that extend beyond the edge of the specified SVG area are not clipped. The extraneous parts will not be shown by your browser, but if you save SVG diagram to a file and then view it in Inkscape or other software, you may find that the drawing extends beyond the edges of the page.

[1] The values of 278 and 210 (mm) used in the above example are a special case: the largest rectangle that just fits both A4 and Letter paper without margins. This makes is suitable for applications that could be used in any country. In practice, using a slightly smaller size of 266 mm × 196 mm would be even better. The latter values allow space for a 7 mm (slightly over ¼″) margin on all sides, and thus should suit nearly all modern consumer printers. Hence, 266 mm × 196 mm is my recommendation for programs that you intend to make widely available and which will be used for producing printouts.

Definitions

In two cases, the library relies on parameters passed as Javascript variables of type *object*. The following terms[2] are used in this document to indicate these types of parameters:

- a *point* refers to a Javascript object consisting of an x and y coordinate, e.g. *{x: 0, y: 0}*,
- an *appearance* refers to a Javascript object containing a set of properties that determine the appearance of shapes and lines, or text. e.g. *{strokeWidth: 0.01, stroke: 'grey', fill: 'yellow'}*. The supported properties and their meaning are defined in Appendix 1.

The gPlot.js methods

Method (function)	Parameter	Purpose	Default
Plot		Creates the Javascript object that you will use for manipulating your SVG drawing. Normally, *Plot* is used once, at the start of a program, and only with the *new* operator (more technically, it is a 'constructor'). It must be called before any other *gPlot.js* method. For example: `drawing = new Plot('svg');` The Javascript object created was called `drawing` in this example. Hence, all following examples are of the form `drawing.methodName();`	
	svgID	The Element ID of an existing SVG element on the webpage that is to be used as the drawing area.	Required
	xPosition	How far from the left side of the drawing area to place the origin. It must be a positive number. This parameter is ignored if a *viewbox* is defined as part of the HTML SVG declaration.	It is placed half way across
	yPosition	How far from the top of the drawing area to place the origin. It must be a positive number. This parameter is ignored if a *viewbox* is defined as part of the HTML SVG declaration.	It is placed half way down
circle		Draws a circle. For example, a circle of radius 50 centered at (20, 50): `drawing.circle(50, {x : 20, y : 50});`	
	radius	Radius.	Required
	center	A point object.	{x: 0, y: 0}
	appearance	An appearance object.	{}
circularArc		Adds a circular arc, drawn clockwise. For example, `drawing.circularArc(50, -10, 10);`	
	radius	A number representing the radius of the arc.	Required
	angle1	The start angle increasing clockwise from top.	Required
	angle2	The end angle increasing clockwise from top.	Required
	closed	Boolean to indicate if the ends of the arc are to be connected to the center to make the shape of a 'slice of pie.'	false
	center	A point object representing the center of curvature of the arc.	{x: 0, y: 0}
	appearance	An appearance object. The *fill* attributes are ignored if the arc is not closed.	{}

[2] Please note that these definitions of *point* and *appearance* relate to this document only – these are not general definitions relating to either Javascript or SVG.

