Package ‘CHNOSZ’

January 12, 2014

Date 2014-01-12

Version 1.0.3

Title Chemical Thermodynamics and Activity Diagrams

Author Jeffrey Dick

Maintainer Jeffrey Dick <j3ffdick@gmail.com>

Depends R (>= 2.12.0)

Suggests limSolve, parallel, testthat

Description This package includes functions and data sets to support chemical thermodynamic modeling in biochemistry and low-temperature geochemistry. The features include calculation of the standard molal thermodynamic properties and chemical affinities of reactions involving minerals and/or biomolecules; a database of thermodynamic properties of aqueous, crystalline and gaseous species; amino acid group additivity for the standard molal thermodynamic properties of neutral and ionized proteins; use of the revised Helgeson-Kirkham-Flowers equations of state for aqueous species; construction of equilibrium activity diagrams as a function of temperature, pressure, and chemical activities or fugacities of basis species.

License GPL (>= 2)

BuildResaveData no

ZipData no

URL http://www.chnosz.net/, http://chnosz.r-forge.r-project.org/

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-01-12 16:51:14
R topics documented:

CHNOSZ-package ......................................................... 3
affinity ................................................................. 6
anim.TCA ................................................................. 12
basis ....................................................................... 14
buffer ................................................................. 16
diagram ............................................................... 19
eos ................................................................. 26
EOSregress ............................................................... 30
eqdata ................................................................. 33
equilirate ............................................................ 35
examples ............................................................ 38
extdata ................................................................. 39
findit ................................................................. 45
IAPWS95 ................................................................. 47
info ................................................................. 49
ionize.aa .............................................................. 51
iprotein ............................................................... 54
makeup ............................................................... 57
more.aa ............................................................... 59
objective ............................................................ 60
protein ............................................................. 63
protein.info .......................................................... 67
read.expr ............................................................. 71
revisit ............................................................... 76
sideeffects ......................................................... 80
species ............................................................. 82
subcrt ............................................................... 84
swap.basis ......................................................... 91
taxonomy .......................................................... 92
thermo ............................................................. 94
transfer ............................................................ 101
util.affinity ........................................................ 103
util.args ........................................................... 105
util.array .......................................................... 106
util.blast .......................................................... 108
util.character .................................................... 111
util.data .......................................................... 112
util.expression .................................................. 116
util.fasta .......................................................... 119
util.formula ...................................................... 122
util.list ........................................................... 124
util.matrix ......................................................... 125
util.misc .......................................................... 126
util.plot .......................................................... 128
util.program ..................................................... 131
util.seq ........................................................... 132
Description

CHNOSZ is a package for thermodynamic calculations, primarily with applications in geochemistry and biochemistry. It can be used to calculate the standard molal thermodynamic properties and chemical affinities of reactions relevant to geobiochemical processes, and to visualize the equilibrium activities of species on chemical speciation and predominance diagrams. The package can be used interactively and in batch mode, through the use of R source files containing a sequence of commands. The major features of the package are outlined below, with links to specific help topics in this document, which constitutes the primary technical description of the package. If you are a new user, the ‘anintro’ vignette (An introduction to CHNOSZ) may offer a more comfortable way to get started with using the package.

Details

Major features in CHNOSZ:

- Thermodynamic database - assembles literature values of the standard thermodynamic properties and equations of state parameters of minerals, aqueous organic and inorganic species, gases and liquids (thermo).
- Group additivity for proteins - estimate the standard thermodynamic properties and equations of state parameters for unfolded proteins from their amino acid composition; includes an additive calculation of ionization state of proteins as a function of temperature and pH (protein).
- File and internet access - read protein sequences from FASTA files, and download sequence information from UniProt (read.fasta, protein).
- Equations of state - calculate the standard thermodynamic properties of proteins or other species in the database, and reactions between them, as a function of temperature and pressure (hkf, cg1, subcrt).
- Stoichiometry - count elements in chemical formulas of species, check and optionally correct mass balance of chemical reactions (makeup).
- System of interest - define the basis species for a system together with one or more species of interest; compute the stoichiometries of the formation reactions of the species of interest (basis, species).
- Chemical affinity - calculate the chemical affinities of the formation reactions of the species of interest at a single point, or as a function of one or more of chemical activities of the basis species, temperature and/or pressure (affinity).
• Chemical activity - calculate the equilibrium activities of the species of interest as a function of the same variables used in the affinity calculation, using a reference state transformation (either the Boltzmann distribution or a reaction matrix approach). (diagram, equil.reaction, equil.boltzmann).

• Buffer calculations - compute activities of basis species that are determined by a buffer of one or more species (e.g., pyrite-pyrrhotite-magnetite; acetic acid-CO2) (buffer).

• Activity diagrams - plot the equilibrium activities at a single point (as barplots), or as a function of one (species activity diagrams) or two (predominance diagrams) variables (diagram).

• Activity statistics - calculate summary statistics for equilibrium activities of species (revisit).

• Multidimensional optimization (new in 0.9-3) - using an iterative gridded optimization, find a combination of chemical activities of basis species, temperature and/or pressure that maximize or minimize the value of a target statistic (findit).

• Mass transfer calculations (experimental) - calculate changes in solution composition and formation of secondary species as a function of incremental reaction of a mineral (or protein) (transfer).

Here are some tips for new users:

• Install the package from CRAN using install.packages or its GUI menu equivalent.

• To begin working with the package after installation, type library(CHNOSZ) at the command line (or select the name of the package from the GUI menu).

• Running the examples shown in the various help topics is a great way to become more familiar with the usage of the functions. From help.start, select ‘Packages’ then ‘CHNOSZ’ and then select a function of interest. Individual examples can be run by pasting the example block directly into the R console.

• Type the command examples() to run all of the examples provided in CHNOSZ. This takes about five to ten minutes depending on your system. If things go as expected, the entire set will run without any warnings or errors.

• Some of the examples require internet or file access or user intervention, or are intentionally written to demonstrate conditions that lead to errors. This offensive code is hidden from R’s package checking mechanism using the dontrun tag. You can experiment with dontrun examples by pasting the code to the R console.

• A couple of other things to note about the examples: 1) There are some stopifnot statements that represent expected outcomes of the calculations; if the expectation is not met, the stopifnot statement causes an error. These tests are useful for checking the code during package development cycles, but are usually not of critical importance for the set-up of the problem (though they do sometimes employ useful programming tricks). 2) Commands written with an enclosing pair of parentheses (z <- "like this one") are used to display the result of an assignment operation (<-), the value of which is needed later in the calculation. In interactive use, the outermost pair of enclosing parentheses is generally not needed.

• To learn how to update the thermodynamic database, look at its documentation in thermo.

Compatibility

The recommended version of R is 2.14.0 or greater (to find vignettes in the vignettes directory, and for availability of parallel in the standard library). As of version 0.9-9, the package depends on
R version 2.12.0 or greater (so useDynLib in the NAMESPACE can find the shared library on Windows). Starting with version 0.9-6 of the package, the dependency was given as R version 2.10.0 or greater (to read compressed data files). Before version 0.9-6 of the package, the dependency was given as R version 2.7.0 or greater (major update of the X11 device in 2.7.0). Without accessing the compressed data files in extdata it should be possible to run CHNOSZ on Unix-alikes under R versions 2.4.0 or greater (availability of the ‘stringsAsFactors’ argument of data.frame).

Acknowledgements

This package would not exist without the fearless leadership and encouragement of Professor Harold C. Helgeson. Hal and his associates are in some way responsible for many of the equations and data contained in this package. A direct contribution of code is the file H2092D.f, taken from the SUPCRT92 distribution, with only cosmetic modifications (masking of WRITE and STOP statements) made for compatibility with an R environment. The revised Helgeson-Kirkham-Flowers equations of state are used in this package, together with the thermodynamic properties and parameters for many species taken from numerous papers coauthored by Helgeson.

Work on this package at U.C. Berkeley from ~2003–2008 was supported by research grants solicited by HCH from the U.S. National Science Foundation and Department of Energy. In 2009–2011, the major research project stimulating development of this package at Arizona State University was funded by the National Science Foundation under grant EAR-0847616. The files in extdata/bison are excerpts of results of BLAST calculations made on the Saguaro high performance computer at ASU.

Known Bugs

The values generated by buffer may not be applied correctly by affinity in calculating the affinities of the formation reactions. (The values returned by affinity(..., return.buffer=TRUE) do appear to be correct in the examples).

subcrt does not correctly identify the stable polymorph of some minerals at high temperature.

diagram causes an error while plotting stability field boundaries if the x and y resolutions are not identical.

Examples

```r
### Getting Started
## the 'thermo' object contains thermodynamic data and is also where
## user's settings (definition of chemical system) are stored
data(thermo)

## standard thermodynamic properties of species
subcrt("H2O")
subcrt("alanine")
# names of proteins have an underscore
subcrt("LYSC_CHICK")
# custom temperature range
T <- seq(8, 500, 100)
subcrt("H2O", T=T, P=1000)
# temperature - pressure grid
P <- seq(1000, 4000, 1000)
```
Calculate the chemical affinities of formation reactions of species. Do it for single values of temperature, pressure, ionic strength and chemical activities of the basis species, or as a function of one or more of these variables. Or, return other properties including standard molal Gibbs energies of basis species and species of interest, and standard molal Gibbs energies, equilibrium constants and activity products of formation reactions.
affinity

Arguments

... numeric, zero or more named arguments, used to identify the variables of interest in the calculations

property character, denoting the property to be calculated. Default is ‘A’, for chemical affinity of formation reactions of species of interest

sout list, output from subcrt function

exceed.Ttr logical, allow subcrt to compute properties for phases beyond their transition temperature?

return.buffer logical. If TRUE, and a buffer has been associated with one or more basis species in the system, return the values of the activities of the basis species calculated using the buffer (it is not necessary in this case to have defined any species of interest). Default is FALSE

balance character. This argument is passed to buffer to identify a conserved basis species (or ‘PBB’) in a chemical activity buffer. Default is ‘PBB’

iprotein numeric, indices of proteins in thermo$protein for which to calculate properties

logaNprotein numeric, logarithms of activities of proteins identified in iprotein

Details

affinity calculates the chemical affinities of reactions to form the species of interest from the basis species. The calculation of chemical affinities relies on the current definitions of the basis species and species of interest. It is possible to use the results of affinity to generate equilibrium activity diagrams using diagram.

The equation used to calculate chemical affinity $A$ is

$$ A = RT \ln \left( \frac{K}{Q} \right) $$

(Kondepudi and Prigogine, 1998), where $K$ denotes the equilibrium constant of the reaction and $Q$ stands for the activity product of the species in the reaction. (The equilibrium constant is related to standard Gibbs energy of reaction, $\Delta G^\circ_r$, by $\Delta G^\circ_r = -2.303RT \ln K$, where $R$ and $T$ stand for, respectively, the gas constant and temperature). With the approach of a given reaction to a state of equilibrium, the chemical affinity tends toward zero, or $K = Q$.

Valid properties are ‘A’ or NULL for chemical affinity, ‘logK’ or ‘logQ’ for logarithm of equilibrium constant and reaction activity product, or any of the properties available in subcrt except for ‘rho’. The properties returned are those of the formation reactions of the species of interest from the basis species. It is also possible to calculate the properties of the species of interest themselves (not their formation reactions) by setting the property to ‘G.species’, ‘Cp.species’, etc. Except for ‘A’, the properties of proteins or their reactions calculated in this manner are restricted to nonionized proteins.

Zero, one, or more leading arguments to the function identify which of the chemical activities of basis species, temperature, pressure and/or ionic strength to vary. The names of each of these arguments may be the formula of any of the basis species of the system, or ‘T’, ‘P’, ‘pe’, ‘pH’, ‘Eh’, or ‘IS’ (but names may not be repeated). To use the names of charged basis species such as ‘K+’ and ‘SO4-2’ as the arguments, they should be enclosed in quotes (see the example for aluminum speciation in diagram). The value of each argument is of the form c(min, max) or c(min, max, res) where min and max refer to the minimum and maximum values of variable identified by the name of the argument, and res denotes the resolution, or number of points along which to do
the calculations (this is assigned a default value of 128 if it is missing). For any arguments that refer to basis species, the numerical values are the logarithms of the activities of that basis species, or logarithms of fugacities if it is a gas. Unlike the energy function, the units of ‘T’ and ‘P’ in affinity are those the user has set using T.units and P.units (on program start-up these are °C and bar, respectively).

If one or more buffers are assigned to the definition of basis species, affinity calls buffer to calculate the logarithms of activities of these basis species from the buffer.

The iprotein and loga.protein arguments can be used to compute the chemical affinities of formation reactions of proteins that are not in the current species definition. This approach can be utilized in order to calculate the properties of many proteins in a fraction of the time it would take to calculate them individually. The appropriate basis species still must be defined prior to calling affinity. iprotein contains indices of desired proteins in thermo$protein; affinity adds to the species list the amino acid residues and and terminal H2O group (indicated by “RESIDUE” in thermo$protein) then calculates the properties of the reactions for the residues and terminal group, including ionization effects, and adds them together to get those of the indicated proteins.

In CHNOSZ version 0.9, energy gained a new argument ‘transect’ which is set to TRUE by energy.args when the length(s) of the variables is(are) greater than three. In this mode of operation, instead of performing the calculations on an n-dimensional grid, the affinities are calculated on an n-dimensional transect through chemical potential (possibly including T and/or P) space.

Value

For affinity, a list, elements of which are sout output from subcrt, property name of the calculated property (‘A’ for chemical affinity), basis and species definition of basis species and species of interest in effect at runtime, T and P temperature and pressure, in the system units of Kelvin and bar, set to numeric() (length=0) if either one is a variable, vars the names of the variables, vals the values of the variables (a list, one element for each variable), values the result of the calculation (a list, one element for each species, with names taken from the species index in thermo$obigt). The elements of the lists in vals and values are arrays of n dimensions, where n is the number of variables. The values of chemical affinity of formation reactions of the species are returned in dimensionless units (for use with decimal logarithms, i.e., A/2.303RT).

Names other than ‘T’ or ‘P’ in vars generally refer to basis species, and the corresponding vals are the logarithms of activity or fugacity. However, if one or more of pe, Eh or pH is among the variables of interest, vals holds the values of the those variables as indicated.

References


See Also

Usually, `equilibrate` is the next step in calculations of chemical equilibrium. The help for `buffer` has some examples of using chemical activity buffers. The guts of the calculations provided by `affinity` involve `energy.args` and `energy`, which normally are not part of the user interaction.

Examples

```r
## set up a system and calculate
## chemical affinities of formation reactions
basis(c("SiO2", "MgO", "H2O", "O2"), c(-5, -5, 0, 999))
species(c("quartz", "enstatite", "forsterite"))
# chemical affinities (A/2.303RT) at 25 deg C and 1 bar
affinity()

# at higher temperature and pressure
affinity(T=500, P=2000)

# ten different temperatures at one pressure
affinity(T=c(500, 1000, 1500, 2000), P=2000)

# at 25 temperatures and pressures
affinity(T=c(500, 1000, 1500, 2000), P=c(1000, 5000, 500))

# as a function of logarithm of activity of MgO
affinity(MgO=c(-10, -5, 10))

## equilibrium constants of formation reactions
affinity(property="logK")

# Standard molal Gibbs energies of species,
# user units (default: cal/mol)
affinity(property="G.species")

# Standard molal Gibbs energies of reactions
affinity(property="G")

## amino acid synthesis at low and high temperatures
## after Amend and Shock, 1998

# select the basis species and species of interest
# and set their activities, first for the 18 degree C case
basis(c("H2O", "CO2", "NH4+", "H2", "H+", "H2S"),
     log10(c(1, 1e-4, 5e-8, 2e-9, 5e-9, 1e-15))
species(sort(aminoacids("Z")),
     log10(c(3.9, 0.7, 1.1, 3.3, 0.5, 3.8, 1.0, 5.8, 1.2, 0.7,
            0.8, 1.0, 2.8, 0.5, 0.5, 4.6, 5.8, 0.6, 0.9, 2.8/1e9))
T <- 18
TK <- convert(T, "K")

# calculate A/2.303RT (dimensionless), convert to G of reaction (cal/mol)
a <- affinity(T=TK)
G.18.cal <- convert(unlist(args$values), "G", T=TK)

# convert to kJ/mol
```

G.18.kJ <- convert(G.18.cal, "J")/1000
# the 100 degree C case
basis(c("H2O", "CO2", "NH4+", "H2", "H+", "H2S"),
    log10(c(1, 2.2e-3, 2.9e-6, 3.4e-4, 1.9e-6, 1.6e-3)))
species(1:20, log10(c(2.8e-9, 5.8e-10, 7.9e-10, 2.4e-9, 3.6e-10,
    2.7e-9, 7.2e-10, 4.2e-9, 8.6e-10, 5.0e-10, 5.7e-10, 7.2e-10, 2.0e-9,
    3.6e-10, 3.6e-10, 3.3e-9, 4.2e-9, 4.3e-10, 6.5e-10, 2.0e-9)))
T <- 100
TK <- convert(T, "K")
a <- affinity(T=T)
G.100.cal <- convert(unlist(a$values), "G", T=TK)
G.100.kJ <- convert(G.100.cal, "J")/1000
# the average oxidation states of carbon
Z.C <- ZC(thermo$big$t$formula[thermo$species$i$species])
# put everything together a la Table 3 in the paper
print(out <- data.frame(G.18=G.18.kJ, G.100=G.100.kJ, Z.C=Z.C))
# make a plot; set units to get correct label
E.units("J")
plot(out$Z.C, out$G.18, pch=20, xlim=c(-1.1, 1.1), ylim=c(-200, 500),
    xlab=axis.label("ZC"), ylab=axis.label("DGr"))
points(out$Z.C, out$G.100, col="red", pch=20)
legend("topleft", pch=c(20, 20), col=c("black", "red"),
    legend=describe.property(c("T", "T"), c(18, 100)))
title(main="Amino acid synthesis, after Amend and Shock, 1998")
# 9 amino acids have negative delta Gr under hydrothermal conditions
# (cf. AS98 with 11; we are using more recent thermodynamic data)
stopifnot(sum(out$G.100 < 0)==9)
# reset units and species to run next examples
E.units("cal")
species(delete=TRUE)

