Proceedings of the 5th International Conference on Nonlinear Dynamics
ND-KhPI2016
September 27-30, 2016, Kharkov, Ukraine

Algorithms and JavaScript Programs in Calculation of R-Functions and Producing Their Two- and Three-Dimensional Charts

Roman A. Uvarov[1*]

Abstract
Usage of HTML5 canvas element suitable for two- and three-dimensional charts of the function is unveiled. Appliance of JavaScript dynamics to this element is implemented. R-functions and level curves construction details are specified. Two- and three-dimension function charts are represented.

Keywords
HTML5, canvas, JavaScript, level curves, theory of R-functions

[1] Scientific Fellow at Podgorny Institute for Mechanical Engineering Problems of NAS of Ukraine, 2/10 Dm. Pozharsky str., Kharkiv, Ukraine
*** Corresponding author:** rqa0001@gmail.com

Introduction

Automation of the process for charting the various functions in order to visualize their study under the modern state of information technology development can be solved in many different ways, either complex, requiring a lot of software and hardware resources, or relatively simple, requiring basic knowledge of programming. Object-oriented programming has firmly occupied a niche in the approaches to the implementation of computational algorithms. One of the modern object-oriented programming languages, extendable for a particular purpose and not demanding to resources, is a high-level JavaScript language.

Modules used to translate the JavaScript language are embedded into modern web browsers. These modules are also updated with the upgrade of web browsers, adding the support of features of constantly evolving language. Web browsers translate the HTML language that describes the visual representation of static web pages, such as a graphical user interface, and the JavaScript language allows us to describe and define the dynamics of the interaction of web page elements with each other as well as the user with a web page on the client level.

Thus, the researcher requires a web browser and basic knowledge of JavaScript and HTML with CSS for the full implementation of computational algorithms. This gives an advantage to the researcher in a sense of obtaining quick results, because the research does not require special heavy IDEs and compilers, running or creating executable programs under a certain platform, which is controlled by a certain operating system. Also, since the program code is not compiled but remains in the open form, it can be easily modified by the researcher, for example, in the case of changes in the computational algorithm. So, simple in terms of implementation and usage and multi-functional in terms of the given task processing, the approach to the implementation of computational algorithms in JavaScript language is a handy for using, studying, and solving the practical problems.

1. Application of JavaScript and HTML5 Canvas Element to Rendering of the Level Curves of Function

To render a chart of two-dimensional or three-dimensional functions, we will use the canvas element, introduced in HTML5 and supported by modern web browsers such as Internet Explorer,

Opera, Firefox, Chrome, Safari. Depending on the context of this element usage, the represented chart of the function will be either one-dimensional or two-dimensional, when the input getContext method parameter value is "2d", or three-dimensional, when the value is "webgl" or "experimental-webgl".

The ***aim of the work*** is to show quick and easy appliance of JavaScript to producing the charts of two- and three-dimensional functions also constructed with R-operations.

We will begin with producing a two-dimensional chart of the function. First, let's create a canvas element in HTML5 (Fig. 1), which will be processed by draw() method on loading a web page.

```
<html>
<head><title>Canvas code</title>
<script>JavaScript code will be here...</script>
</head>
<body onload="draw()">
<canvas id="canvas" width="800" height="600"></canvas>
</body>
</html>
```

Figure 1. Creating a canvas element on a web page

After that in the script section, containing a JavaScript program, we will create a draw() method, defining the context of canvas object. Since we're planning to build a two-dimensional chart of the function, the value of context will be equal to "2d". Also under the same method we'll determine a scale of function display as well as signs to show or hide negative coordinate semi-axes of Cartesian coordinate system. We'll complete the draw() method by calling showAxes() method to construct coordinate axes and by calling funGraph() method to build any two-dimensional function, which on Figure 2 will be funOmega function, but its construction will be discussed later.

```
function draw() {
 var canvas = document.getElementById("canvas");
 if (null==canvas || !canvas.getContext) return;

 var axes={}, ctx=canvas.getContext("2d");
 axes.x0 = .5*canvas.width;     // x0 pixels from left to x=0
 axes.y0 = .5*canvas.height;    // y0 pixels from top to y=0
 axes.scale = 3;                // 3 pixels from x=0 to x=1
 axes.doNegativeX = true;
 axes.doNegativeY = true;

 showAxes(ctx,axes);
 funGraph(ctx,axes,funOmega,"noColor",2);
}
```

