Two gnomonic libraries for JavaScript

by Steve Lelievre

This document was revised on 2023-12-08.

These libraries are intended to help programmers produce simple browser-based sundial software using only HTML and JavaScript. Using the browser to run programs removes the need for you, or other users of your software, to install applications. The same version of your program works on any operating system, and irrespective of whether the program is stored on your computer's disk or a website.

The library *gMath.js* is the simpler and smaller of the two libraries. It allows trigonometry using degrees instead of JavaScript's usual radians, and offers a few other basic math functions that may be helpful in a sundial program.

The library *gPlot.js* is for drawing. It can be used to create lines, polygons, circles, circular arcs, ellipses, elliptical arcs, text, and curved text[1]. There are facilities for printing and saving a drawing to disk.

Because the output is in Scalable Vector Graphics format (SVG), it can easily be transferred to your favorite drawing package for further processing.

These libraries do not have functions that carry out gnomonic calculations for you so, for instance, there are no implementations of formulae for solar calculations. The idea is exactly that you can concentrate on calculations using your sundialing knowledge without needing to learn all the complexities of how one draws on a webpage with native SVG.

Your JavaScript program can be contained in the same disk file as the HTML or in a separate file. If you are new to HTML and JavaScript, you will likely find it easiest to get started by using a very simple HTML file (no more than a shell) and put the program code in a separate JavaScript source file, like this:

myDial.htm	myDial.js
This is the file you open with your browser. The example is a shell for a blank webpage – the shortest possible file needed with *gPlot.js*. The corresponding *.js* file contains the program that tells the browser how to draw a diagram onto the blank webpage. The highlighted part has to match the name (and path if applicable) of the file containing your JavaScript.	This is the file that contains the JavaScript program. The example starts by using standard JavaScript (not part of *gPlot.js*) to set the title of the browser tab. It then creates an empty drawing and adjusts the size. It adds two circles (one centered at the origin and one not), a colored square (rotated by 10 degrees), and some text.

<!DOCTYPE html>
<html lang='en'><head><meta charset='utf-8'>
<script src='https://www.gnomoni.ca/gMath.js'></script>
<script src='https://www.gnomoni.ca/gPlot.js'></script>
<title></title>
</head><body></body>
<script src = 'myDial.js'></script>
</html>
	window.title = 'My Dial Program';
	let drawing = new Plot();
	drawing.setSizeAndOrigin('266mm', '196mm');
	drawing.circle(96);
	drawing.circle(80, {x:15, y: 5});
	drawing.rectangle(50, 50, auto, 10, {fill:'#f0c0c0', stroke: 'none'});
	drawing.text("Hello, world!", {x: 0, y: 60}, 'middle', auto, {fontSize: 20});

For a more realistic but still relatively simple example, see the appendix. It contain a short program for creating a Horizontal Dial, with annotations.

[1] Because *gPlot.js* is just a set of 'wrappers' for standard Javascript/SVG commands, it does not take away your ability to use HTML, SVG tags, and regular Javascript facilities on the same drawing. That is, if you need to do something fancy, you can access the SVG object directly to add elements not provided by the library. Likewise, you could also create some of the drawing by using the applicable tags in HTML and then use *gPlot.js* to add the rest.

gMath.js

To use *gMath.js*, your HTML must include the statement

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

Function	Purpose		
pi	A predefined constant – not a function – with the value of π.		
rads(x)	Degrees to radians		
degrees(x)	Radians to degrees		
sin(x)	Sine of degrees		
cos(x)	Cosine of degrees		
tan(x)	Tangent of degrees		
asin(x)	Arcsine as degrees		
acos(x)	Arccosine as degrees		
atan(x)	Arctangent as degrees		
atan2(y, x)	Quadrant arctangent as degrees		
abs(x)	Mathematical absolute		
floor(x)	Mathematical floor		
ceil(x)	Mathematical ceiling		
round(x)	Rounds a number to the nearest integer		

polynomial(x, coefficients)	Evaluates[2] a polynomial, e.g.
	`p = polynomial(2.4, [7, 19.6, 0.1, 66.5, 4]);`
	is equivalent to
	`p = 7 + 19.6 * 2.4 + 0.1 * 2.4 ** 2 + 66.5 * 2.4 ** 3 + 4 * 2.4 ** 4;`		

	x	The value of the polynomial's variable.	Required
	coefficients	An array of coefficients [a, b, c, d, …] for a polynomial of the form $ax^0 + bx^1 + cx^2 + dx^3 \ldots$	Required