## affinities of metabolic reactions
## after Amend and Shock, 2001, Fig. 7
## use aq state for all basis species (including O2)
basis(c("CO2", "H2", "NH3", "O2", "H2S", "H+"), "aq")
## we're going to make H2O
species("H2O")
## a function to create the plots
doplot <- function(T) {
  res <- 20
  # calculate affinity/2.303RT as a function of loga(H2) and loga(O2)
  a <- affinity(H2=c(-10, 0, res), O2=c(-10, 0, res), T=T)
  T.K <- convert(T, "K")
  # temperature in Kelvin
  acal <- convert(a$values[[1]], "G", T.K) # affinity (cal/mol)
  akJ <- convert(acal, "J")/1000 # affinity (kJ/mol)
  # now contour the values
  xyvals <- seq(-10, 0, length.out=res)
  contour(x=xyvals, y=xyvals, z=t(akJ), levels=seq(-150, -250, -20),
      labcex=1, xlab=axis.label("H2"), ylab=axis.label("O2"))
  # show the temperature
  legend("topleft", bty="white", cex=1,
      legend=describe.property("T", T, digits=0, ret.val=TRUE) )
}
# plot layout with space for title at top
layout(matrix(c(1, 1, 2, 3, 4, 5), ncol=2, byrow=TRUE), heights=c(1, 4, 4))
par(mar=c(0, 0, 0, 0))
plot.new()
# we use subcrt() to generate a reaction for titling the plot
rxnexpr <- describe.reaction(subcrt("H2O", 1))$reaction, states="all")
# also in the title is the property with its units
E.units("J")
Gexpr <- axis.label("DGGr", prefix="k")[[1]]
text(0.5, 0.6, substitute(paste(G="--for--r", list(G=Gexpr, r=rxnexpr)), cex=2)
text(0.5, 0.2, "after Amend and ShocK, 2001 Figure 7", cex=2)
# now make the plots
par(mar=c(3, 3, 0.5, 0.5), cex=1.3, mgp=c(2, 1, 0))
sapply(c(25, 55, 100, 150), doplot)
# affinity() can handle the three dimensions simultaneously
print(affinity(H2(c(-10, 0, 3), O2=c(-10, 0, 3), T=c(25, 150, 4))$values)
# this is so the plots in the next examples show up OK
E.units("cal")
layout(matrix(1))

## calculations along a transect: methanogenesis and biosynthetic
## reactions in hydrothermal systems, after Shock and Canovas, 2010
## this file has their mixing path results for Rainbow hydrothermal field
file <- system.file("extdata/cpetc/SC10_Rainbow.csv", package="CHNOSZ")
rb <- read.csv(file, check.names=FALSE)
# write all synthesis reactions in terms of these basis species
# it's okay not to set the activities of the basis species now
# because they'll be changing along with temperature
basis(c("CO2", "H2", "NH4+", "H2O", "H2S", "H+"))
# now a selection of the species from SC10, with activities equal to 1e-6
species(c("CH4", "formaldehyde", "ethylene", "glycolic acid",
    "n-nonanoic acid", "leucine", "aspartic acid", "tryptophan", "deoxyribose",
    "adenine", "cytosine"), -6)
# the exception is methane; unlike SC10 we use a constant activity 1e-3
# (accounting for variable activities of the species of interest here
# is possible but would require longer code ....)
species("CH4", -3)
# synchronized change of temperature and five basis activities
a <- affinity(T=rb$T, CO2=rb$co2, H2=rb$h2, NH4+=rb$nh4+, H2S=rb$h2s, pH=rb$ph)
# the tricky part: affinity() uses dimensionless values (A/2.303RT)
# but we want to show the values in cal/mol
a$values <- lapply(a$values, function(val) {
    -convert(val, "G", T=convert(a$vals[[1]], "K"))
})
# if we didn't have balance=1 here the values would be
# divided by the number of moles of CO2 in the reactions ...
diagram(a, balance=1, ylim=c(-100000, 100000), ylab=axis.label("A"),
    col=topo.colors(4), lwd=2)
# add a zero-affinity line and a title
abline(h=0, lty=2, lwd=2)
title(main="Affinities of organic synthesis, after Shock and Canovas, 2010")
Functions to Make Animations

Description

Make animated stability diagrams by creating a series of PNG files.

Usage

```r
anim.TCA(redox = list(O2 = c(-95, -60)), high.T = FALSE,
          nframes = 100, phlim = c(0,10), width = 420, height = 320)
anim.plasma(width=480, height=480)
anim.carboxylase(T = 25:125, ntop = 5, lcex = 0.8, width = 420, height = 320)
```

Arguments

- `redox` list, redox variable and limits
- `high.T` logical, overlay high-temperature diagram?
- `nframes` numeric, number of frames to be animated
- `phlim` numeric, pH limits to use for animation
- `width` numeric, width of plot device
- `height` numeric, height of plot device
- `T` numeric, temperature range for animation
- `ntop` numeric, number of names to show in legend
- `lcex` numeric, character expansion factor for legend

Details

These functions create a series of PNG figures that can be converted into an animated diagram. The PNG files are created in the ’png’ directory within the current working directory; the functions stop with an error if either this directory is not present or it is present but not empty. After making the PNG files, they are converted to an animated GIF using the ’convert’ tool from the ImageMagick software distribution ([http://www.imagemagick.org](http://www.imagemagick.org)), if it is available on the system. The system command is called using the `system` function on unix-alikes, and using `shell` on Windows platforms. When installing ImageMagick on Windows, be sure to leave the ’Add application directory to your system path’ option checked; this will make the ’convert’ command from ImageMagick available in the shell.

To ensure the results described below, each function here does remove any existing system definition by calling `data(thermo)`.

`anim.TCA` creates a series of figures showing how a logaH2O - logfO2 activity diagram for various species involved in the tricarboxylic acid (TCA) cycle changes as a function of pH. Alternatively, set `redox` to `list(H2=c(-26,0))` to draw a logaH2O - logaH2 diagram. The diagrams are made at 25 °C unless `high.T` is TRUE, in which case high-temperature (100 °C) stability fields are overlain.
The number of frames to be used for the animation (as pH increases ranges between the values specified in phlim) is given by nframes.

anim.plasma produces a series of equilibrium activity diagrams for proteins in human blood plasma, as a function of log aO2 and log aH2, at 25 °C. Unlike most other examples in CHNOSZ, the chemical potentials of hydrogen and oxygen in the system are represented by the activities of O2 and H2, and H2O is not used as a basis species. Therefore, the equilibrium activities of H2O vary by many orders of magnitude across these diagrams. The list of proteins is taken from Anderson and Anderson (2003); see the description for the data file AA03.csv in extdata. The first diagram shows the equilibrium predominance fields with all 71 listed proteins in the calculation. In each subsequent diagram, the protein whose predominance field occupies the greatest area on the diagram is removed. The range of heat.colors indicates the reported reference abundances of the proteins, with the deepest (reddest) colors corresponding to the highest abundances.

anim.carboxylase animates equilibrium rank-activity diagrams along a combined temperature and logaH2 gradient, or makes a single plot on the default device (without conversion to animated GIF) if a single temperature is provided. The proteins in the calculation are 24 carboxylases from a variety of organisms. There are 12 ribulose phosphate carboxylase and 12 acetyl-coenzyme A carboxylase; 6 of each type are from nominally mesophilic organisms and 6 from nominally thermophilic organisms, shown as blue and red symbols on the diagrams. The activities of hydrogen at each temperature are calculated using $\log a_{H_2(aq)} = -11 + 3/(40 \times T (°C))$; this equation comes from a model of relative stabilities of proteins in a hot-spring environment (Dick and Shock, 2011).

References


Examples

```r
## Not run:
# animate a stability diagram with a high-temperature overlay
anim.TCA(high=T=TRUE)
# using H2 instead of O2
anim.TCA(list(H2=c(-20,0)))
## End(Not run)

# using anim.carboxylase in non-animation mode
anim.carboxylase(T=100)
```
Define Basis Species

Description
Define the basis species of a chemical system. Change their physical states or chemical activities or fugacities. Get the stoichiometries of the basis species in selected species of interest.

Usage
```r
basis(species = NULL, state = NULL, logact = NULL, delete = FALSE)
put.basis(ispecies, logact)
mod.basis(species, state, logact)
preset.basis(key = NULL)
preset.logact(species)
```

Arguments
- `species` character, names or formulas of species, or numeric, indices of species
- `state` character, physical states or names of buffers
- `logact` numeric, logarithms of activities or fugacities
- `delete` logical, delete the current basis species definition?
- `ispecies` numeric, indices of species (rownumbers of `thermo$obigt`)
- `key` character, a keyword identifying a preset basis definition

Details
basis is used to define the basis species in a system of interest, and in many workflows is followed by calls to `species`, `affinity` and `diagram` for making equilibrium chemical activity diagrams. The other functions documented here are supporting functions for basis and generally are not intended to be called by the user.

The basis species represent the possible range of chemical compositions for all the species of interest. Any valid set of basis species used here must meet two conditions: 1) the number of basis species is the same as the number of chemical elements (including charge) in those species and 2) the square matrix representing the elemental stoichiometries of the basis species has a real inverse. Basis species might, but do not always (and not if a charged basis species is present), correspond to the thermodynamic components of a system.

To create a basis definition, call basis with the names or formulas of the basis species in the first argument. Alternatively, the first argument may consist of numeric values indicating the species indices (rownumbers in `thermo$obigt`), but a mixture of character and numeric values will generate an error. The special names ‘pH’, ‘pe’ and ‘Eh’ can be included in the species argument; they get translated into the names of the proton (‘H⁺’) and electron (‘e⁻’) as appropriate.

If the new basis definition meets all requirements, it is stored in `thermo$basis`, replacing any previous basis definition; `put.basis` does the actual storing of the basis definition.
The physical states or logarithms of activities of species in the basis definition can be changed directly using `mod.basis` but usually more conveniently by calling `basis` with the formulas of species that are in the basis set, or their species indices. If either of the second or third arguments to `basis` is of type character, it refers to the name of a state (if present in `thermo$obigt$state`) or to the name of a chemical activity `buffer` (if present in `thermo$buffers$name`). If either of these arguments is numeric it specifies the logarithms of activities (or fugacities for gases) of the basis species. In case `‘pH’`, `‘pe’` or `‘Eh’` are named, the logarithm of activity of the basis species is converted from these values. (For example, a value of 7 for pH is stored as a logarithm of activity of -7).

Whenever `basis` is called with NULL values of both state and `logact`, the new set of species, if they are a valid basis set, completely replaces any existing basis definition. If this occurs, any existing species definition (created by the `species` function) is deleted. However, `swapNbasis` can be used to change the species (the compositions and/or physical states thereof) in the basis set while maintaining the list of species of interest, with the added benefit of equivalence of the chemical potentials of the elements before and after the swap.

Call `basis` with `delete` set to TRUE to clear the basis definition. Any current basis definition (before being deleted) is returned by this call or by calling `basis` with all default arguments.

If the value of `basis` is one of the keywords in the following table, the corresponding set of basis species is loaded (defined in `presetNbasis`), and their activities set to reference values (defined in `presetNlogact`). This approach is used by many of the examples in the package. The basis species identified by these keywords are aqueous except for `H_2O (liq)`, `O_2 (gas)` and `Fe_2O_3` (hematite, cr1).

<table>
<thead>
<tr>
<th>CHNOS</th>
<th>CO_2, H_2O, NH_3, H_2S, O_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHNOS+</td>
<td>CO_2, H_2O, NH_3, H_2S, O_2, H^+</td>
</tr>
<tr>
<td>CHNOSe</td>
<td>CO_2, H_2O, NH_3, H_2S, e^-, H^+</td>
</tr>
<tr>
<td>CHNOPS+</td>
<td>CO_2, H_2O, NH_3, H_3PO_4, H_2S, e^-, H^+</td>
</tr>
<tr>
<td>MgCHNOPS+</td>
<td>Mg^{2+}, CO_2, H_2O, NH_3, H_3PO_4, H_2S, e^-, H^+</td>
</tr>
<tr>
<td>FeCHNOS</td>
<td>Fe_2O_3, CO_2, H_2O, NH_3, H_2S, O_2</td>
</tr>
<tr>
<td>FeCHNOS+</td>
<td>Fe_2O_3, CO_2, H_2O, NH_3, H_2S, O_2, H^+</td>
</tr>
</tbody>
</table>

**Value**

`basis` returns the value of `thermo$basis` after any modifications; or, if `delete` is TRUE, its value before deletion.

**See Also**

`info` to query the thermodynamic database in order to find what species are available. `makeup` is used by `basis` to generate the stoichiometric matrix from chemical formulas. `species` is the usual next step after `basis`. `swap.basis` is used to change the chemical compounds (species formulas) used in the basis definition while keeping the chemical potentials of the elements unaltered.

**Examples**

```r
## define basis species
# one, two and three element examples
```
buffer

Calculating Buffered Chemical Activities

Description

Calculate values of activity or fugacity of basis species buffered by an assemblage of one or more species.

Usage

```r
buffer(logK = NULL, ibasis = NULL, logact.basis = NULL, 
is.buffer = NULL, balance = "PBB")
mod.buffer(name, species = NULL, state = get("thermo")$opt$state, 
logact = -3)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logK</td>
<td>list, equilibrium constants of formation reactions of species, or NULL, indicates to load the species present in the buffer.</td>
</tr>
<tr>
<td>ibasis</td>
<td>numeric, which of the basis species whose activities are being calculated.</td>
</tr>
<tr>
<td>logact.basis</td>
<td>list, logarithms of activities of the basis species.</td>
</tr>
<tr>
<td>is.buffer</td>
<td>numeric, rownumbers of the buffering species in <code>thermo$species</code>.</td>
</tr>
<tr>
<td>balance</td>
<td>character, balance on this (name/formula of basis species or ‘PBB’) in buffer reactions.</td>
</tr>
</tbody>
</table>
buffer

name character, name of buffer to add to or find in thermo$buffers.
species character, names or formulas of species in a buffer.
state character, physical states of species in buffer.
logact numeric, logarithms of activities of species in buffer.

Details

A buffer is treated here as an assembly of one or more species whose presence constrains values of the chemical activity (or fugacity) of one or more basis species. To perform calculations for buffers, the user generally does not call buffer directly, but instead uses basis to associate the name of the buffer with one or more basis species. After this, calls to affinity will invoke the required calculations. The calculated values of the buffered activities can be retrieved by setting return.buffer to TRUE (in affinity). The maximum number of buffered chemical activities possible for any buffer is equal to the number of species in the buffer; however, the user may then elect to work with the values for only one or some of the basis species calculated with the buffer.

The identification of a conserved basis species (or other reaction balancing rule) is required in calculations for buffers of more than one species. For example, in the pyrite-pyrrhotite-magnetite buffer (FeS$_2$-FeS-Fe$_3$O$_4$) a basis species common to each species is one representing Fe; hence, when writing reactions between the species in this buffer one may conserve Fe while utilizing H$_2$S and O$_2$ as the variables of interest. The calculation for buffers attempts to determine which of the available basis species qualifies as a conserved quantity. This can be overridden with balance.

The default value of balance is 'pBB', which instructs the function to use the protein backbone group as the conserved quantity in buffers consisting of proteins, but has no overriding effect on the computations for buffers without proteins.

In the calculation of the buffered activities the user calls affinity which first determines the affinities of formation of all the species of interest (including species in the buffers) using the current reference activities of the basis species. affinity then calls buffer to calculate buffered values of the activities of basis species; the affinities of formation of all the species of interest are regenerated using the new (buffered) activities of basis species and returned to the user.

To view the available buffers, print the thermo$buffer object. Buffer definitions can be added to this dataframe with mod.buffer. It is possible to set the logarithms of activities of the species in the buffer through the logact argument; if this is missing unit activity is assigned to crystalline species in buffer, otherwise (for aqueous species) the default value of activity is $10^{-3}$. If name identifies an already defined buffer, this function modifies the logarithms of activities or states of species in that buffer, optionally restricted to only those species given in species.

It is possible to assign different buffers to different basis species, in which case the order of their calculation depends on their order in thermo$buffers. This function is compatible with systems of proteins, but note that for buffers made of proteins the buffer calculations presently use whole protein formulas (instead of residue equivalents) and consider nonionized proteins only (i.e., calculating values of pH buffered by proteins is so far not implemented).

Value

List of logarithms of chemical activities (or fugacities) of the basis species derived from chemical activities of the species in the buffer.
References


See Also

protein for an example using a buffer made of proteins.

Examples

