**marelac**: Tools for Aquatic Sciences

Karline Soetaert  
NIOZ-Yerseke  
The Netherlands

Thomas Petzoldt  
Technische Universität Dresden  
Germany

Filip Meysman  
NIOZ-Yerseke  
The Netherlands

---

**Abstract**

R package **marelac** (Soetaert, Petzoldt, and Meysman 2010) contains chemical and physical constants and functions, datasets, routines for unit conversion, and other utilities useful for MArine, Riverine, Estuarine, LAcustrine and Coastal sciences.

*Keywords*: marine, riverine, estuarine, lacustrine, coastal science, utilities, constants, R.

---

**1. Introduction**

R package **marelac** has been designed as a tool for use by scientists working in the MArine, Riverine, Estuarine, LAcustrine and Coastal sciences.

It contains:

- chemical and physical constants, datasets, e.g. atomic weights, gas constants, the earth's bathymetry.

- conversion factors, e.g. gram to mol to liter conversions; conversions between different barometric units, temperature units, salinity units.

- physical functions, e.g. to estimate concentrations of conservative substances as a function of salinity, gas transfer coefficients, diffusion coefficients, estimating the Coriolis force, gravity ...

- the thermophysical properties of the seawater, as from the UNESCO polynomial (Fofonoff and Millard 1983) or as from the more recent derivation based on a Gibbs function (Feistel 2008; McDougall, Feistel, Millero, Jackett, Wright, King, Marion, Chen, and Spitzer 2009a).

Package **marelac** does *not* contain chemical functions dealing with the aquatic carbonate system (acidification, pH). These function can be found in two other R packages, i.e. **seacarb** (Lavigne and Gattuso 2010) and **AquaEnv** (Hofmann, Soetaert, Middelburg, and Meysman 2010).
2. Constants and datasets

2.1. Physical constants

Dataset `Constants` contains commonly used physical and chemical constants, as in Mohr and Taylor (2005):

```r
> data.frame(cbind(acronym = names(Constants),
+                  matrix(ncol = 3, byrow = TRUE, data = unlist(Constants),
+                  dimnames=list(NULL, c("value", "units", "description"))))
```

<table>
<thead>
<tr>
<th>acronym</th>
<th>value</th>
<th>units</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>9.8</td>
<td>m/s²</td>
<td>gravity acceleration</td>
</tr>
<tr>
<td>SB</td>
<td>5.6697e-08</td>
<td>W/m² K⁻⁴</td>
<td>Stefan-Boltzmann constant</td>
</tr>
<tr>
<td>gasCt1</td>
<td>0.08205784</td>
<td>L atm K⁻¹ mol⁻¹</td>
<td>ideal gas constant</td>
</tr>
<tr>
<td>gasCt2</td>
<td>8.31447215</td>
<td>m³ Pa K⁻¹ mol⁻¹</td>
<td>ideal gas constant</td>
</tr>
<tr>
<td>gasCt3</td>
<td>83.1451</td>
<td>cm³ bar K⁻¹ mol⁻¹</td>
<td>ideal gas constant</td>
</tr>
<tr>
<td>E</td>
<td>1.60217653e-19</td>
<td>C</td>
<td>Elementary charge</td>
</tr>
<tr>
<td>F</td>
<td>96485.3</td>
<td>C/mol</td>
<td>Faraday constant</td>
</tr>
<tr>
<td>P0</td>
<td>101325</td>
<td>Pa</td>
<td>one standard atmosphere</td>
</tr>
<tr>
<td>B1</td>
<td>1.3806505e-23</td>
<td>J/K</td>
<td>Boltzmann constant</td>
</tr>
<tr>
<td>B2</td>
<td>8.617343e-05</td>
<td>eV/K</td>
<td>Boltzmann constant</td>
</tr>
<tr>
<td>Na</td>
<td>6.0221415e+23</td>
<td>mol⁻¹</td>
<td>Avogadro constant</td>
</tr>
<tr>
<td>C</td>
<td>299792458</td>
<td>m s⁻¹</td>
<td>Vacuum light speed</td>
</tr>
</tbody>
</table>

2.2. Ocean characteristics

Dataset `Oceans` contains surfaces and volumes of the world oceans as in Sarmiento and Gruber (2006):

```r
> data.frame(cbind(acronym = names(Oceans),
+                  matrix(ncol = 3, byrow = TRUE, data = unlist(Oceans),
+                  dimnames=list(NULL, c("value", "units", "description"))))
```

