all files / solar-calc/lib/ sun.js

95.63% Statements 197/206 **83.93%** Branches 47/56 **100%** Functions 32/32 **96.32%** Lines 183/190

```
1       "use strict";
2
3   4×  var _createClass = (function () { function defineProperties(target, props) { for (var key in props) { var prop = props[key]; prop.configura
4
5   30× var _classCallCheck = function (instance, Constructor) { I if (!(instance instanceof Constructor)) { throw new TypeError("Cannot call a cl
6
7   2×  var Sun = (function () {
8   1×    function Sun(date, latitude, longitude) {
9   30×     _classCallCheck(this, Sun);
10
11  30×     this.date = date;
12  30×     this.latitude = latitude;
13  30×     this.longitude = longitude;
14
15  30×     this.julianDate = getJD(date);
16        }
17
18  2×    _createClass(Sun, {
19          solarNoon: {
20            get: function () {
21  2×          return calcSolNoon(this.julianDate, this.longitude, this.date);
22            }
23          },
24          timeAtAngle: {
25            value: function timeAtAngle(angle, rising) {
26  24×         return calcSunriseSet(rising, angle, this.julianDate, this.date, this.latitude, this.longitude);
27            }
28          }
29        });
30
31  2×    return Sun;
32      })();
33
34  2×  var formatDate = function formatDate(date, minutes) {
35  14×   var seconds = (minutes - Math.floor(minutes)) * 60;
36  14×   return new Date(Date.UTC(date.getFullYear(), date.getMonth(), date.getDate(), 0, minutes, seconds));
37      };
38
39  1×  function calcTimeJulianCent(jd) {
40  1112× var T = (jd - 2451545) / 36525;
41  1112× return T;
42      }
43
44  1×  function isLeapYear(yr) {
45  10528× return yr % 4 === 0 && yr % 100 !== 0 || yr % 400 === 0;
46      }
47
48  1×  function calcDoyFromJD(jd) {
49  10012× var z = Math.floor(jd + 0.5);
50  10012× var f = jd + 0.5 - z;
51  10012× var A;
52  10012× if (z < 2299161) {
53  10000×   A = z;
54        } else {
55  12×     var alpha = Math.floor((z - 1867216.25) / 36524.25);
56  12×     A = z + 1 + alpha - Math.floor(alpha / 4);
57        }
58  10012× var B = A + 1524;
59  10012× var C = Math.floor((B - 122.1) / 365.25);
60  10012× var D = Math.floor(365.25 * C);
61  10012× var E = Math.floor((B - D) / 30.6001);
62  10012× var day = B - D - Math.floor(30.6001 * E) + f;
63  10012× var month = E < 14 ? E - 1 : E - 13;
64  10012× var year = month > 2 ? C - 4716 : C - 4715;
65
66  10012× var k = isLeapYear(year) ? 1 : 2;
67  10012× var doy = Math.floor(275 * month / 9) - k * Math.floor((month + 9) / 12) + day - 30;
68  10012× return doy;
69      }
70
71  1×  function radToDeg(angleRad) {
72  3328× return 180 * angleRad / Math.PI;
73      }
74
75  1×  function degToRad(angleDeg) {
76  16648× return Math.PI * angleDeg / 180;
77      }
78
79  1×  function calcGeomMeanLongSun(t) {
80  2235× var L0 = 280.46646 + t * (36000.76983 + t * 0.0003032);
81  2235× while (L0 > 360) {
82  42364×   L0 -= 360;
83        }
84  2235× while (L0 < 0) {
85          L0 += 360;
```

```
86          }
87  2235×   return L0; // in degrees
88      }
89
90    1×  function calcGeomMeanAnomalySun(t) {
91  2220×   var M = 357.52911 + t * (35999.05029 - 0.0001537 * t);
92  2220×   return M; // in degrees
93      }
94
95    1×  function calcEccentricityEarthOrbit(t) {
96  1112×   var e = 0.016708634 - t * (0.000042037 + 1.267e-7 * t);
97  1112×   return e; // unitless
98      }
99
100   1×  function calcSunEqOfCenter(t) {
101 1108×   var m = calcGeomMeanAnomalySun(t);
102 1108×   var mrad = degToRad(m);
103 1108×   var sinm = Math.sin(mrad);
104 1108×   var sin2m = Math.sin(mrad + mrad);
105 1108×   var sin3m = Math.sin(mrad + mrad + mrad);