```r
## list the buffers
thermo$buffers

# another way to do it, for a specific buffer
print(mod.buffer("PPM"))

## buffer made of one species
# calculate the activity of CO2 in equilibrium with
# (a buffer made of) acetic acid at a given activity
basis("CHNOS")
basis("CO2", "AC")
# what activity of acetic acid are we using?
print(mod.buffer("AC"))
# return the activity of CO2
(logaCO2 <- affinity(return.buffer=TRUE)$CO2)
stopifnot(all.equal(logaCO2, -7.05752136))
# as a function of oxygen fugacity
affinity(O2=c(-85,-70,4),return.buffer=TRUE)
# as a function of logfO2 and temperature
affinity(O2=c(-85,-70,4),T=c(25,100,4),return.buffer=TRUE)
# change the activity of species in the buffer
mod.buffer("AC",logac=-10)
affinity(O2=c(-85,-70,4),T=c(25,100,4),return.buffer=TRUE)
# see demos('CO2Ac') for a different strategy using the
# 'what' argument of diagram

## buffer made of three species
## Pyrite-Pyrhotite-Magnetite (PPM)
# specify basis species and initial activities
basis(c("FeS2", "H2S", "O2", "H2O"),c(0,-10,-50,0))
# note that the affinity of formation of pyrite,
# which corresponds to FeS2 in the basis, is zero
species(c("pyrite","pyrrhotite","magnetite"))
affinity(T=c(200,400,11),P=2000)$values
# setup H2S and O2 to be buffered by PPM
basis(c("H2S","O2"),c("PPM","PPM"))
# inspect values of H2S activity and O2 fugacity
affinity(T=c(200, 400, 11), P=2000, return.buffer=TRUE, exceed.Ttr=TRUE)
# now, the affinities of formation reactions of
# species in the buffer are all equal to zero
print(a <- affinity(T=c(200, 400, 11), P=2000,
    exceed.Ttr=TRUE)$values)
```
for(i in 1:length(a)) stopifnot(isTRUE(
  all.equal(as.numeric(a[i]),rep(0,length(a[i]))))

## buffer made of one species: show values of logfO2 on an
## Eh-pH diagram; after Garrels, 1960, Figure 6
basis("CHNOS")
# here we will buffer the activity of the electron by O2
mod.buffer("O2","O2","gas",999)
basis("e-","O2")
# start our plot, then loop over values of logfO2
thermo.plot.new(xlim=c(0,14),ylim=c(-0.8,1.2),
  xlab="pH",ylab=axis.label("Eh"))
# the upper and lower lines correspond to the upper
# and lower stability limits of water
logfO2 <- c(-20,-40,-60,-83.1)
for(i in 1:5) {
  # update the logarithm of fugacity (logact) of O2 in the buffer
  mod.buffer("O2","O2","gas",logfO2[i])
  # get the values of the logarithm of activity of the electron
  a <- affinity(pH=c(0,14,15),return.buffer=TRUE)
  # convert values of pe (-logact of the electron) to Eh
  Eh <- convert(-as.numeric(a$e-"e"),"Eh")
  lines(seq(0,14,length.out=15),Eh)
  # add some labels
  text(seq(0,14,length.out=15)[i*2+2],Eh[i*2+2],
    paste("logfO2=",logfO2[i],sep=""))
}
title(main=paste("Relation between logfO2(g), Eh and pH at\n",
  "25 degC and 1 bar. After Garrels, 1960\""))

## buffer made of two species
## conditions for metastable equilibrium among
## CO2 and acetic acid. note their starting activities:
print(mod.buffer("CO2-AC"))
basis("CHNOS")
basis("O2","CO2-AC")
affinity(return.buffer=TRUE)  # logfO2 = -75.94248
basis("CO2",123)  # what the buffer reactions are balanced on
affinity(return.buffer=TRUE)  # unchanged
# consider more oxidizing conditions
mod.buffer("CO2-AC",logact=c(0,-10))
affinity(return.buffer=TRUE)

---

**Equilibrium Chemical Activity Diagrams**

**Description**

Plot equilibrium chemical activity (1-D speciation) or equal-activity (2-D predominance) diagrams as a function of chemical activities of basis species, temperature and/or pressure.
Usage

diagram(eout, what = "loga.equil", alpha = FALSE, normalize = FALSE,
        balance=NULL, groups=as.list(1:length(eout$values)), xrange=NULL,
        mar=NULL, yline=par("mgp")[1]+0.7, side=1:4,
        xlog=TRUE, xlim=NULL, ylim=NULL, xlab=NULL, ylab=NULL,
        cex=par("cex"), cex.names=1, cex.axis=par("cex"),
        lty=NULL, lwd=par("lwd"), dotted=0,
        bg=par("bg"), col=par("col"), col.names=par("col"), fill=NULL,
        names=NULL, main=NULL, legend.x="topright", add=FALSE, plot.it=TRUE)
strip(affinity, ispecies = NULL, col = NULL, ns = NULL,
     xticks = NULL, ymin = -0.2, xpad = 1, cex.names = 0.7)
find.tp(x)

Arguments

eout list, object returned by equilibrate or affinity
what character, what property to calculate and plot
alpha logical, for speciation diagrams, plot degree of formation instead of activities?
normalize logical, normalize chemical formulas or reaction properties by the balance vector?
balance character, balancing constraint; see balance
groups list of numeric, groups of species to consider as a single effective species
xrange numeric, range of x-values between which predominance field boundaries are plotted
mar numeric, margins of plot frame
yline numeric, margin line on which to plot the y-axis name
side numeric, which sides of plot to draw axes
xlim numeric, limits of x-axis
ylim numeric, limits of y-axis
xlab character, label to use for x-axis
ylab character, label to use for y-axis
ylog logical, use a logarithmic y-axis (on 1D degree diagrams)?
cex numeric, character expansion (scaling relative to current)
cex.names numeric, character expansion factor to be used for names of species on plots
cex.axis numeric, character expansion factor for names of axes
lty numeric, line types to be used in plots
lwd numeric, line width
dotted numeric, how often to skip plotting points on predominance field boundaries (to gain the effect of dotted or dashed boundary lines)
bg character, background color for legend
col character, color of activity lines (1D diagram) or predominance field boundaries (2D diagram), or colors of bars in a strip diagram (strip)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>col.names</td>
<td>character, colors for labels of species</td>
</tr>
<tr>
<td>fill</td>
<td>character, colors used to fill predominance fields</td>
</tr>
<tr>
<td>names</td>
<td>character, names of species for activity lines or predominance fields</td>
</tr>
<tr>
<td>main</td>
<td>character, a main title for the plot; NULL means to plot no title</td>
</tr>
<tr>
<td>legend.x</td>
<td>character, description of legend placement passed to legend</td>
</tr>
<tr>
<td>add</td>
<td>logical, add to current plot?</td>
</tr>
<tr>
<td>plot.it</td>
<td>logical, make a plot?</td>
</tr>
<tr>
<td>affinity</td>
<td>list, object returned by affinity</td>
</tr>
<tr>
<td>ispecies</td>
<td>numeric, which species to consider (default of NULL is to consider all species)</td>
</tr>
<tr>
<td>ns</td>
<td>numeric, numbers of species, used to make inset plots for strip diagrams</td>
</tr>
<tr>
<td>xticks</td>
<td>numeric, location of supplemental tick marks on x-axis</td>
</tr>
<tr>
<td>ymin</td>
<td>numeric, lower limit of y-axis</td>
</tr>
<tr>
<td>xpad</td>
<td>numeric, amount to extend x-axis on each side</td>
</tr>
<tr>
<td>x</td>
<td>matrix, value of the predominant list element from diagram</td>
</tr>
</tbody>
</table>

**Details**

diagram takes as its primary input the results from `equilibrate` and displays diagrams representing the equilibrium chemical activities of the species. 0-D diagrams, at a single point, are shown as barcharts. 1-D diagrams, for a single variable on the x-axis, are plotted as lines. 2-D diagrams, for two variables, are plotted as predominance fields. The allowed variables are any that `affinity` accepts: temperature, pressure, or the chemical activities of the basis species.

If alpha is TRUE, the fractional degrees of formation (ratios of activities to total activity) are plotted. This setting is useful for visualizing the molalities (activities for ideal interactions) of monomer groups in a system of biopolymers (e.g. proteins). If `groups` is supplied, the activities of the species identified in each numeric vector of this list are summed together and subsequently treated as a single species; the names of the new species are taken from the names of this list.

For 1-D diagrams, the default setting for the y-axis is a logarithmic scale (unless alpha is TRUE) with limits corresponding to the range of logarithms of activities (or 0,1 if alpha is TRUE); these actions can be overridden by `ylog` and `ylim`. A `legend` is placed at the location identified by `legend.x`, or omitted if `legend.x` is FALSE. If `legend.x` is NA, instead of a legend, the lines are labeled with the names of the species near the maximum value. The line type and line width can be controlled with `lty` and `lwd`, respectively.

On 2-D diagrams, the fields represent the species with the highest equilibrium activity, after any normalize or alpha operations. `fill` determines the color of the predominance fields, `col` that of the boundary lines. By default, `heat.colors` are used to fill the predominance fields in diagrams on the screen plot device. The style of the boundary lines on 2-D diagrams can be altered by supplying one or more non-zero integers in `dotted`, which indicates the fraction of line segments to omit; a value of ‘1’ or NULL for dotted has the effect of not drawing the boundary lines.

For all diagrams, the names of the species and their colors in `col.names` can be changed, as can `cex`, `cex.names`, and `cex.axis` to adjust the overall character expansion factors (see `par`) and those of the species names and axis labels. The x- and y-axis labels are automatically generated unless they are supplied in `xlab` and `ylab`. A new plot is started unless `add` is TRUE. If `plot.it` is FALSE,
no plot will be generated but all the intermediate computations will be performed and the results returned.

`diagram` also accepts the output from `affinity`, for which three actions are possible: 1) plot a property of a reaction, such as the equilibrium constant (\(\log K\)) (0-D or 1-D); 2) plot the equilibrium activity of a selected basis species in all of the formation reactions (0-D, 1-D or 2-D); 3) plot predominance fields, based on the relative magnitudes of the affinities of the formation reactions (2-D only).

Some of the arguments have different effects when the output from `affinity` is being used instead of the equilibrium activities from `equilibrate`. If `what` is missing, option (1) is selected if the number of dimensions is 0 or 1, and option (3) is selected if the number of dimensions is 2. The latter is referred to as the maximum affinity method. In cases where it applies (see Warning), the maximum affinity method is much faster than an equilibrium calculation. `balance` is the option, sent to `balance`, that determines the balancing coefficients used in the normalization (this argument has no effect on the calculations of equilibrium activities.) If `what` is the name of a basis species, it refers to option (2) above. A contour plot is made in the case of 2-D diagrams of the equilibrium activity of a basis species (see `demos("CO2Ac")`), and only the first species of interest is used in the calculation; a warning is produced if there is more than one.

A different incarnation of 1-D speciation diagrams is provided by `strip`. This function generates any number of strip diagrams in a single plot. The diagrams are made up of colors bars whose heights represent the relative abundances of species; the color bars are arranged in order of abundance and the total height of the stack of colors bars is constant. If `ispecies` is a list, the number of strip diagrams is equal to the number of elements of the list, and the elements of this list are numeric vectors that identify the species to consider for each diagram. The strips are labeled with the names of `ispecies`. If `col` is NULL, the colors of the bars are generated using `rainbow`. Supplemental ticks can be added to the x-axis at the locations specified in `xtick`; they are larger than the standard ticks and have colors corresponding to those of the color bars. `ymin` can be decreased in order to add more space at the bottom of the plot, and `xpad` can be changed in order to increase or decrease the size of the x-axis relative to the width of the strips. An inset dot-and-line plot is created below each strip if `ns` is given. This argument has the same format as `ispecies`, and can be used e.g. to display the relative numbers of species for comparison with the stability calculations.

`find.tp` finds the locations in a matrix of integers that are surrounded by the greatest number of different values. The function counts the unique values in a 3x3 grid around each point and returns a matrix of indices (similar to `which(...)`, `arr.ind = TRUE`) for the maximum count (ties result in more than one pair of indices). It can be used with the output from `diagram` for calculations in 2 dimensions to approximately locate the triple points on the diagram.

**Value**

For speciation diagrams, an `invisible` list of the chemical activities of the species, or their degrees of formation (if `alpha` is TRUE), at each point. For predominance diagrams, an invisible list with elements `species`, the dataframe describing the species, `out`, which species predominates at each grid point, and `A`, a list of the calculated values of the chemical affinity (per balanced quantity) (\(\log 10\) dimensionless) at each point.

**Warning**

The maximum affinity method is premised on the calculation of the affinities of formation reactions at equal activities of the species of interest. Then, the species with the highest affinity of formation,
after normalization by the balancing coefficients, corresponds to the predominant species in an equilibrium calculation. The examples below “work” because they are relatively simple - the balancing coefficients are unity or all the same value (aqueous aluminum example), or the species are solids with unit activities (the mineral examples). The examples shown for proteins elsewhere also take the balancing coefficients to unity, after normalizing by protein length. However, if aqueous species are present with different balancing coefficients, the maximum affinity method is not dependable, as shown in the TCA metabolite example below.

References


See Also

Other examples are present in the help for protein and buffer, and even more can be found in demos.

Examples

```r
### 1-D diagrams: logarithms of activities

## Aqueous sulfur species (after Seewald, 1997 and 2001)
basis("CHNOS+")
basis("pH", 5)
species(c("H2S", "S2-2", "S3-2", "S2O3-2", "S2O4-2", "S3O6-2", "S5O6-2", "S2O6-2", "HSO3-", "SO2", "HSO4-"))
a <- affinity(O2=c(-50, -15), T=325, P=350)
```
e <- equilibrate(a, loga.balance=-2)
diagram(e, ylim=c(-30, 0), legend.x="topleft")
title(main=paste("Aqueous sulfur speciation, 325 degC, 350 bar\\n", "After Seewald, 1997"))
# try it with and without the loga.balance argument (total activity of
# the balanced quantity, in this case H2S aka sulfur)

## Degrees of formation of ionized forms of glycine
## After Fig. 1 of Aksu and Doyle, 2001
basis("CHNOS+")
species(ispecies <- info(c("glycinium", "glycine", "glycinate")))
a <- affinity(pH=c(0, 14))
e <- equilibrate(a)
diagram(e, alpha=TRUE, lwd=1)
title(main=paste("Degrees of formation of aqueous glycine species\\n", "after Aksu and Doyle, 2001"))

## Degrees of formation of ATP species as a function of
## temperature, after LaRowe and Helgeson, 2007, Fig. 10b
# to make a similar diagram, activity of Mg+2 here is set to
# 10^-4, which is different from LH07, who used 10^-3 total molality
basis(c("CO2", "NH3", "H2O", "H3PO4", "O2", "H+", "Mg+2"),
c(999, 999, 999, 999, 999, 999, -5, -4))
species(c("HATP-3", "H2ATP-2", "MgATP-2", "MgHATP-"))
a <- affinity(T=c(0, 120, 25))
e <- equilibrate(a)
diagram(e, alpha=TRUE)
title(main=paste("Degrees of formation of ATP species\\n", "pH=5, log(aMg2)=3. After LaRowe and Helgeson, 2007"),
cex.main=0.9)

### 2-D diagrams: predominance diagrams
### these use the maximum affinity method

## Fe-S-O at 200 deg C, after Helgeson, 1970
basis(c("Fe", "O2", "S2"))
species(c("iron", "ferrous-oxide", "magnetite",
"hematite", "pyrite", "pyrrhotite"))
# include the high-temperature phase of pyrrhotite
species("pyrrhotite", "cr2")
a <- affinity(S2=c(-50, 0), O2=c(-90, -10), T=200)
diagram(a, fill="heat")
title(main=paste("Fe-S-O, 200 degrees C, 1 bar",
"After Helgeson, 1970", sep="\\n"))

### pe-pH diagram for hydrated iron sulfides,
### goethite and pyrite, after Majzlan et al., 2006
# add some of these species to the database
add.obigt()
basis(c("Fe+2", "SO4-2", "H2O", "H+", "e-"),
c(0, log10(3), log10(0.75), 999, 999))
species(c("rhomboclase", "ferricopiapite", "hydronium jarosite",
"goethite", "melanterite", "pyrite"))
a <- affinity(pH=c(-1, 4, 256), pe=c(-5, 23, 256))
d <- diagram(a, main="Fe-S-O-H, after Majzlan et al., 2006")
# the first four species show up along the top of the diagram
stopifnot(all.equal(unique(d$d$predominant)[256,]), 1:4)
water.lines(yaxis="pe")
text(3, 22, describe.basis(thermo$basis[2:3,], digits=2, oneline=TRUE))
text(3, 21, describe.property(c("T", "P"), c(25, 1), oneline=TRUE))
# reset the database
data(thermo)

## Aqueous Aluminum Species F-/OH-, after Tagirov and Schott, 2001

## some of the species are not in the default database
add.obigt()

# the 999s have no effect on the diagram:
# pH and log_a(F-) are plotting variables
# O2 is not in the formation reactions
# Al3+ is the balanced quantity
basis(c("Al3+", "F-", "H+", "O2", "H2O"), c(rep(999, 4), 0))
species(c("Al3+", "Al(OH)4-", "AlOH+2", "Al(OH)2+", "Al(OH)3",
        "AlF+2", "AlF2-", "AlF3", "AlF4-", "Al(OH)2F2-", "Al(OH)2F",
        "AlOHF2"), "aq")
a <- affinity(pH=c(0, 10), "F-"=c(-1, -9), T=200)
diagram(a, fill=heat)

# Temperature-Pressure: kyanite-sillimanite-andalusite
# cf. Fig. 49 of Helgeson et al., 1978
# this is a system of one component (Al2SiO5), but we need the same
# number of basis species as elements; and avoid using H2O or other
# aqueous species because of T/P limits of the water() calculations;
basis(c("corundum", "quartz", "oxygen"))
species(c("kyanite", "sillimanite", "andalusite"))
# database has transition temperatures of kyanite and andalusite
# at 1 bar only, so we permit calculation at higher temperatures
a <- affinity(T=c(200, 900, 99), P=c(0, 9000, 101), exceed.TTr=TRUE)
d <- diagram(a, fill=NULL)
bexpr <- sapply(c("Al2O3", "SiO2", "H2O"), expr.species, simplify=FALSE)
btext <- substitute(Al2O3 - SiO2 - H2O, unlist(bexpr))
mtitle(c(as.expression(btext), "after Helgeson et al., 1978"))

# find the approximate position of the triple point
tp <- find.tp(d$d$predominant)

# some testing of the overall geometry
stopifnot(species()[name[d$d$predominant][1, 1]]=="andalusite")
stopifnot(species()[name[d$d$predominant][1, 101]]=="kyanite")
stopifnot(species()[name[d$d$predominant][99, 101]]=="sillimanite")

### other examples
## a case where the maximum affinity method doesn’t reproduce an equilibrium predominance diagram

basis("CHNOS+")

# this adds data for some metabolites in the TCA cycle
# from Dalla-Betta and Schulte, 2010
add.obigt()

species(c("oxaloacetate-2", "malate-2", "fumarate-2",
   "a-ketoglutarate-2", "citrate-3"))

a <- affinity(O2=c(-80, -60), H2O=c(-5, 5))
diagram(a, dotted=1, fill="heat")
e <- equilibrate(a)
diagram(e, add=TRUE, names=NULL, col="purple")
e <- equilibrate(a, normalize=TRUE)
diagram(e, add=TRUE, names=NULL)
title(main=paste("maximum affinity method (fields)\n",
   "equilibrium calculations (lines)"))
data(thermo)

## calculate the equilibrium logarithm of activity of a basis species in different reactions
basis("CHNOS")

species(c("ethanol", "lactic acid", "deoxyribose", "ribose"))
a <- affinity(T=c(0, 150))
diagram(a, what="O2", legend.x="topleft", col=rev(rainbow(4)), lwd=2)
title(main="equilibrium logfO2 for 1e-3 mol/kg of CO2 and ... ")

## using strip()

## proteins from different mammals
organisms <- c("BOVIN", "CANFA", "HUMAN", "MOUSE", "PIG")
proteins <- c("AQP1", "MYG", "PS3")
basis("CHNOS+")

species(rep(proteins, each=5), organisms)
a <- affinity(O2=c(-85, -65, 128))
ispecies <- list(1:5, 6:10, 11:15)
desc <- c("(Aquaporin-1)", "(Myoglobin)",
   "(Cellular tumor antigen p53")")
names(ispecies) <- paste(proteins, desc)
col <- rainbow(5)
strip(a, ispecies=ispecies, yline=-0.7, col=col)

legend("bottomright", legend=organisms, col=col,
   lty=1, lwd=4, bty="n")
title(main=paste("Equilibrium degrees of formation of",
   "proteins from different mammals", sep="\n"))
Description
Calculate thermodynamic properties using the revised Helgeson-Kirkham-Flowers (HKF) equations of state for aqueous species, or using a generic heat capacity equation for crystalline, gas, and liquid species.

Usage