<table>
<thead>
<tr>
<th>acronym</th>
<th>value</th>
<th>units</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>1.35e+25</td>
<td>kg</td>
<td>total mass of the oceans</td>
</tr>
<tr>
<td>Vol</td>
<td>1.34e+18</td>
<td>m³</td>
<td>total volume of the oceans</td>
</tr>
<tr>
<td>VolSurf</td>
<td>1.81e+16</td>
<td>m³</td>
<td>volume of the surface ocean (0-50m)</td>
</tr>
<tr>
<td>VolDeep</td>
<td>9.44e+17</td>
<td>m³</td>
<td>volume of the deep ocean (&gt;1200m)</td>
</tr>
<tr>
<td>Area</td>
<td>3.58e+14</td>
<td>m²</td>
<td>total area of the oceans</td>
</tr>
<tr>
<td>AreaIF</td>
<td>3.32e+14</td>
<td>m²</td>
<td>annual mean ice-free area of the oceans</td>
</tr>
<tr>
<td>AreaAtl</td>
<td>7.5e+13</td>
<td>m²</td>
<td>area of the Atlantic ocean, &gt;45dgS</td>
</tr>
<tr>
<td>AreaPac</td>
<td>1.51e+14</td>
<td>m²</td>
<td>area of the Pacific ocean, &gt;45dgS</td>
</tr>
<tr>
<td>AreaInd</td>
<td>5.7e+13</td>
<td>m²</td>
<td>area of the Indian ocean, &gt;45dgS</td>
</tr>
</tbody>
</table>
Figure 1: Image plot of ocean bathymetry - see text for R-code

<table>
<thead>
<tr>
<th></th>
<th>AreaArct</th>
<th>AreaEncl</th>
<th>Depth</th>
<th>RiverFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>9.6e+12</td>
<td>4.5e+12</td>
<td>3690</td>
<td>3.7e+13</td>
</tr>
</tbody>
</table>

area of the Arctic ocean
area of enclosed seas (e.g. Mediterranean)
mean depth of the oceans
Total river flow

2.3. World bathymetric data

Data set Bathymetry from the marelac package can be used to generate the bathymetry (and hypsometry) of the world oceans (and land) (Fig.1):

```r
require(marelac)
image(Bathymetry$x, Bathymetry$y, Bathymetry$z, col = femmecol(100),
     asp = TRUE, xlab = "", ylab = "")
contour(Bathymetry$x, Bathymetry$y, Bathymetry$z, add = TRUE)
```

2.4. Surface of 1 dg by 1 dg grid cells of the earth

Function earth_surf estimates the surface (m$^2$) of the bathymetric grid cells of 1dg by 1dg, based on their latitude.

As an example, we use it to estimate the surface of the earth; the true surface is 510072000 km$^2$:

```r
> SURF <- outer(X = Bathymetry$x,
+               Y = Bathymetry$y,
+               FUN <- function(X, Y) earth_surf(Y, X))
> sum(SURF)
```
Similarly, we can estimate the surface and volume of the oceans; it should be $3.58 \times 10^{14}$ and $1.34 \times 10^{18}$ respectively.

```r
> sum(SURF*(Bathymetry$z < 0))
[1] 3.618831e+14
> -sum(SURF*Bathymetry$z*(Bathymetry$z < 0))
[1] 1.336255e+18
```

Combined with the dataset Bathymetry, function `earth_surf` allows to estimate the total earth surface at certain water depths (Fig. 2):

```r
> SurfDepth <- vector()
> dseq <- seq(-7500, -250, by = 250)
> for (i in 2:length(dseq)) {
+   ii <- which (Bathymetry$z > dseq[i-1] & Bathymetry$z <= dseq[i])
+   SurfDepth[i-1]<-sum(SURF[ii])
+ }

> plot(dseq[-1], SurfDepth, xlab="depth, m", log = "y",
+      ylab = "m2", main = "Surface at ocean depths")
```

2.5. AtomicWeight

Dataset AtomicWeight holds the atomic weight of most chemical elements according to the IUPAC table (Wieser 2006). The data set contains `NA` for elements which have no stable
isotopes (except U, Th, Pa). The data set can be called in two versions. \textbf{AtomicWeight} shows the full table and \texttt{atomicweight} can be used for symbolic computations with the elements (see also \texttt{molweight}).

