

JSXGraph Reference Card

Include JSXGraph in HTML

Three parts are needed: Include files containing the software, an HTML element, and JavaScript code.

Include files:

Two files have to be included: `jsxgraph.css`, and `jsxgraph-core.js`.

```
- <link rel="stylesheet" type="text/css"
    href="domain/jsxgraph.css"/>
- <script type="text/javascript"
    src="domain/jsxgraphcore.js"></script>
```

`domain` is the location of the files. This can be a local directory or `http://cdnjs.cloudflare.com/ajax/libs/jsxgraph/0.92/jsxgraphcore.js` `http://jsxgraph.uni-bayreuth.de/distrib/`

HTML element containing the construction:

```
<div id="box" class="jxgbox"
    style="width:600px; height:600px;"></div>
```

JavaScript code:

```
<script type="text/javascript">
    var b = JXG.JSXGraph.initBoard('box',{axis:true});
</script>
```

Initializing the board

```
var b = JXG.JSXGraph.initBoard('box',{attributes});
```

– *Attributes of the board*

```
boundingBox:      [x1,y1,x2,y2] user coordinates of the
                    upper left and bottom right corner
keepaspectratio:true/false      default: false
zoomX, zoomY:      zoom factor in x/y-axis direction
zoomfactor:        overall zoom factor in both directions
axis, grid, showNavigation, showCopyright, zoom, pan:
show axis, grid, zoom/navigation buttons, copyright text; enable mouse wheel zoom, shift+mouse panning
```

Properties and methods of the board:

```
b.snapToGrid:true/false:      grid mode
b.suspendUpdate()              stop updating (if speed is needed)
b.unsuspendUpdate()            restart updating
b.addChild(b2)                  Connect board b2 to board b
```

Basic commands

```
var e1 = b.create('type', [parents], {attributes});
e1.setProperty({key1:value1, key2:value2, ...});
```

Point

```
b.create('point', [parents], {atts});
```

Parent elements:

```
[x,y]                Euclidean coordinates
[z,x,y]              Homogeneous coordinates (z in first place)
[function(){return p1.X();},
 function(){return p2.Y();}]  Functions for x, y, (and z)
[function(){return [a,b];}]    Function returning array
[function(){return new JXG.Coords(...);}]
                                Function returning Coords object
```

Methods

```
p.X(), p.Y()          x-coordinate, y-coordinate
p.Z()                 (Homogeneous) z-coordinate
p.Distance(q)          Distance from p to point q
```

Glider

Point on circle, line, curve, or turtle.

```
b.create('glider', [parents], {atts});
```

Parent elements:

```
[x,y,c]              Initial coordinates and object to glide on
[c]                  Object to glide on (initially at origin)
```

Coordinates may also be defined by functions, see Point.

Line

```
b.create('line', [parents], {atts});
```

Parent elements:

```
[p1,p2]              line through 2 points
[c,a,b]              line defined by 3 coordinates (can also be functions)
[[x1,y1],[x2,y2]]    line by 2 coordinate pairs
```

In case of coordinates as parents, the line is the set of solutions of the equation $a \cdot x + b \cdot y + c \cdot z = 0$.

Circle

```
b.create('circle', [parents], {atts});
```

Parent elements:

```
[p1,p2]              2 points: center and point on circle line
[p,r]                center, radius (constant or function)
[p,c],[c,p]          center, circle from which the radius is taken
[p,l],[l,p]          center, line segment for the radius
[p1,p2,p3]           circle through 3 points
Points may also be specified as array of coordinates.
```

Polygon

```
b.create('polygon', [p1,p2,...], {atts});
[p1,p2,...]          The array of points
is connected by line segments and the inner area is filled.
b.create('regularpolygon', [p1,p2,n], {atts});
```

Slider

```
var s = b.create('slider', [[a,b],[c,d],[e,f,g]], {atts});
[a,b],[c,d]:          visual start and end position of the slider
[e,f,g]:              the slider returns values between e and g,
                        the initial position is at value f
snapWidth:num         minimum distance between 2 values
s.Value():             returns the position of the slider  $\in [e, g]$ 
```

Group

```
b.create('group', [p1,p2,...], {atts});
[p1,p2,...]           array of points
Invisible grouping of points. If one point is moved, the others are transformed accordingly.
```

Curve

```
- b.create('functiongraph', [parents], {atts});
                                Function graph,  $x \mapsto f(x)$ 
```

```
[function(x){return x*x;}, -1,1]    function term
                                optional: start, end
```

```
- b.create('curve', [parents], {atts});
· Parameter curve,  $t \mapsto (f(t), g(t))$ :
```

```
[function(t){return 5*t;}, function(t){return t*t;}, 0,2]
                                x function, y function, optional: start, end
```

· *Polar curve*: Defined by the equation $r = f(\phi)$.

```
[function(phi){return 5*phi;}, [1,2], 0, Math.PI]
                                Defining function, optional: center, start, end
```

· *Data plot*:

```
[[1,2,3],[4,-2,3]]              array of x- and y-coordinates, or
[[1,2,3], function(x){return x*x;}]
                                array of x-coordinates, function term
```

```
- b.create('spline', [p1,p2,...], {atts});
[p1,p2,...]                  Cubic spline: array of points
```

```
- b.create('riemannsum', [f,n,type], {atts});
Riemann sum of type 'left', 'right', 'middle', 'trapezoidal', 'upper', or 'lower'
```

```
- b.create('integral', [[a,b],f], {atts});
```

Display the area $\int_a^b f(x)dx$.