```r
clg(property = NULL, T = 298.15, P = 1, ghs = NULL, eos = NULL)
hkf(property = NULL, T = 298.15, P = 1, ghs = NULL, eos = NULL,
    contrib = c("n","s","o"), H2O.PT = NULL, H2O.PrTr = NULL,
    domega = TRUE)
gfun(rhohat, Tc, P, alpha, daldT, beta)
```

Arguments

- `property` character, name(s) of properties to calculate
- `T` numeric, temperature(s) at which to calculate properties (K)
- `P` numeric, pressure(s) at which to calculate properties (bar)
- `ghs` dataframe, values of the standard molal Gibbs energy and enthalpy of formation from the elements and entropy at 25 °C and 1 bar
- `eos` dataframe, values of the equations-of-state parameters
- `contrib` character, which contributions to consider in the revised HKF equations equations of state: (n)onsolvation, (s)olvation (the ω terms), or (o)rigination contributions (i.e., the property itself at 25 °C and 1 bar). Default is c("n", "s", "o"), for all contributions
- `H2O.PT` dataframe, values of the electrostatic properties of water at the temperature(s) and pressure(s) of interest
- `H2O.PrTr` dataframe, values of the electrostatic properties of water at the reference temperature and pressure
- `domega` logical, calculate the T and P derivatives of omega?
- `rhohat` numeric, density of water (g/cm3)
- `Tc` numeric, temperature (°C)
- `alpha` numeric, coefficient of isobaric expansivity (K^-1)
- `daldT` numeric, temperature derivative of coefficient of isobaric expansivity (K^-2)
- `beta` numeric, coefficient of isothermal compressibility (bar^-1)

Details
The equations of state permit the calculation of the standard molal properties of species as a function of temperature and pressure. For interactive use, `subcrt` is usually more convenient than calling these functions directly.

The property argument is required and refers to one or more of ‘G’, ‘H’, ‘S’, ‘Cp’ and ‘V’, and for aqueous species only, ‘kt’ and ‘E’. The units of these properties are the first ones shown in the description for `subcrt`. The names of the properties are matched without regard to case.
The revised HKF equations of state (Helgeson et al., 1981; Tanger and Helgeson, 1988; Shock and Helgeson, 1988) are incorporated in hkf. The equations-of-state parameters are \(a_1, a_2, a_3, a_4, c_1, c_2, \text{omega} \) and \(Z\); the units of these parameters are as indicated for thermo$sobigt$, sans the order of magnitude multipliers. Note that the equation-of-state parameter \(\text{omega}\) (appearing in the \(g\)-function for the temperature derivatives of the omega parameter; Shock et al., 1992) is taken from thermo$sobigt$ and not from the makeup of the species, although in most cases the two values are coincident. \(H_2O.PT\) and \(H_2O.PrTr\) are optional arguments that contain the electrostatic properties of \(H_2O\) required for the calculations. If either of these is NULL (the default), the required values are retrieved using water.

Unless \(\text{omega}\), the value of which is recycled to the number of species (rows of ghs and eos), is FALSE for any of the species, the temperature and pressure derivatives of the \(\text{omega}\) parameter for charged species (where \(Z \neq 0\)) are calculated using the \(g\)- and \(f\)-functions described by Shock et al., 1992 and Johnson et al., 1992, and coded here in gfun. This option is currently blocked when the IAPWS-95 equations are activated (see water).

The parameters in the cgl equations of state for crystalline, gas and liquid species (except liquid water) include \(v, a, b, c, d, e, f\) and \(\lambda\). The terms denoted by \(a, b\) and \(c\) correspond to the Maier-Kelley equation for heat capacity (Maier and Kelley, 1932); the additional terms are useful for representing heat capacities of minerals (Robie and Hemingway, 1995) and gaseous or liquid organic species (Helgeson et al., 1998). The standard molal volumes (‘\(v\)’) of species in these calculations are taken to be independent of temperature and pressure.

For both hkf and cgl, if at least one equations-of-state parameter for a species is provided, any NA values of the other parameters are reset to zero. If all equations-of-state parameters are NA, but values of ‘\(cp\)’ and/or ‘\(v\)’ are available, those values are used in the integration of ‘\(g\)’, ‘\(h\)’ and ‘\(s\)’ as a function of temperature.

The \(T\) and \(P\) arguments (and \(\rho\)ohat, \(Tc\), \(alpha\), \(dalpha\) for gfun) should all be the same length; the functions perform no argument validating.

**Value**

A list of length equal to the number of species (i.e., number rows of supplied ghs and eos values). Each element of the list contains a dataframe, each column of which corresponds to one of the specified properties; the number of rows is equal to the number of pressure-temperature points.

**Warning**

The temperature and pressure range of validity of the revised HKF equations of state for aqueous species corresponds to the stability region of liquid water or the supercritical fluid at conditions between 0 to 1000 °C and 1 to 5000 bar (Tanger and Helgeson, 1988; Shock and Helgeson, 1988). The hkf function does not check these limits and will compute properties as long as the requisite electrostatic properties of water are available. There are conceptually no temperature limits (other than 0 Kelvin) for the validity of the cgl equations of state. However, the actual working upper temperature limits correspond to the temperatures of phase transitions of minerals or to those temperatures beyond which extrapolations from experimental data become highly uncertain. These temperature limits are stored in the thermodynamic database for some minerals, but cgl ignores them; however, subcrt warns if they are exceeded.
References


See Also

info for retrieving equations of state parameters from the thermodynamic database, water for equations of state of water, subcrt for calculations that use these equations.

Examples

```r
## aqueous species
a <- info(info("methane","aq"))
hkf(property="Cp",ghs=a,eos=a)
# the non-solvation heat capacity
hkf(property="Cp",ghs=a,eos=a,contrib="n")
# at different temperature and pressure
hkf(property="Cp",ghs=a,eos=a,T=(373.15,473.15),P=1000)

## crystalline, gas, liquid species
a <- info(info("methane","gas"))
cgl(property="Cp",ghs=a,eos=a)
# melting and vaporization of n-octane
```
EOSregress

**Description**

Fit experimental volumes and heat capacities using regression equations. Possible models include the Helgeson-Kirkham-Flowers (HKF) equations of state, or other equations defined using any combination of terms derived from the temperature, pressure and thermodynamic and electrostatic properties of water and/or user-defined functions of temperature and pressure.

**Usage**

```r
EOSregress(exptdata, var = "", T.max = 9999)
EOSvar(var, T, P)
EOScalc(coefficients, T, P)
EOSplot(exptdata, var = NULL, T.max = 9999, T.plot = NULL,
        fun.legend = "topleft", coefficients = NULL)
EOSlab(var, coeff = "")
EOScoeffs(species, property)
```

**Arguments**

- **exptdata**: dataframe, experimental data
- **var**: character, name(s) of variables in the regression equations
- **T.max**: numeric, maximum temperature for regression, in degrees Kelvin
- **T**: numeric, temperature in degrees Kelvin
- **P**: numeric, pressure in bars
- **T.plot**: numeric, upper limit of temperature range to plot
- **fun.legend**: character, where to place legend on plot
- **coefficients**: dataframe, coefficients to use to make line on plot
- **coeff**: numeric, value of equation of state parameter for plot legend
- **species**: character, name of aqueous species
- **property**: character, ‘Cp’ or ‘V’
Details

EOSregress uses `lm` to regress the experimental heat capacity or volume data in `exptdata`, which is a data frame with columns ‘T’ (temperature in degrees Kelvin), ‘P’ (pressure in bars), and ‘Cp’ or ‘V’ (heat capacity in cal/mol.K or volume in cm³/mol). Only data below the temperature of T_max are included in the regression. The regression formula is specified by a vector of names in `var`. The names of the variables can be any combination of the following (listed in the order of search):

- variables listed in the following table, any available property of water (e.g. ‘V’, ‘alpha’, ‘QBorn’), or the name of a function that can be found using `get` in the default environment (e.g. a function defined by the user in the global environment; the arguments of the function should be `T` and `P`; see example).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T (temperature)</td>
</tr>
<tr>
<td>P</td>
<td>P (pressure)</td>
</tr>
<tr>
<td>TTheta</td>
<td>(T – Θ) (Θ = 228 K)</td>
</tr>
<tr>
<td>invTTheta</td>
<td>1/(T – Θ)</td>
</tr>
<tr>
<td>TTheta2</td>
<td>(T – Θ)²</td>
</tr>
<tr>
<td>invTTheta2</td>
<td>1/(T – Θ)²</td>
</tr>
<tr>
<td>invPpsi</td>
<td>1/(P + Ψ) (Ψ = 2600 bar)</td>
</tr>
<tr>
<td>invPpsiTTheta</td>
<td>1/((P + Ψ)(T – Θ))</td>
</tr>
<tr>
<td>TXBorn</td>
<td>TX (temperature times X Born function)</td>
</tr>
<tr>
<td>drho.dT</td>
<td>dρ/dT (temperature derivative of density of water)</td>
</tr>
<tr>
<td>V.kT</td>
<td>VκT (volume times isothermal compressibility of water)</td>
</tr>
</tbody>
</table>

`EOSvar` calculates the value of the variable named `var` (defined as described above) at the specified `T` (temperature in degrees Kelvin) and `P` (pressure in bars). This function is used by `EOSregress` to get the values of the variables used in the regression.

`EOScalc` calculates the predicted heat capacities or volumes using coefficients provided by the result of `EOSregress`, at the temperatures and pressures specified by `T` and `P`.

`EOSplot` takes a table of data in `exptdata`, runs `EOSregress` and `EOScalc` and plots the results. The experimental data are plotted as points, and the calculated values as a smooth line. The point symbols are filled circles where the calculated value is within 10% of the experimental value; open circles otherwise.

`EOSlab` produces labels for the variables listed above that can be used as `expression` in plots. The value of `coeff` is prefixed to the name of the variable (using `substitute`, with a multiplication symbol). For the properties listed in the table above, and selected properties listed in `water`, the label is formatted using `plotmath` expressions (e.g., with italicized symbols and Greek letters). If `var` is a user-defined function, the function can be given a ‘label’ attribute to provide `plotmath`-style formatting; in this case the appropriate multiplication or division symbol should be specified (see example below).

`EOScoeffs` retrieves coefficients in the Helgeson-Kirkham-Flowers equations from the thermodynamic database (thermo$obigt$) for the given aqueous species. If the property is ‘Cp’, the resulting data frame has column names of ‘(Intercept)’, ‘invTTheta2’ and ‘TX’, respectively holding the coefficients $c_1$, $c_2$ and $ω$ in the equation $Cp^o = c_1 + c_2/(T – Θ)^2 + ωTX$. If the property is ‘V’, the data frame has column names of ‘(Intercept)’, ‘invTTheta’ and ‘Q’, respectively holding the coefficients $σ$, $ξ$ and $−ω$ in $V^o = σ + ξ/(T – Θ) − ωQ$.

The motivation for writing these functions is to explore alternatives or possible modifications to the revised Helgeson-Kirkham-Flowers equations applied to aqueous nonelectrolytes. As pointed out
by Schulte et al., 2001, the functional forms of the equations do not permit retrieving values of the solvation parameter (ω) that closely represent the observed trends in both heat capacity and volume at high temperatures (above ca. 200 °C).

Value

For EOSregress, an object of class “lm”. EOSvar and EOScalc both return numeric values. EOScoeffs returns a data frame.

References


See Also

See lm for the details of the regression calculations.

Examples