> AtomicWeight

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Symbol</th>
<th>Weight</th>
<th>Footnotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hydrogen</td>
<td>H</td>
<td>1.00794(7)</td>
<td>gmr</td>
</tr>
<tr>
<td>2</td>
<td>helium</td>
<td>He</td>
<td>4.002602(2)</td>
<td>gr</td>
</tr>
<tr>
<td>3</td>
<td>lithium</td>
<td>Li</td>
<td>6.941(2)</td>
<td>+gmr</td>
</tr>
<tr>
<td>4</td>
<td>beryllium</td>
<td>Be</td>
<td>9.012182(3)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>boron</td>
<td>B</td>
<td>10.811(7)</td>
<td>gmr</td>
</tr>
<tr>
<td>6</td>
<td>carbon</td>
<td>C</td>
<td>12.0107(8)</td>
<td>gr</td>
</tr>
<tr>
<td>7</td>
<td>nitrogen</td>
<td>N</td>
<td>14.0067(2)</td>
<td>gr</td>
</tr>
<tr>
<td>8</td>
<td>oxygen</td>
<td>O</td>
<td>15.9994(3)</td>
<td>gr</td>
</tr>
<tr>
<td>9</td>
<td>fluorine</td>
<td>F</td>
<td>18.9984032(5)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>neon</td>
<td>Ne</td>
<td>20.1797(6)</td>
<td>gm</td>
</tr>
<tr>
<td>11</td>
<td>sodium</td>
<td>Na</td>
<td>22.98976928(2)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>magnesium</td>
<td>Mg</td>
<td>24.3050(6)</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>aluminium</td>
<td>Al</td>
<td>26.9815386(8)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>silicon</td>
<td>Si</td>
<td>28.0855(3)</td>
<td>r</td>
</tr>
<tr>
<td>15</td>
<td>phosphorus</td>
<td>P</td>
<td>30.973762(2)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>sulfur</td>
<td>S</td>
<td>32.065(5)</td>
<td>gr</td>
</tr>
<tr>
<td>17</td>
<td>chlorine</td>
<td>Cl</td>
<td>35.453(2)</td>
<td>gmr</td>
</tr>
<tr>
<td>18</td>
<td>argon</td>
<td>Ar</td>
<td>39.948(1)</td>
<td>gr</td>
</tr>
<tr>
<td>19</td>
<td>potassium</td>
<td>K</td>
<td>39.0983(1)</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>calcium</td>
<td>Ca</td>
<td>40.078(4)</td>
<td>g</td>
</tr>
<tr>
<td>21</td>
<td>scandium</td>
<td>Sc</td>
<td>44.955912(6)</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>titanium</td>
<td>Ti</td>
<td>47.867(1)</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>vanadium</td>
<td>V</td>
<td>50.9415(1)</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>chromium</td>
<td>Cr</td>
<td>51.9961(6)</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>manganese</td>
<td>Mn</td>
<td>54.938045(5)</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>iron</td>
<td>Fe</td>
<td>55.845(2)</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>cobalt</td>
<td>Co</td>
<td>58.933195(5)</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>nickel</td>
<td>Ni</td>
<td>58.6934(2)</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>copper</td>
<td>Cu</td>
<td>63.546(3)</td>
<td>r</td>
</tr>
<tr>
<td>30</td>
<td>zinc</td>
<td>Zn</td>
<td>65.409(4)</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>gallium</td>
<td>Ga</td>
<td>69.723(1)</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>germanium</td>
<td>Ge</td>
<td>72.64(1)</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>arsenic</td>
<td>As</td>
<td>74.92160(2)</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>selenium</td>
<td>Se</td>
<td>78.96(3)</td>
<td>r</td>
</tr>
<tr>
<td>35</td>
<td>bromine</td>
<td>Br</td>
<td>79.904(1)</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>krypton</td>
<td>Kr</td>
<td>83.798(2)</td>
<td>gm</td>
</tr>
<tr>
<td>37</td>
<td>rubidium</td>
<td>Rb</td>
<td>85.4678(3)</td>
<td>g</td>
</tr>
<tr>
<td>38</td>
<td>strontium</td>
<td>Sr</td>
<td>87.62(1)</td>
<td>gr</td>
</tr>
<tr>
<td>39</td>
<td>yttrium</td>
<td>Y</td>
<td>88.90585(2)</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>zirconium</td>
<td>Zr</td>
<td>91.224(2)</td>
<td>g</td>
</tr>
<tr>
<td>Element</td>
<td>Symbol</td>
<td>Atomic Weight (g)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>-------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>niobium</td>
<td>Nb</td>
<td>92.90638(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>molybdenum</td>
<td>Mo</td>
<td>95.94(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>technetium</td>
<td>Tc</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ruthenium</td>
<td>Ru</td>
<td>101.07(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhodium</td>
<td>Rh</td>
<td>102.90550(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>palladium</td>
<td>Pd</td>
<td>106.42(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>silver</td>
<td>Ag</td>
<td>107.8682(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cadmium</td>
<td>Cd</td>
<td>112.411(8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>indium</td>
<td>In</td>
<td>114.818(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tin</td>
<td>Sn</td>
<td>118.710(7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>antimony</td>
<td>Sb</td>
<td>121.760(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tellurium</td>
<td>Te</td>
<td>127.60(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iodine</td>
<td>I</td>
<td>126.90447(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xenon</td>
<td>Xe</td>
<td>131.293(6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>caesium</td>
<td>Cs</td>
<td>132.9054519(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>barium</td>
<td>Ba</td>
<td>137.327(7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lanthanum</td>
<td>La</td>
<td>138.90547(7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cerium</td>
<td>Ce</td>
<td>140.116(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>praseodymium</td>
<td>Pr</td>
<td>140.90765(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>neodymium</td>
<td>Nd</td>
<td>144.242(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>promethium</td>
<td>Pm</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>samarium</td>
<td>Sm</td>
<td>150.36(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>europium</td>
<td>Eu</td>
<td>151.964(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gadolinium</td>
<td>Gd</td>
<td>157.25(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>terbium</td>
<td>Tb</td>
<td>158.92535(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dysprosium</td>
<td>Dy</td>
<td>162.500(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>holmium</td>
<td>Ho</td>
<td>164.93032(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>erbium</td>
<td>Er</td>
<td>167.259(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thulium</td>
<td>Tm</td>
<td>168.93421(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ytterbium</td>
<td>Yb</td>
<td>173.04(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lutetium</td>
<td>Lu</td>
<td>174.967(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hafnium</td>
<td>Hf</td>
<td>178.49(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tantalum</td>
<td>Ta</td>
<td>180.94788(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tungsten</td>
<td>W</td>
<td>183.84(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rhenium</td>
<td>Re</td>
<td>186.207(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osmium</td>
<td>Os</td>
<td>190.23(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iridium</td>
<td>Ir</td>
<td>192.217(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>platinum</td>
<td>Pt</td>
<td>195.084(9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gold</td>
<td>Au</td>
<td>196.966569(4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mercury</td>
<td>Hg</td>
<td>200.59(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thallium</td>
<td>Tl</td>
<td>204.3833(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lead</td>
<td>Pb</td>
<td>207.2(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bismuth</td>
<td>Bi</td>
<td>208.98040(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>polonium</td>
<td>Po</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>astatine</td>
<td>At</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>radon</td>
<td>Rn</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>francium</td>
<td>Fr</td>
<td>*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
> AtomicWeight[8, ]