106 1108×   var C = sinm * (1.914602 - t * (0.004817 + 0.000014 * t)) + sin2m * (0.019993 - 0.000101 * t) + sin3m * 0.000289;
107 1108×   return C; // in degrees
108     }
109
110   1×  function calcSunTrueLong(t) {
111 1108×   var l0 = calcGeomMeanLongSun(t);
112 1108×   var c = calcSunEqOfCenter(t);
113 1108×   var O = l0 + c;
114 1108×   return O; // in degrees
115     }
116
117   1×  function calcSunApparentLong(t) {
118 1108×   var o = calcSunTrueLong(t);
119 1108×   var omega = 125.04 - 1934.136 * t;
120 1108×   var lambda = o - 0.00569 - 0.00478 * Math.sin(degToRad(omega));
121 1108×   return lambda; // in degrees
122     }
123
124   1×  function calcMeanObliquityOfEcliptic(t) {
125 2220×   var seconds = 21.448 - t * (46.815 + t * (0.00059 - t * 0.001813));
126 2220×   var e0 = 23 + (26 + seconds / 60) / 60;
127 2220×   return e0; // in degrees
128     }
129
130   1×  function calcObliquityCorrection(t) {
131 2220×   var e0 = calcMeanObliquityOfEcliptic(t);
132 2220×   var omega = 125.04 - 1934.136 * t;
133 2220×   var e = e0 + 0.00256 * Math.cos(degToRad(omega));
134 2220×   return e; // in degrees
135     }
136
137   1×  function calcSunDeclination(t) {
138 1108×   var e = calcObliquityCorrection(t);
139 1108×   var lambda = calcSunApparentLong(t);
140
141 1108×   var sint = Math.sin(degToRad(e)) * Math.sin(degToRad(lambda));
142 1108×   var theta = radToDeg(Math.asin(sint));
143 1108×   return theta; // in degrees
144     }
145
146   1×  function calcEquationOfTime(t) {
147 1112×   var epsilon = calcObliquityCorrection(t);
148 1112×   var l0 = calcGeomMeanLongSun(t);
149 1112×   var e = calcEccentricityEarthOrbit(t);
150 1112×   var m = calcGeomMeanAnomalySun(t);
151
152 1112×   var y = Math.tan(degToRad(epsilon) / 2);
153 1112×   y *= y;
154
155 1112×   var sin2l0 = Math.sin(2 * degToRad(l0));
156 1112×   var sinm = Math.sin(degToRad(m));
157 1112×   var cos2l0 = Math.cos(2 * degToRad(l0));
158 1112×   var sin4l0 = Math.sin(4 * degToRad(l0));
159 1112×   var sin2m = Math.sin(2 * degToRad(m));
160
161 1112×   var Etime = y * sin2l0 - 2 * e * sinm + 4 * e * y * sinm * cos2l0 - 0.5 * y * y * sin4l0 - 1.25 * e * e * sin2m;
162 1112×   return radToDeg(Etime) * 4; // in minutes of time
163     }
164
165   1×  function calcHourAngle(angle, lat, solarDec) {
166 1108×   var latRad = degToRad(lat);
167 1108×   var sdRad = degToRad(solarDec);
168 1108×   var HAarg = Math.cos(degToRad(90 + angle)) / (Math.cos(latRad) * Math.cos(sdRad)) - Math.tan(latRad) * Math.tan(sdRad);
169 1108×   var HA = Math.acos(HAarg);
170 1108×   return HA; // in radians (for sunset, use -HA)
171     }
172
173   1×  function isNumber(inputVal) {
174 1084×   var oneDecimal = false;
175 1084×   var inputStr = "" + inputVal;
176 1084×   for (var i = 0; i < inputStr.length; i++) {
177 1479×     var oneChar = inputStr.charAt(i);
```

```
178 1479×     I if (i === 0 && (oneChar === "-" || oneChar === "+")) {
179             continue;
180           }
181 1479×     if (oneChar === "." && !oneDecimal) {
182 24×         oneDecimal = true;
183 24×         continue;
184           }
185 1455×     if (oneChar < "0" || oneChar > "9") {
186 1060×       return false;
187           }
188         }
189 24×   return true;
190       }
191
192 1×   function getJD(date) {
193 30×   var year = date.getFullYear();
194 30×   var month = date.getMonth() + 1;
195 30×   var day = date.getDate();
196
197 30×   var A = Math.floor(year / 100);
198 30×   var B = 2 - A + Math.floor(A / 4);
199 30×   var JD = Math.floor(365.25 * (year + 4716)) + Math.floor(30.6001 * (month + 1)) + day + B - 1524.5;
200 30×   return JD;