```r
## fit experimental heat capacities of CH4
## using revised Helgeson-Kirkham-Flowers equations
# read the data from Hnedkovsky and Wood, 1997
f <- system.file("extdata/cpetc/Cp.CH4.HW97.csv", package="CHNSZ")
d <- read.csv(f)
# have to convert J to cal and MPa to bar
d$Cp <- convert(d$Cp, "cal")
d$p <- convert(d$p, "bar")
# specify the terms in the HKF equations
var <- c("invTTheta2", "TXBorn")
# perform regression, with a temperature limit
EOSlm <- EOSregress(d, var, T.max=600)
# the result is within 10% of the accepted
# values of c1, c2 and omega for CH4(aq)
CH4coeffs <- EOScoeffs("CH4", "Cp")
dcoeffs <- EOSlm$coefficients - CH4coeffs
stopifnot(all(abs(dcoeffs/CH4coeffs) < 0.1))
## make plots comparing the regressions
## here with the accepted EOS parameters of CH4
par(mfrow=c(2,2))
EOSplot(d, T.max=600)
title("Cp of CH4(aq), fit to 600 K")
legend("bottomleft", pch=1, legend="Hnedkovsky and Wood, 1997")
EOSplot(d, coefficients=CH4coeffs)
title("Cp from EOS parameters in database")
EOSplot(d, T.max=600, T.plot=600)
title("Cp fit to 600 K, plot to 600 K")
EOSplot(d, coefficients=CH4coeffs, T.plot=600)
title("Cp from EOS parameters in database")
```
# continuing from above, with user-defined variables
invTTTheta3 <- function(T, P) (2*T)/(T-T+thermo$opt$Theta)^3
invTX <- function(T, P) 1/T+water("XBorn", T=T, P=P)[1]
# print the calculated values of invTTTheta3
EOSvar("invTTTheta3", d$T, d$P)
# use invTTTheta and invTX in a regression
var <- c("invTTTheta3", "invTX")
EOSregress(d, var)
# give them a "label" attribute for use in the legend
attr(invTTTheta3, "label") <- quote(phantom()%*%2*italic(T)/(italic(T)-italic(T)*Theta)^3)
attr(invTX, "label") <- quote(phantom())/italic(T*X))
# uncomment the following to make the plot
#EOSplot(d, var)

## model experimental volumes of CH4
## using HKF equation and an exploratory one
f <- system.file("extdata/cpetc/v.CH4.HMM96.csv", package="CHNOSZ")
d <- read.csv(f)
d$P <- convert(d$P, "bar")
# the HKF equation
varHKF <- c("invTTheta", "QBorn")
# alpha is the expansivity coefficient of water
varal <- c("invTTheta", "alpha")
par(mfrow=c(2,2))
# for both HKF and the expansivity equation
# we'll fit up to a temperature limit
EOSplot(d, varHKF, T.max=663, T.plot=625)
legend("bottomright", pch=1, legend="Hnedkovsky et al., 1996")
title("V of CH4(aq), HKF equation")
EOSplot(d, varal, T.max=663, T.plot=625)
title("V of CH4(aq), expansivity equation")
EOSplot(d, varHKF, T.max=663)
title("V of CH4(aq), HKF equation")
EOSplot(d, varal, T.max=663)
title("V of CH4(aq), expansivity equation")
# note that the volume regression using the HKF gives
# a result for omega (coefficient on Q) that is
# not consistent with the high-T heat capacities

eqdata  Read data from an EQ6 output file

Description

Extract computational results for aqueous species, solid phases, mineral saturation states, or speciation summaries at each step of reaction progress in an EQ6 output file. The results are written to a comma-separated value file that can be read by other programs.

Usage

eqdata(file, species, property = "log act", outfile = TRUE)
Arguments

file character, path to EQ6 output file
species character, name(s) of species or minerals
property character, property to get
outfile logical or character, file for saving results

Details

The first argument, file, is the name of the EQ6 (Wolery, 1992; Wolery and Daveler, 1992) output file. species indicates the aqueous species, solid phases, minerals, or basis species for which you want values; multiple names can be provided except for basis species, which can be a single value.

property indicates the property to retrieve. Specifying a value other than one listed below will cause an error.

- Aqueous species: ‘conc’, ‘log conc’, ‘log g’, or ‘log act’
- Solid phases: ‘log moles’, ‘moles’, ‘grams’, or ‘volume cc’
- Minerals (saturation states): ‘affinity, kcal’
- Basis species (speciation): ‘molal conc’ or ‘per cent’

The result of the function is a data frame (returned invisibly), with columns zi (reaction progress), T (temperature in °C), ah2o (activity of water) and one column for each of the requested species or, for speciation of basis species, one column for each unique species found in all of the speciation summary blocks for that basis species. Values are listed as NA (not available) for species or phases that are not present in the EQ6 output at any of the increments of reaction progress.

If outfile is TRUE, the result is saved in a file named like ‘file’ . ‘property’. csv, in the same directory as file. The name of the outfile can be provided to override this naming scheme, or this argument can be set to FALSE or NULL, to turn off writing the result to a file.

Thanks to Peter Canovas and Everett Shock for helping to test the code and offering ideas for improvements. The function has been tested with output files generated by EQ3/6 version 7.1 running on a Unix platform.

References


equilibrate

Examples

```r
## Not run:
# if an EQ6 output file named "rainbow2.6o" is in the current
# working directory, the following command will output values
# of log act (logarithm of activity) for the selected aqueous
# species to a file named rainbow2.6o.log act.csv
eqdata("rainbow2.6o",c("h+","so2","aq","h2","aq"),"log act")
## End(Not run)
```

equilibrate  
*Equilibrium Chemical Activities of Species*

Description

Calculate equilibrium chemical activities of species from the affinities of formation of the species at unit activity.

Usage

```r
equilibrate(aout, balance=NULL, loga.balance=NULL, ispecies=1:length(aout$values), normalize=FALSE, stay.normal=FALSE)
equil.boltzmann(Astar, n.balance, loga.balance)
equil.reaction(Astar, n.balance, loga.balance)
```

Arguments

- `aout` list, output from affinity
- `balance` character or numeric, how to balance the transformations
- `ispecies` numeric, which species to include
- `normalize` logical, normalize the molar formulas of species by the balancing coefficients?
- `stay.normal` logical, report results for the normalized formulas?
- `Astar` numeric, affinities of formation reactions excluding species contribution
- `n.balance` numeric, number of moles of conserved component in the formation reactions of the species of interest
- `loga.balance` numeric, logarithm of total activity of balanced quantity

Details

equilibrate provides an interface to calculate the chemical activities of species in metastable equilibrium, in an open system at constant temperature and pressure and chemical activities of basis species, and with linear balancing constraints on transformations.

It takes as input aout, the output from affinity, which may be calculated from a multidimensional grid of conditions. The equilibrium chemical activities of species are calculated using either the
equilibrate

equilibrate or equil.boltzmann functions, the latter only if the balance is on one mole of species.

As aout contains the chemical affinities of formation reactions of each species of interest, equilibrate in order to function needs to be provided constraints on how to balance the reactions representing transformations between the species. balance returns the balancing coefficients, where balance indicates the balancing constraints, according to the following scheme:

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'NULL'</td>
<td>autoselect using which.balance</td>
</tr>
<tr>
<td>name of basis species</td>
<td>balance on this basis species</td>
</tr>
<tr>
<td>'length'</td>
<td>balance on length of proteins</td>
</tr>
<tr>
<td>'1'</td>
<td>balance on one mole of species</td>
</tr>
<tr>
<td>numeric vector</td>
<td>user-defined constraints</td>
</tr>
</tbody>
</table>

The default value of NULL for balance indicates to select the first shared basis species in all formation reactions identified using which.balance, or if that fails, to set the balance to '1'. However, if all the species (as listed in code aout$species) are proteins (have an underscore character in their names), the default value of NULL for balance indicates to use 'length' as the balance.

NOTE: the summation of activities assumes an ideal system, so ‘molality’ is implied by ‘activity’ in the following. loga.balance gives the logarithm of the total activity of balance (which is total activity of species for ‘1’ or total activity of amino acid residue-equivalents for ‘length’). If loga.balance is missing, its value is taken from the activities of species listed in aout; this default is usually the desired operation.

normalize if TRUE indicates to normalize the molar formulas of species by the balance coefficients. This operation is intended for systems of polymers, such as proteins, whose conventional formulas are much larger than the basis species. The normalization also applies to the balancing coefficients, which as a result consist of ‘1’s. normalize has the same effect as did diagram(..., residue=TRUE) in versions of CHNOSZ before 0.9-9. After normalization and equilibration, the equilibrium activities are then un-normalized (for the original formulas of the species), unless stay.normal is TRUE.

ispecies can be supplied to identify a subset of the species to include in the calculation.

equil.boltzmann is used to calculation the equilibrium activities if balance is ‘1’ (including the normalized result when normalize is TRUE), otherwise equil.reaction is called.

Value

equil.reaction and equil.boltzman each return a list with dimensions and length equal to those of Astar, giving the log10 of the equilibrium activities of the species of interest. equilibrate returns a list, containing first the values in aout, to which are appended m.balance (the balancing coefficients if normalize is TRUE, a vector of ‘1’s otherwise), n.balance (the balancing coefficients if normalize is FALSE, a vector of ‘1’s otherwise) and loga.equil (the calculated equilibrium activities of the species). balance returns a list containing the balancing coefficients (n) and a textual description (description).

Algorithms

The input values to equil.reaction and equil.boltzmann are in a list, Astar, all elements of the list having the same dimensions; they can be vectors, matrices, or higher-dimensional arrays.
Each list element contains the chemical affinities of the formation reactions of one of the species of interest (in dimensionless base-10 units, i.e. \( \Delta G/2.303RT \)), calculated at unit activity of the species of interest. The equilibrium activities (in base-10 log units) of the species of interest returned by either function satisfy the constraints that 1) the final chemical affinities of the formation reactions of the species are all equal and 2) the total activity of the conserved component is equal to \((\log a_{balance})\). The first constraint does not impose a complete equilibrium, where the affinities of the formation reactions are all equal to zero, but allows for a metastable equilibrium, where the affinities of the formation reactions are equal to each other.

In `equilreaction` (the algorithm described in Dick, 2008 and the only one available prior to CHNOSZ-0.8), the calculations of relative abundances of species are based on solving a system of equations representing the two constraints stated above. A close approximation of the solution is found using `uniroot`. Prior to CHNOSZ_0.9-9, the values in the `Astar` were used to generate initial guesses of the logarithms of activities of species; values of `loga.balance` that were hugely different from these guesses could generate errors from `uniroot` such as “f() values at end points not of opposite sign”. Now (from version 0.9-9), a more flexible method for generating guesses is in place.

In `equilboltzmann` (algorithm available beginning with CHNOSZ-0.8), the chemical activities of species are calculated using the Boltzmann distribution. This calculation is faster than the algorithm of `equilreaction`, but is limited to systems where the transformations are all balanced on one mole of species.

**References**


**See Also**

- `diagram` has examples of using `equilibrate` to make equilibrium activity diagrams. `revisit` can be used to perform further analysis of the equilibrium activities. `palply` is used by both `equilreaction` and `equilboltzmann` to parallelize intensive parts of the calculations if `parallel` is loaded.

**Examples**

```r
## equilibrium in a simple system:
## ionization of carbonic acid
basis("CHNOS+")
species(c("CO2", "HCO3-", "CO3-2"))
# set unit activity of the species (\( \Theta = \log 10(1) \))
species(1:3, 0)
# calculate Astar (for unit activity)
res <- 100
Astar <- affinity(pH=c(0, 14, res))$values
# the logarithms of activity for a total activity
# of the balanced quantity (C or CO2) equal to 0.001
loga.boltz <- equil.boltzmann(Astar, c(1, 1, 1), 0.001)
# calculated another way
loga.react <- equil.reaction(Astar, c(1, 1, 1), 0.001)
```
# probably close enough for most purposes
stopifnot(all.equal(loga.boltz, loga.react))
# the first ionization constant (pKa)
ipKa <- which.min(abs(loga.boltz[[1]] - loga.boltz[[2]]))
pKa.equil <- seq(0, 14, length.out=res)[ipKa]
# calculate logK directly
logK <- subcr(c("CO2","H2O","HCO3-","H+"), c(-1, -1, 1, 1), T=25)$out$logK
# we could decrease tolerance here by increasing res
stopifnot(all.equal(pKa.equil, -logK, tol=1e-2))

examples  Run Examples from the Documentation

Description
Run the examples contained in each of the documentation topics.

Usage
examples(do.png = FALSE)
demos(which = c("sources", "NaCl", "cordierite",
  "phosphate", "nucleobase", "orp", "findit",
  "CO2Ac", "nonideal"))

Arguments
do.png logical, generate PNG files for the plots?
which character, which example to run.

Details
examples runs all the examples in the documentation for the package. example is called for each
topic with ask set to FALSE (so all of the figures are shown without prompting the user). If do.png
is TRUE, the plots in the examples are saved as png files having names beginning with the name of
each of the help topics.

demos is a function to run other examples that are provided as demos. demo is called with settings to
not echo the source code and to not ask before making each plot. The demo(s) to run is/are specified
by which; the default is to run them in the order of the list below. See the comments in the source
code for more information about each demo.

| sources     | cross-check the reference list with the thermodynamic database |
| NaCl        | equilibrium constant for aqueous NaCl dissociation (Shock et al., 1992) |
| cordierite  | equilibrium constant of hydrous cordierite dehydration |
| phosphate   | phosphate speciation with pH, temperature and ionic strength |
| nucleobase  | relative stabilities of nucleobases and some amino acids |
| orp         | oxidation-reduction potential of redox standards as a function of temperature |
| findit      | detailed example of usage of findit using log-normal distribution as an objective |
| CO2Ac       | activity of CO2 buffered by acetic acid; comparing affinity(return.buffer=TRUE) with diagram(what="CO2Ac") |
| nonideal    | activity coefficient of charged species (Alberty, 2003), using the IS argument of subcr
References


Examples