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Symbol</th>
<th>Weight</th>
<th>Footnotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>oxygen</td>
<td>O</td>
<td>15.9994(3)</td>
<td>gr</td>
</tr>
</tbody>
</table>

> (W_H2O<- with (atomicweight, 2 * H + O))

[1] 18.01528

### 2.6. Atmospheric composition

The atmospheric composition, given in units of the moles of each gas to the total of moles of gas in dry air is in function `atmComp`:

> atmComp("O2")

```
O2
0.20946
```

> atmComp()
3. Physical functions

3.1. Coriolis

Function coriolis estimates the Coriolis factor, \( f \), units \( \text{sec}^{-1} \) according to the formula:

\[
  f = 2 \times \omega \times \sin(\text{lat})
\]

where \( \omega = 7.292 e^{-5}\text{radians sec}^{-1} \).

The following R-script plots the coriolis factor as a function of latitude (Fig. 3):

```r
> plot(-90:90, coriolis(-90:90), xlab = "latitude, dg North", +     ylab = "/s" , main = "Coriolis factor", type = "l", lwd = 2)
```

3.2. Molecular diffusion coefficients

In function diffcoeff the molecular and ionic diffusion coefficients (\( \text{m}^2\text{s}^{-1} \)), for several species at given salinity (\( S \)) temperature (\( t \)) and pressure (\( P \)) is estimated. The implementation is based on Chapter 4 in (Boudreau 1997).

```r
> diffcoeff(S = 15, t = 15)*24*3600*1e4 # cm2/day
```

\[
\begin{array}{lcccccccccc}
\text{H2O} & \text{O2} & \text{CO2} & \text{H2} & \text{CH4} & \text{DMS} & \text{He} & \text{Ne} \\
1 & 1.478897 & 1.570625 & 1.241991 & 3.429952 & 1.198804 & 0.8770747 & 5.186823 & 2.749411 \\
\text{Ar} & \text{Kr} & \text{Xe} & \text{Rn} & \text{N2} & \text{H2S} & \text{NH3} \\
1 & 1.554712 & 1.166917 & 0.9126865 & 0.8079991 & 1.190863 & 1.180685 & 1.467438 \\
\text{NO} & \text{N2O} & \text{CO} & \text{SO2} & \text{OH} & \text{F} & \text{Cl} & \text{Br} \\
1 & 1.500764 & 1.164872 & 1.195 & 1.03556 & 3.543847 & 0.9577852 & 1.354384 & 1.391657 \\
\text{I} & \text{HC03} & \text{C03} & \text{H2P04} & \text{HP04} & \text{PO4} & \text{HS} \\
1 & 1.364436 & 0.7693272 & 0.6126977 & 0.6168857 & 0.495435 & 0.3991121 & 1.214088 \\
\text{HS03} & \text{S03} & \text{HS04} & \text{SO4} & \text{I03} & \text{NO2} & \text{N03} \\
1 & 0.8836584 & 0.7379176 & 0.8874275 & 0.700226 & 0.7069267 & 1.278582 & 1.283189 \\
\text{H} & \text{Li} & \text{Na} & \text{K} & \text{Cs} & \text{Ag} & \text{NH4} \\
1 & 6.510175 & 0.6738419 & 0.8807268 & 0.8807268 & 1.385375 & 1.106039 & 1.314599 \\
\text{Ca} & \text{Mg} & \text{Fe} & \text{Mn} & \text{Ba} & \text{Be} & \text{Cd} \\
1 & 0.5264259 & 0.4682133 & 0.4657005 & 0.4610938 & 0.5611859 & 0.3911549 & 0.4682133
\end{array}
\]
Values of the diffusion coefficients for a temperature range of 0 to 30 and for the 13 first species is in (Fig. 4):

```r
> difftemp <- diffcoeff(t = 0:30)[,1:13]
```

```r
> matplot(0:30, difftemp, xlab = "temperature", ylab = "m2/sec", + main = "Molecular/ionic diffusion", type = "l")
> legend("topleft", ncol = 2, cex = 0.8, title = "mean", col = 1:13, lty = 1:13, + legend = cbind(names(difftemp), format(colMeans(difftemp),digits = 4)))
```