201       }
202
203 1×   function calcSolNoon(jd, longitude, date) {
204 2×   var tnoon = calcTimeJulianCent(jd - longitude / 360);
205 2×   var eqTime = calcEquationOfTime(tnoon);
206 2×   var solNoonOffset = 720 - longitude * 4 - eqTime; // in minutes
207 2×   var newt = calcTimeJulianCent(jd + solNoonOffset / 1440);
208 2×   eqTime = calcEquationOfTime(newt);
209 2×   var solNoonLocal = 720 - longitude * 4 - eqTime; // in minutes
210 2×   while (solNoonLocal < 0) {
211           solNoonLocal += 1440;
212         }
213 2×     while (solNoonLocal >= 1440) {
214           solNoonLocal -= 1440;
215         }
216 2×     return formatDate(date, solNoonLocal);
217         // return timeString(solNoonLocal, 3);
218       }
219
220 1×   function dayString(jd) {
221 28×     if (jd < 900000 || jd > 2817000) {
222 15×       return "error";
223         } else {
224 13×       var z = Math.floor(jd + 0.5);
225 13×       var f = jd + 0.5 - z;
226 13×       var A;
227 13×       I if (z < 2299161) {
228             A = z;
229           } else {
230 13×         var alpha = Math.floor((z - 1867216.25) / 36524.25);
231 13×         A = z + 1 + alpha - Math.floor(alpha / 4);
232         }
233 13×       var B = A + 1524;
234 13×       var C = Math.floor((B - 122.1) / 365.25);
235 13×       var D = Math.floor(365.25 * C);
236 13×       var E = Math.floor((B - D) / 30.6001);
237 13×       var day = B - D - Math.floor(30.6001 * E) + f;
238 13×       var month = E < 14 ? E - 1 : E - 13;
239 13×       var year = month > 2 ? C - 4716 : C - 4715;
240 13×       return new Date(Date.UTC(year, month - 1, day, 0, 0, 0));
241       }
242     }
243
244 1×   function calcSunriseSetUTC(rise, angle, JD, latitude, longitude) {
245 1108×  var t = calcTimeJulianCent(JD);
246 1108×  var eqTime = calcEquationOfTime(t);
247 1108×  var solarDec = calcSunDeclination(t);
248 1108×  var hourAngle = calcHourAngle(angle, latitude, solarDec);
249       //alert("HA = " + radToDeg(hourAngle));
250 1108×  if (!rise) hourAngle = -hourAngle;
251 1108×  var delta = longitude + radToDeg(hourAngle);
252 1108×  var timeUTC = 720 - 4 * delta - eqTime; // in minutes
253 1108×  return timeUTC;
254     }
255
256 1×   function calcSunriseSet(rise, angle, JD, date, latitude, longitude)
257     // rise = 1 for sunrise, 0 for sunset
258     {
259 24×   var timeUTC = calcSunriseSetUTC(rise, angle, JD, latitude, longitude);
260 24×   var newTimeUTC = calcSunriseSetUTC(rise, angle, JD + timeUTC / 1440, latitude, longitude);
261 24×   if (isNumber(newTimeUTC)) {
262
263 12×     return formatDate(date, newTimeUTC);
264       } else {
265         // no sunrise/set found
266 12×     var doy = calcDoyFromJD(JD);
267 12×     var jdy;
268 12×     E if (latitude > 66.4 && doy > 79 && doy < 267 || latitude < -66.4 && (doy < 83 || doy > 263)) {
269           //previous sunrise/next sunset
```

```
12×       jdy = calcJDofNextPrevRiseSet(!rise, rise, angle, JD, latitude, longitude);
12×       return dayString(jdy);
        } else {
          //previous sunset/next sunrise
          jdy = calcJDofNextPrevRiseSet(rise, rise, angle, JD, latitude, longitude);
          return dayString(jdy);
        }
      }
    }

1× function calcJDofNextPrevRiseSet(next, rise, type, JD, latitude, longitude) {
12×   var julianday = JD;
12×   var increment = next ? 1 : -1;

12×   var time = calcSunriseSetUTC(rise, type, julianday, latitude, longitude);
12×   while (!isNumber(time)) {
1048×   julianday += increment;
1048×   time = calcSunriseSetUTC(rise, type, julianday, latitude, longitude);
      }

12×   return julianday;
    }

2× module.exports = Sun;
```