Figure 2. A method of displaying the coordinate axes and an arbitrary function

Represented in Fig. 3 method showAxes (), constructing coordinate axes, takes into account the signs to show or hide the negative coordinate semi-axes, which are set in the draw() method, and may remain the same all along possible further modifications to the program.

```
function showAxes(ctx,axes) {
 var x0=axes.x0, w=ctx.canvas.width;
 var y0=axes.y0, h=ctx.canvas.height;
 var xmin = axes.doNegativeX ? 0 : x0;
 var ymin = axes.doNegativeY ? 0 : y0;
 var ymax = axes.doNegativeY ? h : 0;
 ctx.beginPath();
 ctx.strokeStyle = "rgb(128,128,128)";
 ctx.moveTo(xmin,y0); ctx.lineTo(w,y0);
 ctx.moveTo(x0,ymin); ctx.lineTo(x0,ymax);
 ctx.stroke();
}
```

Figure 3. Implementation of the method for displaying the coordinate axes

Before proceeding to implement the method of constructing an arbitrary two-dimensional function, we declare an array of colors (Fig. 4), which will be used to represent the eight level curves of the function.

```
var colors = ["rgba(  0,  0,200,0.5)",  // blue
              "rgba(  0,200,  0,0.5)",  // green
              "rgba(200,  0,  0,0.5)",  // red
              "rgba(  0,200,200,0.5)",  // cyan
              "rgba(200,  0,200,0.5)",  // magenta
              "rgba(200,200,  0,0.5)",  // yellow
              "rgba(150,150,150,0.5)",  // light grey
              "rgba( 10, 10, 10,0.5)"]; // dark grey
```

Figure 4. An array of colors of function level curves

Implementation of the method for constructing an arbitrary two-dimensional function will consist of several successive parts. At first, a subroutine (Fig. 5) to find the maximum and minimum values of the function in the field of values as well as to determine the equal width of level curves, which is calculated as a ratio of a maximum function value to a number of level curves (or a number of colors array for level curves).

```
function funGraph (ctx,axes,func,color,thick) {
 var xx, yy, dx=1, dy=1, x0=axes.x0, y0=axes.y0, scale=axes.scale;

 var iMax = Math.round((ctx.canvas.width-x0)/dx);
 var iMin = axes.doNegativeX ? Math.round(-x0/dx) : 0;
 var jMax = Math.round((ctx.canvas.height-y0)/dy);
 var jMin = axes.doNegativeY ? Math.round(-y0/dy) : 0;

 // determining the max value of w function
 var wMax = 0;
 for (var i=iMin;i<=iMax;i++) {
   for (var j=jMin;j<=jMax;j++) {
   xx = dx*i;
   yy = dy*j;
   var w = scale*func(xx/scale, yy/scale);
   if (w>wMax) {wMax=w}
  }
 }
 // defining the equal range of level lines
 var range = wMax / 8;
```

Figure 5. Subroutine to find maximum and minimum values of the function and to determine the equal width of level curves

Next, we'll implement subroutine (Fig. 6) to display the maximum values for each level curve of the function in the upper right corner of the canvas element.

```
//level line results
var llResX = 75;
var llResWidth = 15;
var llResXd = llResX - llResWidth;
ctx.font="15px Georgia";

for (var ci=0;ci<=7;ci++) {
   ctx.fillStyle=colors[ci];
   ctx.fillRect((ctx.canvas.width-llResX),0+(10+5)*ci,10,10);
   ctx.fillText(Math.round(range*(ci+1)*100)/100,
                (ctx.canvas.width-llResXd),10+(10+5)*ci);
}
```

Figure 6. Subroutine to display the maximum values for each level curve of the function

And finally, the third part of the implementation of the method will be devoted to a subroutine (Fig. 7) plotting the functions.

```
//drawing the level lines
ctx.beginPath();
for (var i=iMin;i<=iMax;i++) {
  for (var j=jMin;j<=jMax;j++) {
  xx = dx*i;   yy = dy*j;

  var w = scale*func(xx/scale, yy/scale);
  if (w<0) ctx.moveTo(x0+xx,y0-yy)
   else {
     for (ci=0;ci<=7;ci++) {
       if (w>range*ci && w<=range*(ci+1)) {
         ctx.fillStyle=colors[ci];
       }
     }
     ctx.fillRect(x0+xx,y0-yy,1,1)
   }
  }
}
ctx.stroke();
}
```

Figure 7. Subroutine plotting the functions

This construction will be carried out in the following way. From top to bottom and from left to right all the points of the display area are taken step-by-step, and the value of function transmitted to the method for displaying is computed in each point. If the function value falls into the range of values of the level curves determined in advance in the first part of the implementation of the method, a given point will conform to this level curve color from a prepared set.