3.3. Shear viscosity of water

Function `viscosity` calculates the shear viscosity of water, in centipoise (gm\(^{-1}\)sec\(^{-1}\)). The formula is valid for 0 \(<\) t \(<\) 30 and 0 \(<\) S \(<\) 36 (Fig. 5).
4. Dissolved gas concentrations

`gas_02sat` estimates the saturated concentration of oxygen in mgL\(^{-1}\). Method APHA (Greenberg 1992) is the standard formula in Limnology, the method after Weiss (1970) the traditional formula used in marine sciences.

```r
> gas_02sat(t = 20)
[1] 7.374404
```

Conversion to mmol m\(^{-3}\) can be done as follows:
The effect of salinity on saturated concentration is in (Fig.6).

```r
> plot(t, gas_O2sat(t = t), type = "l", ylim = c(0, 15), lwd = 2,
    + main = "Oxygen saturation", ylab = "mg/l", xlab = "temperature")
> lines(t, gas_O2sat(S = 0, t = t, method = "Weiss"), col = "green",
    + lwd = 2, lty = "dashed")
> lines(t, gas_O2sat(S = 35, t = t, method = "Weiss"), col = "red", lwd = 2)
> legend("topright", c("S=35", "S=0"), col = c("red","green"),
    + lty = c(1, 2), lwd = 2)
```

4.2. Solubilities and saturated concentrations

More solubilities and saturated concentrations (in mmolm$^{-3}$) are in functions `gas_solubility` and `gas_satconc`.

```r
> gas_satconc(species = "O2")

 02
 210.9798
```
We plot the saturated concentrations for a selection of species as a function of temperature and salinity (Fig. 7):

```R
> #
> mf <- par(mfrow = c(2, 2))
> #
> gs <- gas_solubility(t = Temp)
> species <- c("CCl4", "CO2", "N2O", "Rn", "CCl2F2")
> matplot(Temp, gs[, species], type = "l", lty = 1, lwd = 2, xlab = "temperature",
>         ylab = "mmol/m3", main = "solubility (S=35)")
> legend("topright", col = 1:5, lwd = 2, legend = species)
> #
> species2 <- c("Kr", "CH4", "Ar", "O2", "N2", "Ne")
> matplot(Temp, gs[, species2], type = "l", lty = 1, lwd = 2, xlab = "temperature",
>         ylab = "mmol/m3", main = "solubility (S=35)")
> legend("topright", col = 1:6, lwd = 2, legend = species2)
> #
> species <- c("N2", "CO2", "O2", "CH4", "N2O")
> gsat <- gas_satconc(t = Temp, species = species)
```
Figure 7: Saturated concentrations and solubility as a function of temperature and salinity, and for different species

```r
> matplot(Temp, gsat, type = "l", xlab = "temperature", log = "y", lty = 1,
+         ylab = "mmol/m3", main = "Saturated conc (S=35)", lwd = 2)
> legend("right", col = 1:5, lwd = 2, legend = species)
> #
> gsat <- gas_satconc(S = Sal, species = species)
> matplot(Sal, gsat, type = "l", xlab = "salinity", log = "y", lty = 1,
+         ylab = "mmol/m3", main = "Saturated conc (T=20)", lwd = 2)
> legend("right", col = 1:5, lwd = 2, legend = species)
> #
> par("mfrow" = mf)
```

4.3. Partial pressure of water vapor

Function `vapor` estimates the partial pressure of water vapor, divided by the atmospheric pressure (Fig.8).
Figure 8: Partial pressure of water in saturated air as a function of temperature

> plot(0:30, vapor(t = 0:30), xlab = "Temperature, dgC", ylab = "pH2O/P",
+     type = "l")

4.4. Schmidt number and gas transfer velocity

The Schmidt number of a gas (gas_schmidt) is an essential quantity in the gas transfer velocity calculation (gas_transfer). The latter also depends on wind velocity, as measured 10 metres above sea level ($u_{10}$) (Fig. 9).