[2] The function implements Horner's Method. Calculations based on Jean Meeus' *Astronomical Algorithms* can safely use Horner's Method rather than summing exponentiated terms; in fact, Meeus recommends it. Bearing in mind that some of the algorithms in the book involve polynomials with terms up to the tenth power, the main advantage of this function is conciseness and clarity of the source code. There is a slight performance advantage as well.

gPlot.js

Overview

This library provides a basic subset of SVG drawing facilities, chosen based on the needs of dialists. To use the *gPlot.js* library, your HTML must include the statement

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

There are 18 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 (OOP). There is a helpful video introduction to OOP at https://youtu.be/pTB0EiLXUC8.

Glossary

In two cases, the library relies on parameters passed as JavaScript variables of type *object*. These terms[3] indicate specific cases:

- a *point object* refers to a JavaScript object consisting of an *x* and a *y* coordinate, e.g. *{x: 0, y: 0}*
- an *appearance object* refers to a JavaScript object containing a set of properties that determine the appearance of shapes and lines, or of text. e.g. *{strokeWidth: 0.01, stroke: 'grey', fill: 'yellow'}*.

Creating the SVG drawing area

Approach 1

Your program's first invocation of *gPlot.js* function is always a call to the Plot function. If you do not identify a pre-existing SVG area (SVG tag in your HTML), then Plot creates the SVG area for you.

This approach is intended to be the 'easy' option, suited to people who wish to minimize or avoid HTML programming. You need only a very simple HTML document, as seen in the following example:

Approach 2

The other way to define the SVG area is by including the applicable tags in your HTML. Although this was the only approach available in the original version of *gPlot.js*, I do not recommend it – unless or until you are comfortable using the full features of SVG and HTML.

Your page's HTML would declare an SVG element looking something like this:

```
<svg id = 'svg' width = '266mm' height = '196mm'></svg>
```

1 2 3

Box 1 contains the element identifier, which must be given as the first parameter to your call of the *gPlot.js* Plot function.

Next come SVG width and height values that define the screen area used for the drawing.[4]

Boxes 2 and 3 must be the same, and are units of measurement, for example: cm, mm, px, or in (inches). 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 printed out or saved as a file will be sized at 96 pixels per inch (approximately 3.78 pixels per mm.)

---

[3] These definitions of *point* and *appearance* relate to this document only – these are not general definitions from the SVG specification.

[4] 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.

The origin and coordinate system

SVG normally has the top left corner as the origin of the coordinate system, with the *y*-axis increasing down the page. One of the adaptions made in *gPlot.js* is to have *y* increase as you move up the page, as usual for Cartesian coordinates. As well, you can choose the position of the origin within the drawing area. The default position is the center of the drawing area; for drawings such as vertical dials it may be more convenient to place the origin explicitly, such as near the middle top.

Parameter Passing

JavaScript does not allow parameter passing by name, relying instead on position. This means you can not have gaps in the list of parameters you provide (but you can leave of the parameters at the end of list if they have defaults defined.) The simplest approach is to specify actual value for parameters but you can also use the special value *undefined* at the applicable position in the list if you want JavaScript take the default value (if defined) of the parameter. With *gPlot.js* it is possible to use *auto* as a synonym for *undefined*.

Default Appearance

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

To change the defaults, 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 override the current default appearance for individual elements of your SVG diagram. This is done by supplying an appearance parameter to the applicable method.

Cropping

Starting with the December 2023 revisions, any parts of your drawing that extend beyond the edge of the specified drawing area are automatically cropped away by *gPlot.js*.