We'll implement the R-conjunction and R-disjunction operations [1] as it is shown in Fig. 8. We'll use an operation with an index equal to 1 because it is the fastest from a computational point of view.

```
// OR
function r_diz(x, y) {
 var result;
 if (x>y) result=x;
  else    result=y;
 return result;
}
```

```
// AND
function r_con(x, y) {
 var result;
 if (x<y) result=x;
  else    result=y;
 return result;
}
```

Figure 8. R-conjunction and R-disjunction operations

We'll implement support functions and composite function funOmega as it is shown in Fig.9. And funOmega itself will be fetched to display in funGraph(), calling it from draw().

```
function fun1(x, y) {return 60*60-x*x-y*y;}
function fun2(x, y) {return (20*20-x*x);}
function fun3(x, y) {return (10*10-(y-40)*(y-40));}

function funOmega(x, y) {
  var w1 = fun2(x,y);
  var w2 = fun3(x,y);
  var w3 = r_con(w1, w2);
  var w4 = fun1(x,y);
  var w5 = r_con(-w3, w4);
  return w5;
}
```

Figure 9. Support functions and composite function funOmega

The result of the program, parts of which are given here, will be a two-dimensional chart of the function with a maximum values on the level curves shown in Fig. 10.

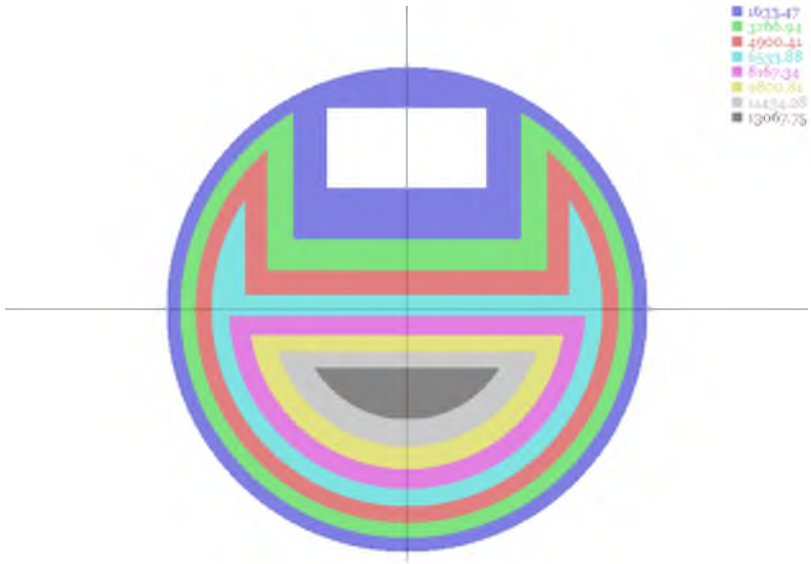

Figure 10. Two-dimensional chart of the function, constructed with R-operations

The results of producing the charts of three-dimensional functions are shown in Fig.11.

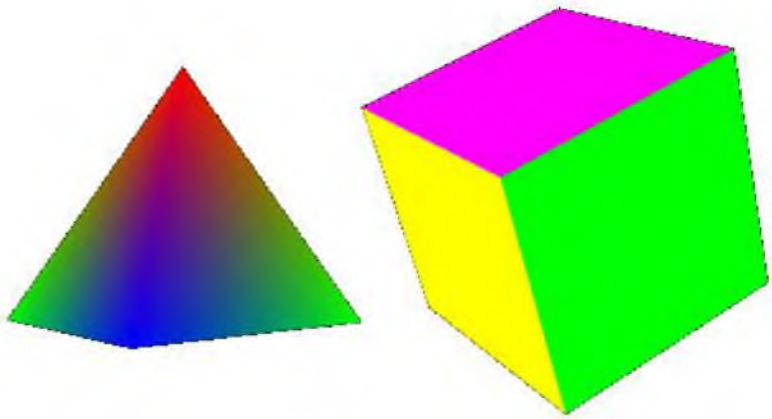

Figure 11. Charts of three-dimensional functions

Conclusions

As a conclusion it may be stated that using the JavaScript language and having a modern web browser at hand, a researcher can simply and quickly implement various computing algorithms, including those for rendering two-dimensional and three-dimensional charts of functions, which also built up with the help of R-operations.

References

[1] Rvachev V.L. *Theory of R-functions and its several applications.* Kiev: Nauk. dumka; 1982.