> gas_schmidt(species = "CO2", t = 20)

CO2
[1,] 665.988

> useq <- 0:15

> plot(useq, gas_transfer(u10 = useq, species = "O2"), type = "l",
+     lwd = 2, xlab = "u10,m/s", ylab = "m/s",
+     main = "O2 gas transfer velocity", ylim = c(0, 3e-4))
> lines(useq, gas_transfer(u10 = useq, species = "O2", method = "Nightingale"),
+     lwd = 2, lty = 2)
> lines(useq, gas_transfer(u10 = useq, species = "O2", method = "Wanninkhof1"),
+     lwd = 2, lty = 3)


5. Seawater properties

5.1. Concentration of conservative species in seawater

Borate, calcite, sulphate and fluoride concentrations can be estimated as a function of the seawater salinity:

```r
> sw_conserv(S = seq(0, 35, by = 5))
```

<table>
<thead>
<tr>
<th>Borate</th>
<th>Calcite</th>
<th>Sulphate</th>
<th>Fluoride</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>2 59.42857</td>
<td>1468.571</td>
<td>4033.633</td>
<td>9.760629</td>
</tr>
<tr>
<td>3 118.85714</td>
<td>2937.143</td>
<td>8067.267</td>
<td>19.521257</td>
</tr>
<tr>
<td>4 178.28571</td>
<td>4405.714</td>
<td>12100.900</td>
<td>29.281886</td>
</tr>
<tr>
<td>5 237.71429</td>
<td>5874.286</td>
<td>16134.534</td>
<td>39.042515</td>
</tr>
<tr>
<td>6 297.14286</td>
<td>7342.857</td>
<td>20168.167</td>
<td>48.803144</td>
</tr>
<tr>
<td>7 356.57143</td>
<td>8811.429</td>
<td>24201.801</td>
<td>58.563772</td>
</tr>
<tr>
<td>8 416.00000</td>
<td>10280.000</td>
<td>28235.434</td>
<td>68.324401</td>
</tr>
</tbody>
</table>

Figure 9: Oxygen gas transfer velocity for seawater, as a function of wind speed

```r
> lines(useq, gas_transfer(u10 = useq, species = "O2", method = "Wanninkhof2"), + lwd = 2, lty = 4)
```
5.2. Two salinity scales

Millero, Feistel, Wright, and McDougall (2008) and McDougall, Jackett, and Millero (2009b) provide a function to derive absolute salinity (expressed in g kg\(^{-1}\)) from measures of practical salinity. Absolute salinity is to be used as the concentration variable entering the thermodynamic functions of seawater (see next section).

The conversion between salinity scales is done with functions:

- `convert_AStoPS` from absolute to practical salinity and
- `convert_PStoAS` from practical to absolute salinity

For example:

```r
> convert_AStoPS(S = 35)
[1] 34.83573

> convert_PStoAS(S = 35)
[1] 35.16504
```

These functions have as extra arguments the gauge pressure (\(p\)), latitude (\(\text{lat}\)), longitude (\(\text{lon}\)), and -optional- the dissolved Si concentration (\(\text{DSi}\)) and the ocean (\(\text{Ocean}\)).

When one of these arguments are provided, they also correct for inconsistencies due to local composition anomalies.

When \(\text{DSi}\) is not given, the correction makes use of a conversion table that estimates the salinity variations as a function of present-day local seawater composition. The conversion in R uses the FORTRAN code developed by D. Jackett (http://www.marine.csiro.au/~jackett/TEOS-10/).

The correction factors are in a data set called `sw_sfac`, a list with the properties used in the conversion functions.

Below we first convert from practical to absolute salinity, for different longitudes, and then plot the correction factors as a function of latitude and longitude and at the seawater surface, i.e. for \(p=0\) (Fig.10).

```r
> convert_PStoAS(S = 35, lat = -10, lon = 0)
[1] 35.16525

> convert_PStoAS(S = 35, lat = 0, lon = 0)
[1] 35.16558
```

\(^1\)Before plotting, the negative numbers in the salinity anomaly table `sw_sfac` are converted to \(\text{NA}\) (not available). In the data set, numbers not available are denoted with \(-99\).
Figure 10: Salinity anomaly to convert from practical to absolute salinity and vice versa.

```r
> convert_PStoAS(S = 35, lat = 10, lon = 0)
[1] 35.16504

> convert_PStoAS(S = 35, lat = -10, lon = 0)
[1] 35.16525

> convert_PStoAS(S = 35, lat = -10, DSi = 1:10, Ocean = "Pacific")
[1] 35.16513 35.16523 35.16532 35.16541 35.16550 35.16560 35.16569
[8] 35.16578 35.16588 35.16597

> dsal <- t(sw_sfac$del_sa[1, , ])
> dsal [dsal < -90] <- NA

> image(sw_sfac$longs, sw_sfac$lats, dsal, col = femmecol(100),
+ asp = TRUE, xlab = "dg", ylab = "dg",
+ main = "salinity conversion - p = 0 bar")
> contour(sw_sfac$longs, sw_sfac$lats, dsal, asp = TRUE, add = TRUE)
```

Finally, the correction factors are plotted versus depth, at four latitudinal cross-sections (Fig.11):
Figure 11: Salinity anomaly to convert from practical to absolute salinity and vice versa for several latitudinal cross-sections (negative = S hemisphere)

```r
> ii <- c(6, 21, 24, 43)
> par(mfrow = c(2, 2))
> for (i in ii)
+ {
+   dsal <- t(sw_sfac$del_sa[ ,i, ])
+   dsal [dsal < -90] <- 0
+   image(sw_sfac$longs, sw_sfac$p, dsal, col = c("black", femmecol(100)),
+         xlab = "longitude, dg", ylab = "depth, m", zlim = c(0, 0.018),
+         main = sw_sfac$lat[i], ylim = c(6000, 0))
+   contour(sw_sfac$longs, sw_sfac$p, dsal, asp = TRUE, add = TRUE)
+ }
```