Method (function)	Parameter	Purpose	Default
clear	Clears the entire diagram of objects created using the library. Use this method if a change of user inputs means that the drawing must be done over. For example: `drawing.clear();` This method does not take any parameters.		
ellipse	Draws an ellipse. For example, with semi-major axis of 20 and semi-minor axis of 10, centered at (20, 50) and the whole object rotated by 30 degrees clockwise about its center: `drawing.ellipse(20, 10, {x : 20, y : 50}, 30);`		
	radius1	Semi-major axis length.	Required
	radius2	Semi-minor axis length.	Required
	center	A point object.	{x: 0, y: 0}
	rotation	A number representing the angle, in degrees, through which the ellipse is rotated (positive clockwise).	0
	appearance	An appearance object.	{}
ellipticalArc	Adds an elliptical arc, drawn clockwise. For example, `drawing.ellipticalArc(50, 30, -10, 10, true, -45);`		
	radius1	Semi-major axis length.	Required
	radius2	Semi-minor axis length.	Required
	angle1	The start angle increasing clockwise from top (before any rotation).	Required
	angle2	The end angle increasing clockwise from top (before any rotation).	Required
	closed	Boolean to indicate if the ends of the arc are to be connected to the center to make the shape of a 'slice of pie.'	false
	center	A point object representing the center of curvature of the arc.	{x: 0, y: 0}
	rotation	A number representing the angle, in degrees, through which the ellipse is rotated (positive clockwise).	0
	appearance	An appearance object. The *fill* attributes are ignored if the arc is not closed.	{}
line	Draws a line between 2 points. For example, a straight red line from the origin to a point at (20, 50): `drawing.line({x : 0, y : 0}, {x : 20, y : 50}, {stroke: 'red'});`		
	start	A point object.	Required
	end	A point object.	Required
	appearance	An appearance object.	{}
multiLine	Draws a multipoint line. For example, a red triangle: `drawing.multiLine([{x : 0, y : 0}, {x : 20, y : 50}, {x : 10, y: 30}], true, {stroke: 'red'});`		
	points	An array of point objects.	Required
	closed	Boolean to indicate whether the path is closed.	false
	appearance	An appearance object.	{}
paintAppearance	Updates the defaults used for the appearance of lines and shapes. See Appendix 1. For example, `drawing.paintAppearance({stroke: 'red', strokeWidth: 0.2});`		
	appearance	An appearance object.	{}
polygon	To draw polygons other than rectangles, use the *multiLine* method.		

Method (function)	Parameter	Purpose	Default
print	Prints the drawing. Note that an additional browser window is opened by this operation; it is removed on completion. Usage example: `drawing.print();` This method does not take any parameters.		
rectangle	Draws a rectangle. For example, a 60 × 30 rectangle centered at (100, 0) and rotated 15 degrees about its center: `drawing.rectangle(60, 30, {x: 100, y: 0}, 15);`		
	width	Width of the rectangle	Required
	height	Height of the rectangle	Required
	center	A point indicating the position of the center of the rectangle	{x: 0, y: 0}
	rotation	A number representing the angle, in degrees, through which the rectangle is rotated (positive clockwise).	0
	appearance	An appearance object.	{}
roundedRectangle	Draws a rectangle with rounded corners. For example, a 40 × 30 with corner radius of 5, centered the origin: `drawing.roundedRectangle(40, 30, 5);`		
	width	Width of the rectangle	Required
	height	Height of the rectangle	Required
	radius	Radius of the corner arcs	0
	center	A point indicating the position of the center of the rectangle	{x: 0, y: 0}
	rotation	A number representing the angle, in degrees, through which the rectangle is rotated (positive clockwise).	0
	appearance	An appearance object.	{}
save	Saves the drawing to a file using SVG format. For example, `drawing.save('dialFace.svg');`		
	filename	Filename as a string.	'gPlot.svg'
	toInkscape	Boolean indicating that the saved SVG is to be regressed to an older version of SVG that is compatible with Inkscape. Use a parameter value of true if curved text is being drawn straight in Inkscape.	false
square	To draw squares, use the *rectangle* or *roundedRectangle* method.		
text	Adds a line of text anchored (placed) at the specified point. For example, `drawing.text('Hello, world!', {x: 100, y: 100});`		
	wording	The wording.	Required
	position	The placement point expressed as a Javascript object.	Required
	anchor	The text's anchor indicates which part of the text ('left', 'middle', or 'right') is placed at the coordinate given by the position parameter.	'middle'
	rotation	A number representing the angle, in degrees, by which the text is rotated (positive clockwise).	0
	appearance	An appearance object.	{}
textAppearance	Updates the defaults used for the appearance of text. See Appendix 1. For example, `drawing.textAppearance({fontFamily: 'courier', textColor: 'blue'});`		
	appearance	An appearance object.	{}