Tangent, normal

```
var e1 = b.create('tangent', [g], {atts});
var e1 = b.create('normal', [g], {atts});
g                            glider on circle, line, polygon, curve, or turtle
```

Conic sections

```
- ellipse, hyperbola:        defined by the two foci points and a point on the conic section or the length of the major axis.
b.create('ellipse', [p1,p2,p3], {atts});
b.create('ellipse', [p1,p2,a], {atts});
b.create('hyperbola', [p1,p2,p3], {atts});
b.create('hyperbola', [p1,p2,a], {atts});
- parabola:                  defined by the focus and the directrix (line).
b.create('parabola', [p1,line], {atts});
- conic section:             defined by 5 points or by the (symmetric) quadratic form
```

$$(x, y, z) \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{01} & a_{11} & a_{12} \\ a_{02} & a_{12} & a_{22} \end{pmatrix} (x, y, z)^{\top}$$

```
b.create('conic', [p1,...,p5], {atts});
b.create('conic', [a00,a11,a22,a01,a02,a12], {atts});
```

Turtle

`var t = b.create('turtle', [parents], {atts});`
`t.X(), t.Y(), t.dir` position, direction (degrees).

Parent elements:

`[x,y,angle]` Optional start values for x , y , and direction

Methods:

`t.back(len);` or `t.bk(len);`
`t.clean();` erase the turtle lines without resetting the turtle
`t.clearScreen();` or `t.cs();` call `t.home()` and `t.clean()`
`t.forward(len);` or `t.fd(len);`
`t.hideTurtle();` or `t.ht();`
`t.home();` Set the turtle to $[0,0]$ and direction to 90.
`t.left(angle);` or `t.lt(angle);`
`t.lookTo(t2.pos);` Turtle looks to the turtle `t2`
`t.lookTo([x,y]);` Turtle looks to a coordinate pair
`t.moveTo([x,y]);` Move the turtle with drawing
`t.penDown();` or `t.pd();`
`t.penUp();` or `t.pu();`
`t.popTurtle();` pop turtle status from stack
`t.pushTurtle();` push turtle status on stack
`t.right(angle);` or `t.rt(angle);`
`t.setPos(x,y);` Move the turtle without drawing
`t.setPenColor(col);` `col`: colorString, e.g. 'red' or '#ff0000'
`t.setPenSize(size);` `size`: number
`t.showTurtle();` or `t.st();`

Text

Display static or dynamic texts.

`el = b.create('text', [x,y,"Hello"]);`
`el = b.create('text', [x,y,f]);` where
`f = function(){ return p.X(); }`
Example for a dynamic text: f returns the x coordinate of the point p .

Image

Display bitmap image (also as data uri).

`el = b.create('image', [uri-string, [x,y], [w,h]]);`
`[x,y]`: position of lower left corner, `[w,h]`: width, height

Other geometric elements

– *angle*: filled area defined by 3 points
`el = b.create('angle', [M,B,C], {atts});`
– *arc*: circular arc defined by 3 points
`el = b.create('arc', [A,B,C], {atts});`
– *arrow*: line through 2 points with arrow head
`el = b.create('arrow', [A,B], {atts});`
– *arrowparallel*: arrow parallel to arrow a starting at point P
`el = b.create('arrowparallel', [a,P], {atts});` or `[P,a]`
– *bisector*: angular bisector defined by 3 points, returns line
`el = b.create('bisector', [A,B,C], {atts});`
angular bisector defined by 2 lines, returns 2 lines
`el = b.create('bisectorlines', [l1,l2], {atts});`
– *incircle*: incircle of triangle defined by 3 points
`el = b.create('incircle', [A,B,C], {atts});`
– *circumcircle*: circle through 3 points (deprecated)
`el = b.create('circumcircle', [A,B,C], {atts});`
– *circumcirclemidpoint*: center of circle through 3 points
`el = b.create('circumcirclemidpoint', [A,B,C]);`
– *circumcircle arc*: circular arc defined by 3 points
`el = b.create('circumcirclearc', [A,B,C], {atts});`
– *midpoint*: midpoint between 2 points or the 2 points defined by a line
– *circumcircle sector*: circular sector defined by 3 points
`el = b.create('circumcirclesector', [A,B,C], {atts});`
`el = b.create('midpoint', [A,B], {atts});` or `[line]`
– *mirrorpoint*: rotate point B around point A by 180°
`el = b.create('mirrorpoint', [A,B], {atts});`
– *parallel*: line parallel to line l through point P
`el = b.create('parallel', [l,P], {atts});` or `[P,l]`
– *parallelpont*: point D such that $ABCD$ from a parallelogram
`el = b.create('parallelpont', [A,B,C], {atts});`
– *perpendicular*: line perpendicular to line l through point P
`el = b.create('perpendicular', [l,P], {atts});` or `[P,l]`
– *perpendicularpoint*: orthogonal projection of P onto l
`el = b.create('orthogonalprojection', [l,P], {});` or `[P,l]`
– *reflection*: reflection of point P over the line l . Superseded by transformations
`el = b.create('reflection', [l,P], {atts});` or `[P,l]`
– *sector*: circle sector defined by 3 points ???
`el = b.create('sector', [A,B,C], {atts});`
– *semi circle*: defined by 2 points p_1 and p_2 .
`b.create('semicircle', [p1,p2], {atts});`
– *intersection*: of 2 objects (lines or circles).
Returns array of length 2 with first and second intersection point (also for line/line intersection).
`b.create('intersection', [o1,o2,n], {atts});`