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 only once, at the start of a program, and only with the *new* operator (more technically, it is behaving as a 'constructor'). It must be called before any other *gPlot.js* method. For example: `drawing = new Plot();` The Javascript object created was called `drawing` in this example. Hence examples given for other functions listed below are of the form `drawing.methodName();`		
	svgID	The Element ID of an SVG element on the webpage that is to be used as the drawing area. If there is an existing SVG element with the applicable ID, it will be used. Otherwise, an element will be added to the web page.	'svg'
	xPosition	How many units 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 an HTML SVG declaration.	It is placed half way across
	yPosition	How many units 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 an HTML SVG declaration.	It is placed half way down
	width	The width of the drawing area. This parameter is ignored if an existing SVG element is found. If no units are given, *px* is assumed.	1000px
	height	The height of the drawing area. This parameter is ignored if an existing SVG element is found. The units used must match the width parameter's units.	1000px

Method (function)	Parameter	Purpose	Default
setSizeAndOrigin	Changes the size and/or origin of a drawing area. For example `drawing. setSizeAndOrigin = function(width = '1000mm', height = '1000mm', xPos = -1, yPos = -1)` Note: for reasons of backward compatibility, the parameter order does not follow the equivalents from the Plot function.		
	width	The width of the drawing area. If no units are given, *px* is assumed.	1000px
	height	The height of the drawing area. Units must match the width parameter.	1000px
	xPosition	How many units from the left side of the drawing area to place the origin. It must be a positive number.	It is placed half way across
	yPosition	How many units from the top of the drawing area to place the origin. It must be a positive number.	It is placed half way down
In the remaining gPlot.js functions, units must not be included when specifying lengths or distances. The units used to define the drawing width and height will always apply.			
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.	{}
clear	Clears the entire diagram of any 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. Objects created in HTML are not removed.		
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*

Method (function)	Parameter	Purpose	Default
	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. For details, see the later Appearance Objects section. For example, `drawing.paintAppearance({stroke: 'red', strokeWidth: 0.2});`		
	appearance	An appearance object.	{ }
polygon	To draw polygons other than rectangles, use the *multiLine* method.		
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*

Method (function)	Parameter	Purpose	Default
	height	Height of the rectangle	*Required*
	center	A point object indicating 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 object indicating 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	A point 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. The values 'center' and 'centre' may be used in place of 'middle'.	'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. For details, see the later Appearance Objects section. For example, `drawing.textAppearance({fontFamily: 'courier', textColor: 'blue'});`		
	appearance	An appearance object.	{}
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

Method (function)	Parameter	Purpose	Default
	center	A point object indicating 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 indicating 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.		

Appearance Objects

Each appearance object property corresponds to an SVG attribute, using values allowed by the Cascading Style Sheet definition (CSS). 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 colors, and CSS value can be used, such as 'red', 'black', etc., with the special value 'none' indicating absence. Coded values such as 'rgb(128,128,128)' or '#f0c0c0' can also be used.

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. 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'
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'

appearance property	Corresponding SVG attribute	Notes	Initial default
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 ˙ (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'

Some useful JavaScript functions

There are many useful functions built in to JavaScript. Here are some examples that provide basic input and output facilities.

```
document.title = 'Test';                                  // Set the title of the browser tab
document.write('message');                                // Output some text to the screen (not as part of your drawing)
document.writeln('message');                              // Output some text to the screen and then start a new line
alert('message');                                         // Put a message in pop-up box
answer = prompt('Question?', 'Default answer');           // Input a string in response to a question in a pop-up box
n = Number(answer);                                       // Convert a string to a number or to Not A Number (NaN)
isNaN(n);                                                 // Boolean to test for NaN; used for example in an if statement
```

Generally, you do not need to use the *Number* function, as JavaScript automatically coerces strings to numbers when needed, and vice versa. Use *Number()* if you want to cast a string variable to a number.