5.3. Thermophysical seawater properties

Package `marelac` also implements several thermodynamic properties of seawater. Either one can choose the formulation based on the UNESCO polynomial (Fofonoff and Millard 1983), which has served the oceanographic community for decades, or the more recent derivation as in Feistel (2008). In the latter case the estimates are based on three individual thermodynamic potentials for fluid water, for ice and for the saline contribution of seawater (the Helmholtz function for pure water, an equation of state for salt-free ice, in the form of a Gibbs potential function, and the saline part of the Gibbs potential).

Note that the formulations from Feistel (2008) use the absolute salinity scale (Millero et al.
2008), while the UNESCO polynomial uses practical salinity.

```r
> sw_cp(S = 40, t = 1:20)
[8] 3962.997 3963.768 3964.553 3965.348 3966.148 3966.949 3967.747

> sw_cp(S = 40, t = 1:20, method = "UNESCO")
[15] 3963.900 3964.906 3965.918 3966.931 3967.936 3968.927
```

The precision of the calculations can be assessed by comparing them to some test values:

```r
> t <- 25.5
> p <- 1023/10  # pressure in bar
> S <- 35.7
> sw_alpha(S, t, p) -0.0003098378393192645
[1] 1.167598e-13

> sw_beta(S, t, p) -0.0007257297978386655
[1] 2.555374e-12

> sw_cp(S, t, p) -3974.42541259729
[1] -5.945121e-07

> sw_tpot(S, t, p) -25.2720983155409
[1] 5.203708e-05

> sw_dens(S, t, p) -1027.95249315662
[1] 9.467044e-08

> sw_enthalpy(S, t, p) -110776.712408975
[1] -2.050104e-05

> sw_entropy(S, t, p) -352.81879771528
[1] -9.916204e-08
```
Below we plot all implemented thermophysical properties as a function of salinity and temperature (Fig. 12, 13). We first define a function that makes the plots:

```r
plotST <- function(fun, title)
+ {
+   Sal <- seq(0, 40, by = 0.5)
+   Temp <- seq(-5, 40, by = 0.5)
+   Val <- outer(X = Sal, Y = Temp, FUN = function(X, Y) fun(S = X, t = Y))
+   contour(Sal, Temp, Val, xlab = "Salinity", ylab = "temperature",
+           main = title, nlevel = 20)
+ }

> plotST(sw_gibbs, "Gibbs function")
> plotST(sw_cp, "Heat capacity")
> plotST(sw_entropy, "Entropy")
> plotST(sw_enthalpy, "Enthalpy")
> plotST(sw_dens, "Density")
> plotST(sw_svel, "Sound velocity")
```

The difference between the two formulations, based on the UNESCO polynomial or the Gibss function is also instructive (Fig. 14):
Figure 12: Seawater properties as a function of salinity and temperature - see text for R-code
Figure 13: Seawater properties as a function of salinity and temperature - continued - see text for R-code
Figure 14: Difference between two methods of calculating some seawater properties as a function of salinity and temperature - see text for R-code

```r
> par(mfrow = c(2, 2))
> par(mar = c(4, 4, 3, 2))
> plotST(function(S, t) sw_dens(S, t, method = "UNESCO") - sw_dens(S, t),
+ "Density UNESCO - Gibbs")
> plotST(function(S, t) sw_cp(S, t, method = "UNESCO") - sw_cp(S, t),
+ "Heat capacity UNESCO - Gibbs")
> plotST(function(S, t) sw_svel(S, t, method = "UNESCO") - sw_svel(S, t),
+ "Sound velocity UNESCO - Gibbs")
> par(mfrow = c(1, 1))
```
6. Conversions

Finally, several functions are included to convert between units of certain properties.

6.1. Gram, mol, liter conversions

`marelac` function `molweight` converts from gram to moles and vice versa. The function is based on a lexical parser and the IUPAC table of atomic weights, so it should be applicable to arbitrary chemical formulae:

```r
> 1/molweight("CO3")

CO3
0.01666419

> 1/molweight("HCO3")

HCO3
0.01638892

> 1/molweight(c("C2H5OH", "CO2", "H2O"))

C2H5OH CO2 H2O
0.02170683 0.02272237 0.05550844

> molweight(c("SiOH4", "NaHC03", "C6H12O6", "Ca(HC03)2", "Pb(NO3)2", "(NH4)2SO4"))

SiOH4 NaHC03 C6H12O6 Ca(HC03)2 Pb(NO3)2 (NH4)2SO4
48.11666 84.00661 180.15588 162.11168 331.20980 132.1395

We can use that to estimate the importance of molecular weight on certain physical properties (Fig.15):

```r
> #species <- colnames(gs) ## thpe: does not work any more, because 1D return value is vector
> species = c("He", "Ne", "N2", "O2",
+  "Ar", "Kr", "Rn", "CH4", "CO2", "N2O", "CCl2F2", "CCl3F", "SF6", "CCl4")
> gs <- gas_solubility(species = species)
> mw <- molweight(species)

> plot(mw, gs, type = "n", xlab = "molecular weight",
+  ylab = "solubility", log = "y")
> text(mw, gs, species)
```

Function `molvol` estimates the volume of one liter of a specific gas or the molar volume of an ideal gas.