Transform

Affine transformation of points, images and texts.

`t = b.create('transform', [data,base], {type:'type'});`
`base`: the transformation is applied to the coordinates of this object.

Possible types:

– `translate`: `data=[x,y]`
– `scale`: `data=[x,y]`
– `reflect`: `data=[line]` or `[x1,y1,x2,y2]`
– `rotate`: `data=[angle,point]` or `[angle,x,y]`
– `shear`: `data=[angle]`
– `generic`: `data=[v11,v12,v13,v21,...,v33]` 3×3 matrix

Methods:

`t.bindTo(p)` the coordinates of p are defined by t
`t.applyOnce(p)` apply the transformation once
`t.melt(s)` combine two transformations to one: $t := t \cdot s$
`p2 = b.create('point', [p1,t], {fixed:true});`
Point p_2 : apply t on point p_1

Attributes of geometric elements

Generic attributes:

`strokeWidth`: number
`strokeColor,fillColor,highlightFillColor,`
`highlightStrokeColor,labelColor`: color string
`strokeOpacity,fillOpacity,highlightFillOpacity,`
`highlightStrokeOpacity`: value between 0 and 1
`visible,trace,draft`: true, false
`dash`: dash style for lines: 0,1,...,6
`infoboxtext`: string

Attributes for point elements:

`face`: possible point faces: '[]', 'o', 'x', '+', '<', '>', 'A', 'v'
`size`: number
`fixed`: true, false

Attributes for line elements:

`straightFirst,straightLast,withTicks`:true, false

Attributes for line, arc and curve elements:

`firstArrow,lastArrow`: true, false

Attributes for polygon elements:

`withLines`: true, false

Attributes for text elements:

`display`: 'html', 'internal'
`fontSize`: numerical value

Attributes for angle elements:

`text`: string

Color string:

HTML color definition or HSV color scheme:

`JXG.hsv2rgb(h,s,v)` $0 \leq h \leq 360, 0 \leq s, v \leq 1$
returns RGB color string.

Mathematical functions

Functions of the intrinsic JavaScript object *Math*:

`Math.abs`, `Math.acos`, `Math.asin`, `Math.atan`, `Math.ceil`,
`Math.cos`, `Math.exp`, `Math.floor`, `Math.log`, `Math.max`,
`Math.min`, `Math.random`, `Math.sin`, `Math.sqrt`, `Math.tan`

`(number).toFixed(3)`: Rounding a number to fixed precision

Additional mathematical functions are methods of `JXG.Board`.

`b.angle(A,B,C)` angle ABC
`b.cosh(x)`, `board.sinh(x)`
`b.pow(a,b)` a^b
`b.D(f,x)` compute $\frac{d}{dx}f$ numerically
`b.I([a,b],f)` compute $\int_a^b f(x)dx$ numerically
`b.root(f,x)` root of the function f .
`b.root(f,x)` Uses Newton method with start value x
`b.factorial(n)` computes $n! = 1 \cdot 2 \cdot 3 \cdots n$
`b.binomial(n,k)` computes $\binom{n}{k}$
`b.distance(arr1,arr2)` Euclidean distance
`b.lagrangePolynomial([p1,p2,...])` returns a polynomial through the given points
`b.neville([p1,p2,...])` polynomial curve interpolation
`c = JXG.Math.Numerics.bezier([p1,p2,...])` Bezier curve
 $p_2, p_3, p_5, p_6, \dots$ are control points. `b.create('curve',c);`
`c = JXG.Math.Numerics.bspline([p1,p2,...],depth)` B-spline curve
`f = JXG.Math.Numerics.regressionPolynomial(n,xArr,yArr)` Regression pol. of deg. n : `b.create('functiongraph',f);`
`b.riemannsum(f,n,type,start,end)` Area of Riemann sum, see *Curves*

– Intersection of objects:

`b.intersection(el1,el2,i,j)` intersection of the elements
 el_1 and el_2 which can be lines, circles or curves

In case of circle and line intersection, $i \in \{0,1\}$ denotes the first or second intersection. In case of an intersection with a curve, i and j are floats which are the start values for the path positions in the Newton method for el_1 and el_2 , resp.

Todo list

'axis', 'ticks'.

Chart

To do ...

Links

Help pages are available at <http://jsxgraph.org>