Appendix: Sample JavaScript for a basic horizontal dial

Contents of file test.htm (this is the file you click to run the program)

```
<!DOCTYPE html>
<html lang='en'><head><meta charset='utf-8'>
<script src='https://www.gnomoni.ca/gMath.js'></script>
<script src='https://www.gnomoni.ca/gPlot.js'></script>
<title></title>
</head><body></body>
<script src = 'test.js'></script>
</html>
```

Contents of file test.js

```
document.title = 'Horizontal Dial';
let latitude = prompt('Latitude (as decimal degrees)?', 48.6);
let drawing = new Plot();
drawing.setSizeAndOrigin('1000px', '700px');
let radius = 300;
let halfday = acos(-tan(abs(latitude)) * tan(23.43));
halfday = ceil(halfday / 15) * 15;
for (hour = -halfday; hour <= halfday; hour += 15 / 4) {
    let angle = atan2(sin(latitude) * sin(hour), cos(hour));
    let appearance = {strokeWidth: 2, stroke: (hour % 1 == 0) ? 'black' : 'silver'};
    drawing.line({x: radius * sin(angle) * 0.9, y: radius * cos(angle) * 0.9}, {x: radius * sin(angle), y: radius * cos(angle)}, appearance);
    if (hour % 1 == 0) drawing.textOnCircle((180 + hour) / 15,  radius * 1.03, angle, auto, auto, {fontSize: 20});
}
drawing.text(`Latitude: ${latitude}°`, {x: 0, y: -radius + 50}, 'middle', auto, {fontSize: 20});
drawing.circle(1, auto, {fill: 'black'});
```

JavaScript statement	Explanation
document.title = 'Horizontal Dial';	Set the title of the browser tab.
let latitude = prompt('Latitude (as decimal degrees)?', 48.6);	Receive the latitude in response to a question. The default answer is 48.6°.
let drawing = new Plot();	Create the plot area.
drawing.setSizeAndOrigin('1000px', '700px');	Set the drawing dimensions. The origin is not specified in this example so by default it is at the center of the drawing area.

JavaScript statement	Explanation
let radius = 300;	Choose an outer radius for the radial hour lines.
let halfday = acos(-tan(abs(latitude)) * tan(23.43));	Calculate the maximum half-daylength as an angle (summer solstice), using standard formula $\cos \omega_0 = -\tan \varphi \tan \delta$
halfday = ceil(halfday / 15) * 15;	Round it up to a whole number of hours.
for (hour = -halfday; hour <= halfday; hour += 15 / 4) {	At 15 minute intervals, do the following ...
let angle = atan2(sin(latitude) * sin(hour), cos(hour));	Calculate the hour line angle for a horizontal dial. The standard formula is $\tan \theta = \sin \varphi \tan \omega$ but I expanded *tan* into *sin* ÷ *cos* so that I could use of *atan2*. By doing it this way, the angle is in the correct quadrant irrespective of hemisphere and time of day.
let appearance = {strokeWidth: 2, stroke: (hour % 1 == 0) ? 'black' : 'silver'};	Set up an appearance definition to have black lines for the hours and silver line for quarter hours. I used JavaScript's conditional assignment to select the required value.
drawing.line({x: radius * sin(angle) * 0.9, y: radius * cos(angle) * 0.9}, {x: radius * sin(angle), y: radius * cos(angle)}, appearance);	Draw radial hour lines between the two coordinates provided. The inside end is at 90% of outer radius. Use the appearance object just defined.
if (hour % 1 == 0) drawing.textOnCircle((180 + hour) / 15, radius * 1.03, angle, auto, auto, {fontSize: 20});	If it's a whole hour, draw a label using text on an invisible circle using an hour number in the range 0 – 23. The circle is slightly bigger than the hourline radius. The text appears at the angle just calculated (angles are measured from the top). Use the default value to have text on the outside of the circle, and use the default value for the center of the circle i.e. at the origin. Write the text at size 20.
}	Repeat for the next hour line.
drawing.text(`Latitude: ${latitude}°`, {x: 0, y: -radius + 50}, 'middle', auto, {fontSize: 20});	Mark the latitude on the drawing using position given, with the text centered at that position, using the default of no rotation and writing the text at size 20.
drawing.circle(1, auto, {fill: 'black'});	Add a tiny circle to indicate the position of the toe of the gnomon. Use the default, which is the origin, for the center of the circle and fill the circle with black ink.