Method (function)	Parameter	Purpose	Default
textOnCircle	Places curved text on an invisible circular arc of the specified radius, anchored at the middle of the text. For example, `drawing.textOnCircle('XII', 50);`		
	wording	The wording.	Required
	radius	A number representing the radius of an invisible arc that the text sits on.	Required
	offset	The text is offset by an angle measured clockwise from top.	0
	outside	Boolean to indicate if the text appears on the outside or inside of the arc.	true
	center	A point object representing the center of the invisible arc.	{x: 0, y: 0}
	appearance	An appearance object.	{}
textOnEllipse	Places curved text on an invisible elliptical arc of the specified dimensions, anchored at the middle of the text. For example, `drawing.textOnEllipse('XII', 50);`		
	wording	The wording.	Required
	radius1	Semi-major axis length.	Required
	radius2	Semi-minor axis length.	Required
	offset	The text is offset by an angle measured clockwise from top (before any rotation).	0
	outside	Boolean to indicate if the text appears on the outside or inside of the arc.	true
	center	A point object representing the center of the invisible arc.	{x: 0, y: 0}
	rotation	A number representing the angle, in degrees, through which the invisible ellipse is rotated (positive clockwise).	0
	appearance	An appearance object.	{}
triangle	To draw triangles, use the *multiLine* method.		

Appendix 1: Appearance Objects

Each appearance object property corresponds to an SVG attribute. Refer to websites for detailed explanations of purpose and allowed values. Once such website is https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute. Note that only a small subset of SVG attributes is supported in *gPlot.js* and the defaults may not be the same as in CSS.

The initial default values are changed by use of *paintAppearance* and *textAppearance*.

For shapes and lines

appearance property	Corresponding SVG attribute	Notes	Initial default
dashes	stroke-dasharray	A string containing a list of alternating dash and gap lengths, separated by spaces or commas. If the list has an odd number of lengths, the list is automatically repeated to get an even number. Hence for example, '4' is treated as '4 4' meaning a dash length of 4 is followed by a gap length of 4.	A continuous stroke is used.
fill	fill	Any CSS color.	'none'

appearance property	Corresponding SVG attribute	Notes	Initial default
fillOpacity	fill-opacity	As opacity, but for the fill only.	1
lineCaps	stroke-linecap	Any CSS line cap.	‘round’
lineJoins	stroke-linejoin	Any CSS line join value.	‘round’
opacity	opacity	A number ranging from 0 (fully transparent) to 1 (fully opaque). This property combines fillOpacity and StrokeOpacity	1
stroke	stroke	Any CSS color.	‘black’
strokeOpacity	stroke-opacity	As opacity, but for the stroke only.	1
strokeWidth	stroke-width	Sets line width, using drawing’s units.	0.1

For text

appearance property	Corresponding SVG attribute	Notes	Initial default
fontFamily	font-family	Any CSS font family.	‘Times New Roman’
fontSize	font-size	Sets font size using drawing’s units. If you want to use typesetting points (pt), convert from 72 pt per inch. For example, to have text at 12 pt in a drawing using millimeters for the units, use a value of 12 × 25.4 ÷ 72 for fontSize (25.4 mm per inch, 72 pt per inch.)	$4.233^{\cdot}$ (12 × 25.4 ÷ 72)
fontStretch	font-stretch	Any CSS font stretch.	‘normal’
fontStyle	font-style	Any CSS font style.	‘normal’
fontVariant	font-variant	Any CSS font variant	‘normal’
fontWeight	font-weight	Any CSS font weight.	‘normal’
letterSpacing	letter-spacing	Any CSS letter spacing.	‘normal’
opacity	opacity	A number ranging from 0 (fully transparent) to 1 (fully opaque). This property combines textOpacity and textOutlineOpacity.	1
appearance property	Corresponding SVG attribute	Notes	Initial default
textColor	fill	Any CSS color.	‘black’
textOpacity	fill-opacity	As opacity, but for the text color only.	1
textOutline	stroke	Any CSS color. Note, SVG text usually has no outline.	‘none’
textOutlineOpacity	stroke-opacity	As opacity, but for the text outline only.	1

textOutlineWidth	stroke-width	Sets line width, using drawing’s units. Note, SVG text usually has no outline.	0
wordSpacing	word-spacing	Any CSS word spacing.	‘normal’