

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://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 brd = JXG.JSXGraph.initBoard('box',{axis:true});
</script>
```

Initializing the board

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

– *Attributes of the board*

<code>boundingbox:</code>	<code>[x₁,y₁,x₂,y₂]</code>	user coordinates of the upper left and bottom right corner
<code>keepaspectratio:true/false</code>		default: false
<code>zoomX, zoomY:</code>		zoom factor in <i>x/y</i> -axis direction
<code>zoomfactor:</code>		overall zoom factor in both directions
<code>axis, grid, showNavigation, showCopyright:</code>		true/false
		show axis, grid, zoom/navigation buttons, display copyright

Properties and methods of the board:

<code>brd.snapToGrid:true/false:</code>	grid mode
<code>brd.suspendUpdate()</code>	stop updating (if speed is needed)
<code>brd.unsuspendUpdate()</code>	restart updating

Basic commands

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

Point

```
brd.create('point',[parents],{attributes});
```

Parent elements:

<code>[x,y]</code>	Euclidean coordinates
<code>[z,x,y]</code>	Homogeneous coordinates (<i>z</i> in first place)
<code>[function(){return p1.X();},</code> <code>function(){return p2.Y();}]</code>	Functions for <i>x, y</i> , (and <i>z</i>)

Methods

<code>p.X()</code> , <code>p.Y()</code>	<i>x</i> -coordinate, <i>y</i> -coordinate
<code>p.Z()</code>	(Homogeneous) <i>z</i> -coordinate
<code>p.Distance(q)</code>	Distance from <i>p</i> to point <i>q</i>

Glider

Point on circle, line, curve, or turtle.

```
brd.create('glider',[parents],{attributes});
```

Parent elements:

<code>[x,y,c]</code>	Initial coordinates and object to glide on
<code>[c]</code>	Object to glide on (initially at origin)

Coordinates may also be defined by functions, see [Point](#).

Line

```
brd.create('line',[parents],{attributes});
```

Parent elements:

<code>[p1,p2]</code>	line through 2 points
<code>[c,a,b]</code>	line defined by 3 coordinates (can also be functions)
<code>[[x1,y1],[x2,y2]]</code>	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

```
brd.create('circle',[parents],{attributes});
```

Parent elements:

<code>[p1,p2]</code>	2 points: center and point on circle line
<code>[p,r]</code>	center, radius (constant or function)
<code>[p,c],[c,p]</code>	center, circle from which the radius is taken
<code>[p,l],[l,p]</code>	center, line segment for the radius
<code>[p1,p2,p3]</code>	circle through 3 points

Points may also be specified as array of coordinates.

Polygon

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

Slider

```
var s = brd.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

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

Curve

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

<code>[function(x){return x*x;},-1,1]</code>	function term
	optional: start, end

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

<code>[function(t){return 5*t;},function(t){return t*t;},0,2]</code>	<i>x</i> function, <i>y</i> 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*:

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

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

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

Tangent, normal

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

Turtle

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

Parent elements:

<code>[x,y,angle]</code>	Optional start values for <i>x, y</i> , 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); 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();
```

Other geometric elements

– *angle*: filled area defined by 3 points
`el = brd.create('angle', [A,B,C], {attributes});`

– *arc*: circular arc defined by 3 points
`el = brd.create('arc', [A,B,C], {attributes});`

– *arrow*: line through 2 points with arrow head
`el = brd.create('arrow', [A,B], {attributes});`

– *arrowparallel*: arrow parallel to arrow *a* starting at point *P*
`el = brd.create('arrowparallel', [a,P], {atts});` or `[P,a]`

– *bisector*: angular bisector defined by 3 points, returns line
`el = brd.create('bisector', [A,B,C], {atts});`

angular bisector defined by 2 lines, returns 2 lines
`el = brd.create('bisectorlines', [l1,l2], {atts});`

– *circumcircle*: circle through 3 points (deprecated)
`el = brd.create('circumcircle', [A,B,C], {atts});`

– *circumcirclemidpoint*: center of circle through 3 points
`el = brd.create('circumcirclemidpoint', [A,B,C]);`

– *midpoint*: midpoint between 2 points or the 2 points defined by a line
`el = brd.create('midpoint', [A,B], {atts});` or `[line]`

– *mirrorpoint*: rotate point *B* around point *A* by 180°
`el = brd.create('mirrorpoint', [A,B], {atts});`

– *parallel*: line parallel to line *l* through point *P*
`el = brd.create('parallel', [l,P], {atts});` or `[P,l]`

– *parallelpoin*t: point *D* such that *ABCD* from a parallelogram
`el = brd.create('parallelpoin`t', [A,B,C], {atts});

– *perpendicular*: line perpendicular to line *l* through point *P*
`el = brd.create('perpendicular', [l,P], {atts});` or `[P,l]`

– *perpendicularpoint*: point defining a perpendicular line to line *l* through point *P*
`el = brd.create('perpendicularpoint', [l,P], {});` or `[P,l]`

– *reflection*: reflection of point *P* over the line *l*. Superseded by transformations
`el = brd.create('reflection', [l,P], {atts});` or `[P,l]`

– *sector*: circle sector defined by 3 points ???
`el = brd.create('sector', [A,B,C], {atts});`

– *semi circle*: defined by 2 points *p*₁ and *p*₂.
`brd.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).
`brd.create('intersection', [o1,o2,n], {atts});`

Text

Display static or dynamic texts.

```
el = brd.create('text', [x,y, "Hello"]);
el = brd.create('text', [x,y,f]);
```

Example for a dynamic text: return the *x* coordinate of the point *p*.

```
f = function(){ return p.X(); }
```

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

Attributes for point elements:

face: point style: '[]', 'o', 'x', or '+'
size: number
fixed: true, false

Attributes for line elements:

straightFirst, straightLast, withTicks: true, false

Attributes for line and arc elements:

firstArrow, lastArrow: true, false

Attributes for polygons elements:

withLines: true, false

Attributes for text elements:

display: 'html', 'internal'

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.

Transform

Affine transformation of objects.

```
t = brd.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 [*x*₁,*y*₁,*x*₂,*y*₂]
- rotate: **data**=[*angle*,*point*] or [*angle*,*x*,*y*]
- shear: **data**=[*angle*]
- generic: **data**=[*v*₁₁,*v*₁₂,*v*₁₃,*v*₂₁, ..., *v*₃₃] 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 = brd.create('point', [p1,t], {fixed:true});
Point *p*₂: apply *t* on point *p*₁

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**.

brd.angle(A,B,C) angle *ABC*
brd.cosh(x), **board.sinh(x)**
brd.pow(a,b) a^b
brd.D(f,x) compute $\frac{d}{dx}f$ numerically
brd.I([a,b],f) compute $\int_a^b f(x)dx$ numerically
brd.root(f,x) root of the function *f*.
Uses Newton method with start value *x*
brd.factorial(n) computes $n! = 1 \cdot 2 \cdot 3 \cdots n$
brd.binomial(n,k) computes $\binom{n}{k}$
brd.distance(arr1,arr2) Euclidean distance
brd.lagrangePolynomial([p1,p2,...]) returns a polynomial through the given points
brd.neville([p1,p2,...]) polynomial curve interpolation
c = JXG.Math.Numerics.bezier([p1,p2,...]) Bezier curve
*p*₂, *p*₃, *p*₅, *p*₆, ... are control points. **brd.create('curve',c);**
f = JXG.Math.Numerics.regressionPolynomial(n,xArr,yArr) Regression pol. of deg. *n*: **brd.create('functiongraph',f);**
brd.riemannsum(f,n,type,start,end) Volume of Riemann sum, see *Curves*

– Intersection of objects:
brd.intersection(e11,e12,i,j) intersection of the elements *e*₁₁ and *e*₂ which can be lines, circles or curves
In case of circle and line intersection, *i* ∈ {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 *e*₁ and *e*₂, resp.

Todo list

'axis', 'image', 'integral', 'ticks'.

Chart

To do ...

Links

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