```r
demos(c("orp", "NaCl"))
```

## Not run:

# use the following to run examples in all help topics
demsc()

## End(Not run)

---

**extdata**

---

**Extra Data**

Description

The files in the subdirectories of extdata support the examples in the package documentation and vignettes.

Details

Files in abundance contain protein abundance data:

- **stress** is a data frame listing proteins identified in selected proteomic stress response experiments. The names of proteins begin at row 3, and columns are all the same length (padded as necessary at the bottom by NAs). Names correspond to ordered locus names (for ‘Sce’) or gene names (for ‘Eco’). The column names identify the experiments, the first row contains the name of the organism (‘Sce’ or ‘Eco’) and the third row has the reference key for the source of the data (listed in `thermo$refs`).

- **AA83.csv** has reference abundances for 71 proteins taken from Fig. 3 of Anderson and Anderson, 2002 (as corrected in Anderson and Anderson, 2003). The columns with data taken from these sources are type (hemoglobin, plasma, tissue, or interleukin), description (name used in the original figure), log10(pg/ml) (*upper limit* of abundance interval shown in Anderson and Anderson, 2003, log10 of concentration in pg/ml). The additional columns are data derived from a search of the SWISS-PROT/UniProtKB database based on the descriptions of the proteins: name (nominal UniProtKB name for this protein), name2 (other UniProtKB names(s) that could apply to the protein), and note (notes based on searching for a protein of this description). The amino acid compositions of all proteins whose names are not NA are included in `thermo$protein`. The `abbrv` column for the proteins contains the description.
given by Anderson and Anderson, 2003, followed by (in parentheses) the UniProtKB accession number. Annotated initiator methionines (e.g. for ferritin, myoglobin, ENOG), signal peptides or propeptides were removed from the proteins (except where they are not annotated in UniProtKB: IGHG1, IGHA1, IGHD, MBP). In cases where multiple isoforms are present in UniProtKB (e.g. Albumin) only the first isoform was taken. In the case of C4 Complement (CO4A) and C5 Complement (CO5), the amino acid composition of only the alpha chains are listed. In the case of the protein described as iC3b, the amino acid sequence is taken to be that of Complement C3c alpha’ chain fragment 1 from CO3, and is given the name CO3.C3c. The non-membrane (soluble) chains of TNF-binding protein (TNR1A) and TNF-alpha (TNFA) were used. Rantes, MIP-1 beta and MIP-1 alpha were taken from C-C motif chemokines (CCL5, CCL4, CCL3 respectively). C-peptide was taken from the corresponding annotation for insulin and here is named INS.C. See protein and readExpr for examples that use this file.

- ISR+08.csv has columns excerpted from Additional File 2 of Ishihama et al. (2008) for protein abundances in E. coli cytosol. The columns in this file are ID (Swiss-Prot ID), accession (Swiss-Prot accession), emPAI (exponentially modified protein abundance index), copynumber (emPAI-derived copy number/cell), GRAVY (Kyte-Doolittel), FunCat (FunCat class description), PSORT (PSORT localisation), ribosomal (yes/no). See readExpr for examples that use this file.

- yeastgfp.csv.xz Has 28 columns; the names of the first five are yorf, gene name, GFP tagged?, GFP visualized?, and abundance. The remaining columns correspond to the 23 subcellular localizations considered in the YeastGFP project (Huh et al., 2003 and Ghaemmaghami et al., 2003) and hold values of either T or F for each protein. ‘yeastgfp.csv’ was downloaded on 2007-02-01 from http://yeastgfp.ucsf.edu using the Advanced Search, setting options to download the entire dataset and to include localization table and abundance, sorted by orf number. See yeastgfp for examples that use this file.

Files in bison contain BLAST results and taxonomic information for a metagenome:

- bisonN_vs_refseq57.blast.xz, bisonS_vs_refseq57.blast.xz, bisonR_vs_refseq57.blast.xz, bisonQ_vs_refseq57.blast.xz, bisonP_vs_refseq57.blast.xz are partial tabular BLAST results for proteins in the Bison Pool Environmental Genome. Protein sequences predicted in the metagenome were downloaded from the Joint Genome Institute’s IMG/M system on 2009-05-13. The target database for the searches was constructed from microbial protein sequences in National Center for Biotechnology Information (NCBI) RefSeq database version 57, representing 7415 microbial genomes. The ‘blastall’ command was used with the default setting for E value cutoff (10.0) and options to make a tabular output file consisting of the top 20 hits for each query sequence. The function read.blast was used to extract only those hits with E values less than or equal to 1e-5 and with sequence similarity (percent identity) at least 30 percent, and to keep only the first hit for each query sequence. The function write.blast was used to save partial BLAST files (only selected columns). The files provided with CHNOSZ contain the first 5,000 hits for each sampling site at Bison Pool, representing between about 7 to 15 percent of the first BLAST hits after similarity and E value filtering.

- gi.taxid.txt.xz is a table that lists the sequence identifiers (gi numbers) that appear in the example BLAST files (see above), together with the corresponding taxon ids used in the NCBI databases. This file is not a subset of the complete ‘gi_taxid_prot.dmp.gz’ available at ftp://ftp.ncbi.nih.gov/pub/taxonomy/ but instead is a subset of ‘gi.taxid.txt’ generated from the RefSeq release catalog using ‘gencat.sh’ in the refseq directory. See id.blast for an example that uses this file and the BLAST files described above.
Files in cpetc contain heat capacity data and other thermodynamic properties:

- PM90.csv Heat capacities of four unfolded aqueous proteins taken from Privalov and Makhatadze, 1990. Names of proteins are in the first column, temperature in °C in the second, and heat capacities in J mol⁻¹ K⁻¹ in the third. See ionize.aa for an example that uses this file.
- RH95.csv Heat capacity data for iron taken from Robie and Hemingway, 1995. Temperature in Kelvin is in the first column, heat capacity in J K⁻¹ mol⁻¹ in the second. See subcrt for an example that uses this file.
- RT71.csv pH titration measurements for unfolded lysozyme (‘LYSC_CHICK’) taken from Roxby and Tanford, 1971. pH is in the first column, net charge in the second. See ionize.aa for an example that uses this file.
- SOJSH.csv Experimental equilibrium constants for the reaction NaCl(aq) = Na⁺ + Cl⁻ as a function of temperature and pressure taken from Fig. 1 of Shock et al., 1992. Data were extracted from the figure using g3data (http://www.frantz.fi/software/g3data.php). See water for an example that uses this file.
- Cp.CH4.Hw97.csv, V.CH4.HwM96.csv Apparent molar heat capacities and volumes of CH₄ in dilute aqueous solutions reported by Hnedkovsky and Wood, 1997 and Hnedkovsky et al., 1996. See EOSregress for examples that use these files.
- BM60_Fig7.dat Eh-pH values for normal, wet and waterlogged soils from Fig. 7 of Baas Becking et al., 1960. See the ‘anintro’ vignette for an example that uses this file.
- SC10_Rainbow.csv Values of temperature (°C), pH and logarithms of activity of CO₂, H₂, NH₄⁺, H₂S and CH₄ for mixing of seawater and hydrothermal fluid at Rainbow field (Mid-Atlantic Ridge), taken from Shock and Canovas, 2010.

Files in fasta contain protein sequences:

- HTCC1062.faa.xz is a FASTA file of 1354 protein sequences in the organism Pelagibacter ubique HTCC1062 downloaded from the NCBI RefSeq collection on 2009-04-12. The search term was Protein: txid335992[Organism:noexp] AND "refseq"[Filter]. See util.fasta and revisit for examples that use this file.
- EF-Tu.aln consists of aligned sequences (394 amino acids) of elongation factor Tu (EF-Tu). The sequences correspond to those taken from UniProtKB for ECOLI (Escherichia coli), THETH (Thermus thermophilus) and THEMA (Thermotoga maritima), and reconstructed ancestral sequences taken from Gaucher et al., 2003 (maximum likelihood bacterial stem and mesophilic bacterial stem, and alternative bacterial stem). See the ‘formation’ vignette for an example that uses this file.

Files in protein contain protein composition data for model organisms. See more.aa and read.expr for examples that use these files.

- Sce.csv.xz Data frame of amino acid composition of 6716 proteins from the Saccharomyces Genome Database (SGD). Values in the first three columns are the ORF names of proteins, SGDID, and GENE names. The remaining twenty columns (ALA..VAL) contain the numbers of the respective amino acids in each protein. The sources of data for ‘Sce.csv’ are the files ‘protein_properties.tab’ and ‘SGD_features.tab’ (for the gene names), downloaded from http://www.yeastgenome.org on 2013-08-24.

Files in refseq contain code and results of processing NCBI Reference Sequences (RefSeq) for microbial proteins, using RefSeq release 61 of 2013-09-09:

• README.txt Instructions for producing the data files.

• gencat.sh Bash script to extract microbial protein records from the RefSeq catalog.

• gi.taxid.txt Output from above. The complete file is too large to distribute with CHNOSZ, but a portion is included in extdata/bison to support processing example BLAST files for the Bison Pool metagenome (based on RefSeq 57, 2013-01-08).

• mkfaa.sh Combine the contents of .faa.gz files into a single FASTA file (to use e.g. for making a BLAST database).

• protein.refseq.R Calculate average amino acid composition of all proteins for each organism identified by a taxonomic ID.

• trim_refseq.R Keep only selected organism names (reduces number of taxa from 6758 to 779, helps to control package size).

• protein_refseq.csv.xz Output from above. See example in protein.info.

• taxid.names.R Generate a table of scientific names for the provided taxids. Requires the complete names.dmp and nodes.dmp from NCBI taxonomy files.


Files in taxonomy contain example taxonomic data files:

• names.dmp and nodes.dmp are excerpts of the taxonomy files available on the NCBI ftp site ([ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz](ftp://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz), accessed 2010-02-15). These example files contain only the entries for *Escherichia coli* K-12, *Saccharomyces cerevisiae*, *Homo sapiens*, *Pyrococcus furiosus* and *Methanocaldococcus jannaschii* (taxids 83333, 4932, 9606, 186497, 243232) and the higher-ranking nodes (genus, family, etc.) in the respective lineages. See taxonomy for examples that use this file.

Files in thermo contain additional thermodynamic data and group additivity definitions:

• OBIGT-2.csv contains supplementary thermodynamic data in the same format as the primary database in data/OBIGT.csv. Data for some entries in the primary database are taken from different literature sources in this file. The default action of `add.obigt` is to add the contents of this file to CHNOSZ’s working database in `thermo$obigt`. See diagram and the code of anim.TCA for examples that use this file.
• `obigt_check.csv` contains the results of running `check.obigt` to check the internal consistency of entries in the primary and supplementary databases.

• `groups_big.csv` Group contribution matrix: five structural groups on the columns ([-CH3], [-CH2], [-CH2OH], [-CO], [-COOH]) and 24 compounds on the rows (alkanes, alcohols, ketones, acids, multiply substituted compounds).

• `groups_small.csv` Group contribution matrix: twelve bond-specific groups on the columns, and 25 compounds on the rows (as above, plus isocitrate). Group identity and naming conventions adapted from Benson and Buss (1958) and Domalski and Hearing (1993). See the ‘xadditivity’ vignette for examples that use this file and `groups_big.csv`.

• `RH98_Table15.csv` Group stoichiometries for high molecular weight crystalline and liquid organic compounds taken from Table 15 of Richard and Helgeson, 1998. The first three columns have the compound name, formula and physical state (‘cr’ or ‘liq’). The remaining columns have the numbers of each group in the compound; the names of the groups (columns) correspond to species in `thermosp.obigt`. The compound named ‘5a(H),14a(H)-cholestan’ in the paper has been changed to ‘5a(H),14b(H)-cholestan’ here to match the group stoichiometry given in the table. See `RH2obigt` for a function that uses this file.

• `DLEN67.csv` Standard Gibbs energies of formation, in kcal/mol, from Dayhoff et al., 1967, for nitrogen (N2) plus 17 compounds shown in Fig. 2 of Dayhoff et al., 1964, at 300, 500, 700 and 1000 K.

References


SGD project. *Saccharomyces* Genome Database, [http://www.yeastgenome.org](http://www.yeastgenome.org)


YeastGFP project. Yeast GFP Fusion Localization Database, http://yeastgfp.ucsf.edu; Current location: http://yeastgfp.yeastgenome.org

**findit**  

---

### Gridded Search to Optimize Objective Functions

**Description**

Use a gridded search to find a combination of one or more of chemical activities of basis species, temperature and/or pressure that maximize or minimize a objective function of the metastable equilibrium chemical activities of the species of interest.

**Usage**

```r
findit (lims = list(), objective = "CV", niter = NULL, iprotein = NULL, plot.it = TRUE, T = 25, P = "Psat", res = NULL, labcex = 0.6,
loga2 = NULL, loga.balance = 0, rat = NULL,
balance = NULL, normalize = FALSE)
```

```r
## S3 method for class 'findit'
plot(x, which=NULL, mar=c(3.5,5,2,2), xlab="iteration", ...)
```

**Arguments**

- `lims`: list, specification of search limits
- `objective`: character, name of objective function to optimize
- `niter`: numeric, number of iterations
- `res`: numeric, grid resolution (number of points on one edge)
- `iprotein`: numeric, indices of proteins
- `plot.it`: logical, make a plot?
- `T`: numeric, temperature
- `P`: numeric, pressure; or character, "Psat"
- `labcex`: numeric, character expansion for plot labels
- `loga2`: numeric, reference logarithms of activity of species
- `loga.balance`: numeric, logarithm of total activity of balanced quantity (passed to `diagram`)
- `rat`: numeric, ratio of edge length in successive iterations
- `balance`: character or numeric, balanced quantity (passed to `diagram`)
- `normalize`: logical, normalize chemical formulas by the balance vector? (passed to `diagram`)
- `x`: list, object of class `findit`
- `which`: numeric, which of the parameters to plot
- `mar`: numeric, plot margin specification
- `xlab`: character, x-axis label
- `...`: additional arguments passed to `plot`
Details

findit implements a gridded optimization to find the minimum or maximum value of an \textit{objective} function. The variables are one or more of the chemical activities, temperature and/or pressure whose ranges are listed in \textit{lims}. Generally, the system (\textit{basis} species and \textit{species} of interest) must be set up before calling this function. If \textit{iprotein} is supplied, indicating a set of proteins to use in the calculation, the definition of the \textit{species} is not required. \textit{lims} is a list, each element of which is vector having a name that is the formula of one of the basis species, \lq T\rq{} or \lq{}P\rq{} and a pair of values indicating the range of the named parameter. The values are the logarithms of activities of the basis species, or temperature or pressure (in the user\rq{}s units; see \textit{util.units}). If either \lq{}T\rq{} or \lq{}P\rq{} is missing from the list in \textit{lims}, the calculations are performed at isothermal and/or isobaric conditions indicated by \textit{T} and \textit{P} arguments.

Taking \textit{nd} as the number of dimensions (number of variables in \textit{lims}), default values of \textit{niter} and \textit{res} come from the following table. These settings have been selected to be able to run the function quickly in the higher dimensions. Detailed studies of a system might have to use more iterations and/or higher resolutions.

<table>
<thead>
<tr>
<th>\textit{nd}</th>
<th>\textit{niter}</th>
<th>\textit{res}</th>
<th>\text{grid points (res}\textsuperscript{nd}\text{)}</th>
<th>\textit{rat}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>128</td>
<td>128</td>
<td>0.7</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>64</td>
<td>4096</td>
<td>0.7</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>16</td>
<td>4096</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>4096</td>
<td>0.9</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>6</td>
<td>7776</td>
<td>0.9</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>4</td>
<td>4096</td>
<td>0.95</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>4</td>
<td>16384</td>
<td>0.95</td>
</tr>
</tbody>
</table>

The function performs \textit{niter} iterations. At first, the limits of the parameters given in \textit{lims} define the extent of a \textit{nd}-dimensional box around the space of interest. The value of objective is calculated at each of the \textit{res}\textsuperscript{nd} grid points and and optimum value located (see \textit{revisit} and \textit{optimal.index}). In the next iteration the new search box is centered on the location of the optimum value, and the edges are shrunk so their length is \textit{rat} \times{} the length in the previous step. If the limits of any of the parameters extend beyond those in \textit{lims}, they are pushed in to fit (preserving the difference between them).

plot.findit plots the values of the parameters and the objective function as a function of the number of iterations.

Value

\textit{findit} returns a list having class \textit{findit} with elements \textit{value} (values of the parameters, and value of the objective function, at each iteration), \textit{lolim} (lower limits of the parameters) and \textit{hilim} (upper limits of the parameters).

See Also

demos("findit") and tests/test-findit.R for more examples.
Examples

```r
## an inorganic example: sulfur species
basis("CHNOS+")
basis("pH", 5)
species(c("H2S", "S2-2", "S3-2", "S2O3-2", "S2O4-2", "S3O6-2", "S4O6-2", "S2O6-2", "HSO3-", "SO2", "HSO4-"))
# to minimize the standard deviations of the
# logarithms of activity the species
objective <- "SD"
# the variables we are interested in
vars <- list(O2=c(-50, -15), pH=c(0, 14), T=c(275, 375))
# optimize logfO2 at constant T and pH
f1 <- findit(vars[1], objective, T=325, P=350, niter=3)
title("S.D. of equilibrium log activities of sulfur species")
# optimize logfO2 and pH at constant T
f2 <- findit(vars[1:2], objective, T=325, P=350, res=16, niter=5)
title("S.D. of equilibrium log activities of sulfur species")
# optimize logfO2, pH and T (at constant P ...)
f3 <- findit(vars, objective, P=350, res=10, niter=10)
title("S.D. of equilibrium log activities of sulfur species")
# the results
print(f1.out <- sapply(f1$value, tail, 1))
print(f2.out <- sapply(f2$value, tail, 1))
print(f3.out <- sapply(f3$value, tail, 1))
# with more variables, we should find a greater degree of optimization
stopifnot(f2.out["SD"] < f1.out["SD"])
stopifnot(f3.out["SD"] < f2.out["SD"])
```

IAPWS95

Properties of Water from IAPWS-95

Description

Calculate thermodynamic properties of water following the IAPWS-95 formulation.

Usage

```r
IAPWS95(property, T = 298.15, rho = 1000)
IAPWS95.idealgas(p, delta, tau)
IAPWS95.residual(p, delta, tau)
WP02.auxiliary(property, T = 298.15)
```

Arguments

- `property` character, name(s) of property(s) to calculate
- `T` numeric, temperature (K)
rho numeric, density (kg m$^{-3}$)

p character, name of property (Helmholtz free energy or its derivatives)

delta numeric, density divided by critical density

tau numeric, critical temperature divided by temperature

Details

IAPWS95 provides an implementation of the IAPWS-95 formulation for properties (including pressure) calculated as a function of temperature and density.

The IAPWS95 function returns values of thermodynamic properties in specific units (per gram). The IAPWS-95 formulation follows the triple point convention used in engineering (values of internal energy and entropy are taken to be zero at the triple point).

Auxiliary equations to the IAPWS-95 formulation (Wagner and Pruss, 2002) are provided in WP02.auxiliary; the property for this function can be one of ‘P.sigma’ (saturation vapor pressure in MPa), ‘dP.sigma.dt’ (derivative of saturation vapor pressure with respect to temperature), or ‘rho.liquid’ or ‘rho.vapor’ (density of liquid or vapor in kg m$^{-3}$).

IAPWS95.idealgas and IAPWS95.residual are supporting functions to IAPWS95 for calculating the ideal-gas and residual parts in the IAPWS-95 formulation. The value of p can be one of ‘phi’, ‘phi.delta’, ‘phi.delta.delta’, ‘phi.tau’, ‘phi.tau.tau’, or ‘phi.delta.tau’, to calculate the specific dimensionless Helmholtz free energy (‘phi’) or one of its respective derivatives.

For IAPWS95 the upper temperature limit of validity is 1000 °C, but extrapolation to much higher temperatures is possible (Wagner and Pruss, 2002). Valid pressures are from the greater of zero bar or the melting pressure at temperature to 10000 bar (with the provision for extrapolation to more extreme conditions). The present functions do not check these limits and will attempt calculations for any range of input parameters, but may return NA for properties that fail to be calculated at given temperatures and pressures and/or produce warnings or even errors when problems are encountered.

Value

A data frame the number of rows of which corresponds to the number of input temperature, pressure and/or density values.

References


See Also

water.IAPWS95 for a wrapper that converts the specific units to molar quantities and is a function of pressure instead of density.
Examples

```r
# calculate pressure for given temperature, density
P <- as.numeric(IAPWS95("P", T=500, rho=838.0235))

# density along saturation curve
T <- seq(273.15, 623.15, 25)
WP02.auxiliary(T=T) # liquid from WP02
WP02.auxiliary("rho.vapor", T) # steam from WP02
```

Description

Search for species by name or formula, retrieve their thermodynamic properties and parameters, and add proteins to the thermodynamic database.

Usage

```r
info(species = NULL, state = NULL, check.it=TRUE)
info.character(species, state = NULL, check.protein=TRUE)
info.numeric(ispecies, check.it=TRUE)
info.approx(species, state = NULL)
info.text(ispecies)
```

Arguments

- `species`: character, names or formulas of species, or (for `info` only) numeric with same meaning as `ispecies`
- `state`: character, physical states of the species
- `check.it`: logical, check GHS and EOS parameters for self-consistency?
- `check.protein`: logical, check if a matching protein can be found?
- `ispecies`: numeric, index of species in the thermodynamic database

Details

`info` is the primary function used for querying the thermodynamic database (`thermo$obigt`). In common usage, it is called recursively; first with a character value (or values) for `species` indicating the name(s) or formula(s) of the species of interest. The result of this call is a numeric value, which can be provided as an argument in a second call to `info` in order to retrieve a data frame of the thermodynamic properties of the species. For its work, `info` calls on the other functions that are described below, which unlike `info` all expect arguments with length=1.

`info.character` searches for matches of the indicated species to names, chemical formulas, and abbreviations (in the 'abbrv' column) in the thermodynamic database. If the text of the species is matched the index of that species is returned. If there are multiple matches for the `species` and `state` is NULL, the index of first match is returned. The order of entries in `thermo$obigt` is
grouped by states in the order ‘aq’, ‘cr’, ‘gas’, ‘liq’, so for species in both aqueous and gaseous states the index of the aqueous species is returned, unless state is set to ‘gas’. The two exceptions are species identified by ‘02’ or ‘oxygen’ (which without any indicated state matches the gaseous species) and ‘H2O’ (which matches the liquid species even if the indicated state is ‘aq’). Normally, if a species match can not be located, the function then looks for proteins with the name of species (using iprotein), computes its properties if found (ip2aa) and adds this to the thermodynamic database (mod.obigt). check.protein prevents the processing of proteins and is provided to avoid an infinite loop in the interaction with mod.obigt.

info.character has additional logic for dealing with proteins and with multiple matches for the ‘cr’ state. If the state is ‘cr’, matches will be counted for states entered as ‘cr1’, ‘cr2’ etc in the database, and all of the species indices will be returned. Note, however, that info only ever returns a single species index, which becomes NA in the case of multiple matches to ‘cr’. This functionality of info.character is used in subcrt to handle minerals with phase transitions.

Names of species including an underscore character are indicative of proteins, e.g. ‘LYSC_CHICK’. If the name of a protein is provided to info.character and the composition of the protein can be found using protein, the thermodynamic properties and parameters of the nonionized protein (calculated using amino acid group additivity) are added to the thermodynamic database. Included in the return value, as for other species, is the index of the protein in the thermodynamic database or NA if the protein is not found. Names of proteins and other species can be mixed.

info.approx searches the database for similar names or formulas using agrep. If one or more of these is found, the results are summarized on the screen, and the indices of the approximately matching species are returned. Species that have no approximate matches are indicated by NA in the return value. When invoked by info, the latter function accepts the species index only for a single approximate match; multiple matches are translated to NA.

info.numeric returns the rows of thermo$obigt indicated by ispeices, after removing any order-of-magnitude scaling factors. If these species are all aqueous or are all not aqueous, the compounded column names used in thermo$obigt are replaced with names appropriate for the corresponding equations of state. A missing value of one of the standard molal Gibbs energy (G) or enthalpy (H) of formation from the elements or entropy (S) is calculated from the other two, if available. If check.it is TRUE, several checks of self consistency among the thermodynamic properties and parameters are performed using checkGHS and checkEOS (this depends on the completeness of the data entry).

See Also

thermo for the thermodynamic database (specifically, thermo$obigt). check.obigt for checking self-consistency of individual entries in the database. protein for gathering compositions and thermodynamic properties of proteins.

Examples

## summary of available data
info()
## Not run:
## run a consistency check on each species in the database
# (marked dontrun because it takes a while)
info(check=TRUE)
ionize.aa

properties of Ionization of Proteins

Description

Calculate the charges of proteins and contributions of ionization to the thermodynamic properties of proteins.

Usage