```r
> molvol(species = "ideal")
```
Figure 15: Gas solubility as a function of molecular weight see text for R-code

```
ideal
24.46536

> molvol(species = "ideal", t = 1:10)

    ideal
[1,] 22.49599
[2,] 22.57804
[3,] 22.66010
[4,] 22.74216
[5,] 22.82421
[6,] 22.90627
[7,] 22.98833
[8,] 23.07039
[9,] 23.15244
[10,] 23.23450

> 1/molvol(species = "O2", t = 0)*1000

O2
44.67259

> 1/molvol(species = "O2", q = 1:6, t = 0)
```
6.2. Average elemental composition of biomass

The average elemental composition of marine plankton (Redfield ratio) is traditionally assumed to be $C_{106}H_{263}O_{110}N_{16}P_1$ (Redfield 1934; Redfield, Ketchum, and Richards 1963; Richards 1965), while Limnologists sometimes assume a ratio of $C_{106}H_{180}O_{45}N_{16}P_1$ (Stumm 1964). Since then, the ratio of C:N:P was widely agreed, but there is still discussion about the average of O and H. Anderson (1995) proposed a new formula $C_{106}H_{175}O_{12}N_{16}P_1$ for marine plankton and similarly Hedges, Baldock, Gélinas, Lee, Peterson, and Wakeham (2002), who used NMR analysis, found an elemental composition with much less hydrogen and oxygen ($C_{106}H_{175-180}O_{35-40}N_{15-20}S_{0.3-0.5}$) than in the original formula.

Function redfield can be used to simplify conversions between the main elements of biomass, where the default molar ratio can be displayed by:

```r
> redfield(1, "P")

  C   H   O   N   P
1 106 263 110 16 1
```

The second argument of the function allows to rescale this to any of the constitutional elements, e.g. to nitrogen:

```r
> redfield(1, "N")

  C   H   O   N   P
1 6.625 16.4375 6.875 1 0.0625
```

In addition, it is also possible to request the output in mass units, e.g. how many mass units of the elements are related to 2 mass units (e.g. mg) of phosphorus:
\begin{verbatim}
> redfield(2, "P", "mass")

   C     H     O     N     P
1 82.20727 17.11695 113.6403 14.47078 2

Finally, mass percentages can be obtained by:

> x <- redfield(1, "P", "mass")
> x / sum(x)

   C     H     O     N     P
1 0.3583026 0.0746047 0.4953044 0.06307127 0.008717054

or by using an alternative alternative elemental composition with:

> stumm <- c(C = 106, H = 180, O = 45, N = 16, P = 1)
> x <- redfield(1, "P", "mass", ratio = stumm)
> x / sum(x)

   C     H     O     N     P
1 0.5240061 0.07467398 0.2963318 0.09223971 0.01274841

Note however, that all these formulae are intended to approximate the average biomass composition and that large differences are natural for specific observations, depending on the involved species and their physiological state.

6.3. Pressure conversions

\texttt{convert\_p} converts between the different barometric scales:

\begin{verbatim}
> convert_p(1, "atm")

       Pa    bar    atm    torr
1 101325.3 1.013253 1.033214 1 760.0008
\end{verbatim}

6.4. Temperature conversions

Function \texttt{convert\_T} converts between different temperature scales (Kelvin, Celsius, Fahrenheit):

\begin{verbatim}
> convert_T(1, "C")

       K    C    F
1 274.15  1 33.8
\end{verbatim}
\end{verbatim}
6.5. Salinity and chlorinity

The relationship between Salinity, chlorinity and conductivity is in various functions:

> convert_StoCl(S = 35)
[1] 19.37394

> convert_RtoS(R = 1)
[1] 27.59808

> convert_StoR(S = 35)
[1] 1.236537

7. Finally

This vignette was made with Sweave (Leisch 2002).
References


**Affiliation:**

Karline Soetaert  
Royal Netherlands Institute of Sea Research (NIOZ)  
4401 NT Yerseke, Netherlands  
E-mail: karline.soetaert@nioz.nl  
URL: http://http://www.nioz.nl/

Thomas Petzoldt  
Institut für Hydrobiologie  
Technische Universität Dresden  
01062 Dresden, Germany  
E-mail: thomas.petzoldt@tu-dresden.de  
URL: http://tu-dresden.de/Members/thomas.petzoldt/

Filip Meysman  
Royal Netherlands Institute of Sea Research (NIOZ)  
4401 NT Yerseke, Netherlands  
E-mail: filip.meysman@nioz.nl