```r
ionize.aa(aa, property = "Z", T = 25, P = "Psat", pH = 7,
           ret.val = NULL, suppress.Cys = FALSE)
```

Arguments

- `aa` data frame, amino acid composition in the format of `thermo$protein`
- `property` character, property to calculate
Details

The properties of ionization of proteins calculated by this function take account of the standard molal thermodynamic properties of ionizable amino acid sidechain groups and the terminal groups in proteins ([AABB]) and their equations of state parameters taken from Dick et al., 2006. The values of the ionization constants (pK) are calculated as a function of temperature, and the charges and the ionization contributions of other thermodynamic properties to the proteins are calculated additively, without consideration of electrostatic interactions, so they are best applied to the unfolded protein reference state.

For each amino acid composition in aa, the additive value of the property is calculated as a function of T, P and pH. Property can be NULL to denote net charge, or if not NULL is one of the properties available in subcrt, or is \( \lambda \) to calculate the dimensionless chemical affinity (\( A/2.303RT \)) of the ionization reaction for the protein. If ret.val is one of ‘pK’, ‘alpha’, or ‘aavals’ it indicates to return the value of the ionization constant, degree of formation, or the values of the property for each ionizable group rather than taking their sums for the amino acid compositions in aa.

Value

The function returns a matrix (possibly with only one row or column) with number of rows corresponding to the longest of T, P or pH (values of any of these with shorter length are recycled) and a column for each of the amino acid compositions in aa.

References


See Also

The amino acid composition in aa can be generated using e.g. ip2aa. *protein.info* and *protein.basis* use this function to compute the properties of ionized proteins.

This function is called by *A.ionization* as part of a calculation of *affinity* if proteins are among the species of interest, ‘H+’ is in the basis, and thermo$opt$ionize is TRUE.
Examples

```r
## Charge of LYSC_CHICK as a function of pH and T
# After Fig. 10 of Dick et al., 2006
# the rownumber of the protein in thermo$protein
ip <- iprotein("LYSC_CHICK")
# its amino acid composition
aa <- ip2aa(ip)
# additive charges of unfolded protein at 25, 100, 150 degrees C
# as a function of pH
ph <- seq(0, 14, 0.1)
Z.25 <- ionize.aa(aa, T=25, pH=pH)
plot(ph, Z.25[, 1], type="l", xlab="pH", ylab="net charge (Z)"
lines(ph, ionize.aa(aa, T=100, pH=pH)[, 1], col="red"
lines(ph, ionize.aa(aa, T=150, pH=pH)[, 1], col="orange"
text(c(12, 10, 9), c(-15, -16, -18),
  labels=paste("T", c(25, 100, 150), sep=""))
# suppress ionization of cysteine as if it was oxidized
# (to form non-ionizable cystine disulfide bonds)
lines(ph, ionize.aa(aa, T=25, pH=pH, suppress.Cys=TRUE)[, 1], lty=2)
text(12, -7, "T=25, oxidized")
# add experimental points
RT71 <- read.csv(system.file("extdata/cpet/RT71.csv", package="CHNOSZ"))
points(RT71$pH, RT71$Z)
legend("topleft", pch=1, legend="Roxy and Tanford, 1971")
title(main=paste("Ionization of unfolded LYSC_CHICK\n",
  "After Dick et al., 2006"))

## Heat capacity of LYSC_CHICK as a function of T
pH <- c(5, 9, 3)
T <- seq(0, 100)
# Cp of non-ionized protein
Cp.nonion <- scbct("LYSC_CHICK", T=T)$out[[1]]$Cp
plot(T, Cp.nonion, xlab=axis.label("T"), type="l",
  ylab=axis.label("Cp"), ylim=c(5000, 8000))
# Cp of ionization and ionized protein
aa <- ip2aa("LYSC_CHICK")
for(ph in c(5, 9, 3)) {
  Cp.ionized <- Cp.nonion + ionize.aa(aa, "Cp", T=T, pH=pH)[, 1]
  lines(T, Cp.ionized, lty=2)
  text(80, Cp.ionized[70], paste("pH =","pH")
}
# Makhatadze and Privalov's group contributions
T <- c(5, 25, 50, 75, 100, 125)
points(T, convert(MP90.cp("LYSC_CHICK", T), "cal"))
# Privalov and Makhatadze's experimental values
e <- read.csv(system.file("extdata/cpet/MP90.csv", package="CHNOSZ"))
e <- e[e$protein=="LYSC_CHICK",]
points(e$T, convert(e$Cp, "cal"), pch=16)
legend("bottomright", pch=c(16, 1, NA, NA), lty=c(NA, NA, 1, 2),
  legend=c("MP90 experiment", "MP90 groups",
  "DLH06 groups no ion", "DLH06 groups ionized"))
```
title("Heat capacity of unfolded LYSC_CHICK")

## Contour plots of net charge and ionization properties of LYSC_CHICK

aa <- ip2aa("LYSC_CHICK")
pH <- seq(0, 14, 0.2)
T <- seq(0, 200, 2)
val <- expand.grid(pH=pH, T=T)
par(mfrow=c(2, 2))
for(X in c("Z", "A", "Cp", "V")) {
  Y <- ionize.aa(aa, property=X, pH=val$pH, T=val$t)
  contour(pH, T, matrix(Y[, 1], ncol=length(T)),
         xlab="pH", ylab=axis.label("T"))
  title(main=axis.label(X))
}
par(mfrow=c(1, 1))
pu <- par("usr")
text(mean(pu[1:2]), sum(pu[3:4])*0.45,
     "additive properties of ionization of LYSC_CHICK")

### iprotein

#### Amino Acid Compositions of Proteins

**Description**

Functions to identify proteins, get and set amino acid compositions, and calculate thermodynamic properties from group additivity.

**Usage**

```r
iprotein(protein, organism=NULL)
ip2aa(protein, organism=NULL, residue=FALSE)
aa2eos(aa, state=get("thermo")$opt$state)
seq2aa(protein, sequence)
aasum(aa, abundance = 1, average = FALSE, protein = NULL, organism = NULL)
read.aa(file = "protein.csv")
add.protein(aa)
```

**Arguments**

- **protein**: character, name of protein; numeric, indices of proteins (rownnumbers of `thermo$protein`)
- **organism**: character, name of organism
- **residue**: logical, compute per-residue counts?
- **aa**: data frame, amino acid composition in the format of `thermo$protein`
- **state**: character, physical state
iprotein returns the row(s) of \texttt{thermoDprotein} that match the protein names. The names can be supplied in the single \texttt{protein} argument or as separated \texttt{protein} and \texttt{organism} (without the underscore). Any protein not matched returns an NA and generates a message.

\texttt{ip2aa} returns the row(s) of \texttt{thermoDprotein} that match the supplied protein names, OR the protein indices found by \texttt{iprotin}. Set \texttt{residue} to TRUE to return the per-residue composition (i.e. amino acid composition of the protein divided by total number of residues). For this function only, if the \texttt{protein} argument is a data frame, it is returned unchanged, except for possibly the per-residue calculation.

\texttt{aa2eos} calculates the thermodynamic properties and equations-of-state parameters for the completely nonionized proteins using group additivity with parameters taken from Dick et al., 2006 (aqueous proteins) and LaRowe and Dick, 2012 (crystalline proteins and revised aqueous methionine sidechain group). The return value is a data frame in the same format as \texttt{thermoDobigt}. \texttt{state} indicates the physical state for the parameters used in the calculation (‘aq’ or ‘cr’).

The remaining functions are more likely to be called directly by the user:

\texttt{seq2aa} returns a data frame of amino acid composition, in the format of \texttt{thermoDprotein}, corresponding to the provided sequence. Here, the \texttt{protein} argument indicates the name of the protein with an underscore (e.g. ‘LYSC\_CHICK’).

\texttt{aasum} returns a data frame representing the sum of amino acid compositions in the rows of the input \texttt{aa} data frame. The amino acid compositions are multiplied by the indicated abundance; that argument is recycled to match the number of rows of \texttt{aa}. If \texttt{average} is TRUE the final sum is divided by the number of input compositions. The name used in the output is taken from the first row of \texttt{aa} or from \texttt{protein} and \texttt{organism} if they are specified. This function is useful for calculating bulk amino acid compositions in stress response experiments or localization studies; see \texttt{readNexp} for examples of its use.

\texttt{readNaa} returns a data frame of amino acid composition based on the contents of the indicated file, which should be a CSV file with the same column names as \texttt{thermoDprotein}.

\texttt{addNprotein} completes the loop; any amino acid composition returned by the *\texttt{aa} functions described above can be added to \texttt{thermoDprotein} using this function to be made available to other
functions in the package. The amino acid compositions of proteins in aa with the same name as one in \texttt{thermoDprotein} are replaced. The value returned by this function is the rownumbers of \texttt{thermoDprotein} that are added and/or replaced.

References


See Also

\texttt{read.fasta} and \texttt{uniprot.aa} for getting amino acid compositions from a FASTA file or downloading them from UniProt.

\texttt{protein.info} for higher-level functions (chemical formulas, summaries of reaction coefficients and energies). \texttt{more.aa} for getting amino acid compositions for model organisms from additional data files in the \texttt{extdata/protein} directory, and \texttt{read.expr} for working with protein abundance and subcellular localization data.

Examples of stability calculations for proteins are in \texttt{protein}.

Examples

```r
# search by name in \texttt{thermoDprotein}
ip1 <- iprotein("LYSC\_CHICK")
ip2 <- iprotein("LYSC", "CHICK")
# these are the same
stopifnot(all.equal(ip1, ip2))
# two organisms with the same protein name
ip3 <- iprotein("MYG", c("HORSE", "PHYCA"))
# their amino acid compositions
ip2aa(ip3)
# their thermodynamic properties by group additivity
aa2eos(ip2aa(ip3))

# an example of an unrecognized protein name
ip4 <- iprotein("MYG\_PHYCA")
stopifnot(is.na(ip4))

# manually adding a new protein
# Human Gastric juice peptide 1
aa <- seq2aa("GAJU\_HUMAN", "LAAGKVEDSD")
ip <- add.protein(aa)
stopifnot(protein.length(ip)==10)
# the chemical formula of this peptide
stopifnot(as.chemical.formula(protein.formula(ip))="C41H69N11O18")
# we can also calculate a formula without using add.protein
as.chemical.formula(protein.formula(seq2aa("pentapeptide\_test", "ANL5G")));

# read a fasta file, calculate H/C and O/C for all proteins
```
# and averages by polypeptide chain, residue and mass
file <- system.file("extdata/fasta/HTCC1062.fasta.xz", package="CHNOSZ")
aa <- read.fasta(file)

pf <- as.data.frame(protein.formula(aa))

plot(pf$pf$C, pf$O/pf$C, pch=NA)

points(pf$pf$C, pf$O/pf$C, pch=20, cex=0.5, col="grey")

# average composition per polypeptide chain
chainaa <- aasum(aa, average=TRUE)

chainpf <- as.data.frame(protein.formula(chainaa))

points(chainpf$pf$C, chainpf$O/chainpf$C, pch=15, col="red")

# average by amino acid residue
pl <- protein.length(aa)

resaa <- aasum(aa, abundance=pl, average=TRUE)

respf <- as.data.frame(protein.formula(resaa))

points(respf$pf$C, respf$O/respf$C, pch=16, col="red")

# average by mass
pm <- mass(pf)

massaa <- aasum(aa, abundance=pm, average=TRUE)

masspf <- as.data.frame(protein.formula(massaa))

points(masspf$pf$C, masspf$O/masspf$C, pch=17, col="red")

# add a legend and title
legend("topright", pch=c(20, 15, 16, 17), col=c("grey", rep("red", 3)),

legend="protein", "chain average", "residue average", "mass average")
title(main=paste("O/C vs H/C for HTCC1062 proteins\n",

"n =", nrow(aa)))

---

## Parse Chemical Formulas

**Description**

Count the charge and number of elements in a chemical formula.

**Usage**

```r
makeup(formula, multiplier = 1, sum = FALSE, count.zero = FALSE)
count.charge(formula)
count.formulas(formula)
count.elements(formula)
```

**Arguments**

- `formula` character, a chemical formula
- `multiplier` numeric, multiplier for the elemental counts in each formula
- `sum` logical, add together the elemental counts in all formulas?
- `count.zero` logical, include zero counts for elements?
Details

`makeup` parses a chemical formula expressed in string notation, returning the numbers of each element in the formula. The formula may carry a charge, indicated by a + or - sign, possibly followed by a magnitude, after the uncharged part of the formula. The formula may have multiple subformulas enclosed in parentheses (but the parentheses may not be nested), each one optionally followed by a numeric coefficient. The formula may have one suffixed subformula, separated by `*` or `:`, optionally preceded by a numeric coefficient. All numbers may contain a decimal point.

`makeup` calls a sequence of supporting functions depending on specific characters present in the formula. If the formula has a charge, it is first parsed using `count.charge`. If the formula has subformulas, in parentheses or as a suffix, they are separated and counted using `count.formulas`. Finally, the elements in each subformula are counted using `count.elements`.

`count.elements` processes a simple chemical formula that must adhere to the following pattern: it starts with an elemental symbol; all elemental symbols start with an uppercase letter, and are followed by another elemental symbol, a number (possibly fractional, possibly signed), or nothing (the end of the formula).

Any sequence of one uppercase letter followed by zero or more lowercase letters is recognized as an elemental symbol by `count.elements`, but `makeup` will issue a warning for elemental symbols that are not present in `thermos$element`.

`makeup` can handle numeric and length > 1 values for the `formula` argument. If the argument is numeric, it identifies row number(s) in `thermos$obigt` from which to take the formulas of species. If `formula` has length > 1, the function returns a list containing the elemental counts in each of the formulas. If `count.zero` is TRUE, the elemental counts for each formula include zeros to indicate elements that are only present in any of the other formulas.

The `multiplier` argument must have either length = 1 or length equal to the number of formulas. The elemental count in each formula is multiplied by the respective value. If `sum` is true, the elemental counts in all formulas (after any multiplying) are summed together to yield a single bulk formula.

Value

`count.charge` returns a list with named elements `charge` and `uncharged`, indicating, respectively, the numeric value of the charge, and the original formula string excluding the charge. `count.formulas` returns a numeric vector with names refering to each of the subformulas or the whole formula if there are no subformulas. `count.elements` and `makeup` return numeric vectors with names refering to each of the elemental symbols in the formula. For `makeup`, if more than one formula is provided, a list of numeric vectors is returned, unless `sum` is TRUE.

See Also

Many other functions in CHNOSZ rely on `makeup` for their operation: `mass` and `entropy` for calculating properties of chemical compounds from their elements; `basis` and `i2A` for constructing stoichiometric matrices (with `count.zero`=TRUE); `subcrt` for checking mass balance of chemical reactions; and others.

Examples
more.aa

Proteins from Model Organisms

Description

Retrieve the amino acid compositions of one or more proteins from *Escherichia coli* or *Saccharomyces cerevisiae*.

Usage

more.aa(protein = NULL, organism)

Arguments

protein character, name of protein
organism character, name of organism (‘Eco’ or ‘Sce’)

---

```r
# the composition of a simple compound
makeup("CO2")  # 1 carbon, 2 oxygen

# the formula of lawsonite, with a parenthetical part and a suffix
makeup("CaAl2Si2O7(6H)2+H2O")

# fractional coefficients are ok
redfield <- c(106, 16, 1)
reddiv10 <- makeup("C10.6N1.6P0.1")
stopifnot(10*reddiv10 == redfield)

# the coefficient for charge is a number with a *preceding* sign
# e.g., ferric iron, with a charge of +3 is expressed as
makeup("Fe+3")

# transcribing the formula the way it appears in many
# publications produces a likely unintended result:
# 3 iron atoms and a charge of +1
makeup("Fe3+")

# these all represent a single negative charge, i.e., electron
makeup("-1")
makeup("Z0-1")
makeup("Z-1+0")

# hypothetical compounds with negative numbers of elements
makeup("C-4(O-2)")  # -4 carbon, -2 oxygen
makeup("C-4O-2")  # -4 carbon, 1 oxygen, -2 charge
makeup("C-4O-2-2")  # -4 carbon, -2 oxygen, -2 charge

# the 'sum' argument can be used to check mass and charge
# balance in a chemical reaction
formula <- c("H2O","H+","Z0-1","O2")
(mf <- makeup(formula, c(-1, 2, 2, 0.5), sum=TRUE))
stopifnot(all(mf==0))
```
more_.aa retrieves the amino acid composition(s) of the indicated proteins in either *Escherichia coli* or *Saccharomyces cerevisiae*. The value of organism can be one of ‘Eco’ or ‘Sce’. The calculation depends on the data files extdata/protein/Eco.csv.xz and extdata/protein/Sce.csv.xz, which contain the amino acid compositions of the proteins. The protein argument should be a vector or a list of vectors of one or more Ordered Locus Names (OLN) or Open Reading Frame (ORF) names that are found in these files. The output data frame contains rows with NA compositions for names that are not matched.

**Value**

A data frame, or list of data frames, containing the amino acid composition(s) of the specified protein(s) in the format of thermo$protein.

**See Also**

extdata describes the sources of compositional data for the proteins. Other examples of usage of more_.aa are shown for read.expr.

**Examples**

```r
# the first 13 names in UniProt for "aminotransferase ecoli"
at.ecoli <- c("BIOA", "ARNB", "SERC", "AAT", "TYRB", "ARGD", "ILVE", "ALAA", "ALAC", "YBDL", "AVTA", "GLMS", "PUUE")
# get the amino acid compositions
# note that ALAA and ALAC are not matched
at.aa <- more.aa(at.ecoli, "Eco")
# what are their formulas and oxidation states
protein.formula(at.aa)
ZC(protein.formula(at.aa))
```

---

**Objective Functions**

**Description**

Calculate statistical and thermodynamic quantities for activities of species. These functions can be specified as objectives in revisit and findit in order to identify optimal chemical conditions.

**Usage**

- `SD(a1)`
- `CV(a1)`
- `shannon(a1)`
- `DGmix(loga1)`
- `qqr(loga1)`
- `logact(loga1, loga2)`
spearman(loga1, loga2)
pearson(loga1, loga2)
RMSD(loga1, loga2)
CVRMSD(loga1, loga2)
DDGmix(loga1, loga2)
DGinf(a1, a2)
DGtr(loga1, loga2, Astar)
get.objfun(objective)

Arguments

- **aQ**: numeric matrix, chemical activities of species
- **logaQ**: numeric matrix, logarithms of activity
- **logaR**: numeric, reference values of logarithms of activity
- **aR**: numeric, reference values of activity
- **astar**: numeric, reference values of chemical affinity
- **objective**: character, name of objective function

Details

The value in **aQ** or **logaQ** is a matrix of chemical activities or logarithms of activity with a column for each species, and a row for each chemical condition. Except for calculations of the Shannon entropy, all logarithmic bases (including in the equations below) are decimal. **SD**, **CV** and **shannon** calculate the standard deviation, coefficient of variation, and Shannon entropy for the values in each row of **aQ**. The Shannon entropy is calculated from the fractional abundances: 

\[ H = \sum (-p \cdot \log_2(p)) \] 

where \( p_i = \frac{a_i}{\sum a_i} \).

**DGmix** calculates the Gibbs energy/2.303RT of ideal mixing from pure components corresponding to one molal (unit activity) solutions: 

\[ \text{DGmix}/2.303RT = \sum (a_1 \cdot \log a_1) \] 

(cf. Eq. 7.20 of Anderson, 2005).

**qqr** calculates the correlation coefficient on a quantile-quantile (Q-Q) plot (see **qqnorm** for each row of **logaQ**, giving some indication of the resemblance of the chemical activities to a log-normal distribution.

**logact** returns the logarithm of activity of a single species identified by index in **logaR** (which of the species in the system).

**spearman**, **pearson**, **RMSD** and **CVRMSD** calculate Spearman’s rank correlation coefficient, the Pearson correlation coefficient, the root mean squared deviation (RMSD) and the coefficient of variation of the RMSD between each row of **loga1** and the values in **loga2**. The **CVRMSD** is computed as the RMSD divided by the mean of the values in **loga1**.

**DDGmix** calculates the difference in Gibbs energy/2.303RT of ideal mixing between the assemblages with logarithms of activity **loga1** and **loga2**.

**DGinf** calculates the difference in Gibbs energy/2.303RT attributed to relative informatic entropy between an initial assemblage with activities **a2** and final assemblage(s) with activities with activities in each row of **a1**. The equation used is 

\[ \text{DGinf}/2.303RT = \sum (p_2 \cdot (\log p_2 - \log p_1)) \] 

where \( p_1 \) and \( p_2 \) are the proportions, i.e. \( p_1 = a_1 / \sum (a_1) \) and \( p_2 = a_2 / \sum (a_2) \). This equation has the
form of the Kullback-Leibler divergence, sometimes known as relative entropy (Ludovisi and Tat-
icchi, 2006). In specific cases (systems where formulas of species are normalized by the balancing
coefficients), the values of $\Delta g_{\text{inf}}$ and $\Delta g_{\text{tr}}$ are equal.

$\Delta g_{\text{tr}}$ calculates the change in Gibbs energy/2.303RT of a system in which species with initial log-
arithms of activity ($\log a Q$) are transformed to the same species with different final logarithms of
activity ($\log a R$) at constant temperature, pressure and chemical potentials of basis species. It is cal-
culated as the sum over species of $(G_2 - G_1)$ where $G_1/RT = -a_1^* A_{\text{star}} + a_1^* \log a_1 - a_1 + \text{a constant}
(\text{where } a_1 = 10^{\log a_1}), \text{ likewise for } G_2, \text{ and where } A_{\text{star}} \text{ is the starved affinity, that is the affinity of}
the reaction to form one mole of the species at unit activity from the basis species in their defined
activities. The equation used arises from integrating $dG = -A/dx_i = -A/dn$ where $x_i$ is the reaction
progress variable, $dn/dx_i = 1$ is the reaction coefficient on the species, and $A = A_{\text{star}} - 2.303RT \log a$
is the chemical affinity.

Each objective function has an attribute (see attributes and structure) named ‘optimum’ that
takes the value of ‘minimal’ (SD, CV, RMSD, CVRMSD, DGMix, DDGMix, DGtr) or ‘maximal’ (logact,
shannon, qqr, spearman, pearson). This attribute is used in optimal/index and extremes to
identify the conditions having optimal values of the objective functions.

get.objfun returns the objective function named in objective, or produces an error if the function
has no ‘optimum’ attribute.

References


See Also

get.objfun for retrieving these functions by name, and revisit and findit for applications
of these functions in chemical systems.

Examples

```r
## a made-up system: 4 species, 1 condition
loga1 <- t(-4:-1)
loga2 <- loga1 + 1
stopifnot(qqr(loga1) < 1)
stopifnot(RMSD(loga1, loga1) == 0)
stopifnot(RMSD(loga1, loga2) == 1)
stopifnot(CVRMSD(loga1, loga2) == -0.4) # 1 / mean(-4:-1)
stopifnot(spearman(loga1, loga2) == 1)
stopifnot(spearman(loga1, rev(loga2)) == -1)
# less statistical, more thermodynamical...
stopifnot(all.equal(DGMix(loga1), -0.1234)) # as expected for decimal logarithms
stopifnot(all.equal(DDGMix(loga1, loga2), 0.0004))

## transforming an equilibrium assemblage of n-alkanes
```
Examples of Calculations for Proteins

Description

This page contains some examples of using the functions in CHNOSZ to calculate thermodynamic properties and relative stabilities of proteins.

See Also

For accessing, updating, and downloading amino acid compositions of proteins, see iprotein. For getting chemical formulas and stoichiometric coefficients in reactions of proteins, see protein.info. For more examples of metastable equilibrium calculations for proteins, see read.expr, more.aa, ionize.aa, and apc for reaction-path calculations.
Examples

## thermodynamic properties and activity diagrams

```r
prot4 <- c("LYSC_CHICK", "BPT1_BOVIN", "CYC_BOVIN", "MYG_PHYCA")
# aqueous protein properties (nonionized)
subcrt(prot4, T=seq(0, 200, 10))$out
# T-logfO2 activity diagram
basis("CHNOS")
species(prot4)
a <- affinity(T=c(0, 200), O2=c(-80, -40))
diagram(a)
```

## one way to calculate the standard Gibbs energy of a reaction to form an ionized protein at 100 degrees and pH 8

```r
basis("CHNOS")
# do this to auto-balance the formation reaction
Gr.nonionized <- subcrt("LYSC_CHICK", 1, T=100)@out@G
basis("pH", 8)
pinfo <- protein.info("LYSC_CHICK", round.it=FALSE, T=100)
Gr.ionization <- pinfo@G.Z - pinfo@G
# standard Gibbs energy of the reaction
# in cal/mol ionized protein:
Gr.ionized <- Gr.nonionized + Gr.ionization
# in cal/mol ionized residue:
Gr.ionized_residue <- Gr.ionized/protein.length("LYSC_CHICK")
```

## Standard molal entropy of a protein reaction

```r
basis("CHNOS")
# here we provide the reaction coefficients of the proteins (per protein backbone); 'subcrt' function calculates the coefficients of the basis species in the reaction
s <- subcrt(c("CSG_METTL", "CSG_METJA"), c(-1/530,1/530), T=seq(0, 350, length.out=50))
thermo.plot.new(xlim=range(s$out$T), ylim=range(s$out$S),
    xlab=axis.label("T"), ylab=axis.label("DS0r"))
lines(s$out$T, s$out$S)
# do it at high pressure as well
s <- subcrt(c("CSG_METTL", "CSG_METJA"), c(-1/530,1/530), T=seq(0,350,length.out=50), P=3000)
lines(s$out$T, s$out$S, lty=2)
# label the plot
title(main=paste("Standard molal entropy\n", "P = Psat (solid), P = 3000 bar (dashed)"))
s$reaction$coeff <- round(s$reaction$coeff, 3)
dsr <- describe.reaction(s$reaction, iname=c(1,2))
text(170, -3, dsr, cex=0.8)
```

### Equilibrium activity diagrams

## surface-layer proteins from Methanococcus and others

```
## as a function of oxygen fugacity, after Dick, 2008, Fig. 5b
# use old properties of [Met] to reproduce this example
```
# make our protein list
organisms <- c("METSC", "METJA", "METFE", "HALJP", "METVO", "METBU", "ACEKI", "GEOSE", "BACLI", "AERSA")
proteins <- c(rep("CSG", 6), rep("SLAP", 4))
proteins <- paste(proteins, organisms, sep="_")

# load the basis species and proteins
basis("CHNOS+")

# calculate affinities; we go to lower logF02 than Dick, 2008
# and find an interesting convergence of stabilities there
a <- affinity(O2=c(-100, -65))
# try normalize=FALSE to make Fig. 5a in the paper
e <- equilibrate(a, normalize=TRUE)
d <- diagram(e, ylim=c(-5, -1), legend.x=NA, names=organisms)

# add water stability line
abline(v=-83.1, lty=2)
title(main="Surface-layer proteins, after Dick, 2008")

# checking the geometry of the diagram
# most preominant along the x-axis
stopifnot(organisms[unique(which.max(e$loga.equil))]==
c("METFE", "METJA", "METVO", "HALJP"))

# stability order close to logF02=-83.1
stopifnot(order(as.data.frame(e$loga.equil)[62,],
    decreasing=TRUE)==c(2, 6, 7, 5, 3, 1, 9, 8, 10, 4))

# reset thermodynamic database
data(thermo)

## relative stabilities of bovine proteins
## as a function of temperature along a glutathione redox buffer
mod.buffer("GSH-GSSG", c("GSH", "GSSG"), logact=c(-3, -7))
basis(c("CO2", "H2O", "NH4+", "SO4-2", "H2", "H+"),
    c(-1, 0, -4, -4, 999, -7))
basis("H2", "GSH-GSSG")
basis("CO2", "gas")
prot <- c("CYC", "RNA1", "BPT1", "ALBU", "INS", "PRIO")
species(prot, "BOVIN")
a <- affinity(T=c(0, 200))
# set line colors according to oxidation state of carbon
ZC <- ZC(species())$ispecies
col <- rep("red", length(prot))
col[ZC > 0] <- "blue"
e <- equilibrate(a, normalize=TRUE)
d <- diagram(e, col=col, legend.x=NA)
title(main="Bovine proteins, GSH/GSSG redox buffer")

## relative stabilities of plasma proteins,
## using chemical activities of H2 and O2
## find that insulin is very stable in oxidizing conditions
basis(c("CO2", "NH3", "H2S", "H2", "O2"), "aq", c(-3, -3, -10))
f <- system.file("extdata/abundance/AA83.csv", package="CHNOSZ")
pdat <- read.csv(f, as.is=TRUE)
### Calculations showing effects of ionization of proteins

#### Eh-pH diagrams for intra/extracellular proteins

```r
calculate(affinity(O2=c(-90, -75)))
```

#### Buffer + ionization: Metastabilities of thiol peroxidases from model bacteria

```r
# (ECOLI, BACSU mesophile; AQUAE thermophile, THIDA acidophile, BACHD alkaliphile)

organism <- c("ECOLI", "AQUAE", "BACSU", "BACHD", "THIDA")

species("TPX", organisms)  # create a buffer with our proteins in it

mod.buffer("TPX", paste("TPX", organisms, sep=" "))

# set up the buffered activities

basis(c("CO2", "H2O", "NH3", "O2", "TPX")

a <- affinity(return.buffer=TRUE, T=50)

basis(c("CO2", "H2O", "NH3", "O2"), as.numeric(a[1:4]))

a <- affinity(pH=c(0, 14, 200), T=c(25, 70, 200))

# title(main="Thiol peroxidases from bacteria")

text(0.5, 66, describe.basis(thermo$basis[-6], oneline=TRUE), adj=0)
```

### Buffer + ionization: relative stabilities
## Calculation of Thermodynamic Properties of Proteins

### Description

Calculate chemical formulas, lengths, standard Gibbs energies and net charges, stoichiometric coefficients of basis species in reactions to form proteins (possibly per residue), and show steps in calculation of chemical activities of proteins in metastable equilibrium.

### Usage

```r
protein.formula(protein, organism = NULL, residue = FALSE)
protein.length(protein, organism = NULL)
protein.info(protein, T = 25, residue = FALSE, round.it = FALSE)
protein.basis(protein, T = 25, normalize = FALSE)
protein.equilib(protein, T = 25, loga.protein = 0)
MP90.cp(protein, T)
group.formulas()
```

### Arguments

- **protein**: character, names of proteins; numeric, species index of proteins; data frame; amino acid composition of proteins
- **organism**: character, names of organisms
protein.info

**Details**

These functions accept `protein` (and optionally `organism`) in the same way as `ip2aa`, that is, as a protein name (optionally with the organism part separated), one or more row numbers in `thermo$protein` that can be identified using `iprotein`, or a data frame in the format of `thermo$protein`.

- `protein.formula` returns a stoichiometric matrix representing the chemical formulas of the proteins that can be passed to e.g. `mass` or `ZC`. The amino acid compositions are multiplied by the output of `group.formulas` to generate the result. `group.formulas` returns the chemical formulas of each of the 20 common amino acid residues in proteins, as well as the terminal -H and -N (treated as the `[H2O]` group).

- `protein.length` returns the lengths (number of amino acids) of the proteins.

- `protein.info` tabulates some properties of proteins. A data frame is returned with a row for each protein, and columns named ‘`protein`’, ‘`length`’, ‘`formula`’, ‘`G`’, ‘`Z`’, ‘`G.Z`’ and ‘`ZC`’, indicating the names of the proteins, their lengths, chemical formulas, and values of the standard molal Gibbs energy of the neutral (nonionized) proteins, net charges and standard molal Gibbs energy of the ionized proteins, and average oxidation states of carbon. ‘`Z`’ and ‘`G.Z`’ are calculated using `ionize.aa` with values of pH taken from `thermo$basis`; ‘`Z`’ and ‘`G.Z`’ become NA if the `basis` species are not loaded or ‘`H+`’ is not in the basis definition. ‘`ZC`’ is calculated using `ZC`. The value of `T` indicates the temperature at which to calculate the Gibbs energies and net charge. The values of standard Gibbs energy are shown in cal/mol; these and other numeric values are rounded at a set number of digits if `round.it` is TRUE. The values (including chemical formula but not ‘`ZC`’) are divided by the lengths of the proteins if `residue` is TRUE.

The following two functions depend on an existing definition of the basis species:

- `protein.basis` calculates the numbers of the basis species (i.e. opposite of the coefficients in the formation reactions) that can be combined to form the composition of each of the proteins. The basis species must be present in `thermo$basis`, and if ‘`H+`’ is among the basis species, the ionization states of the proteins are included. As with `protein.info`, the ionization state of the protein is calculated at the pH defined in `thermo$basis` and at the temperature specified by the `T` argument. If `normalize` is TRUE, the coefficients on the basis species are divided by the lengths of the proteins.

- `protein.equil` produces a series of messages showing step-by-step a calculation of the chemical activities of proteins in metastable equilibrium. For the first protein, it shows the standard Gibbs energies of the reaction to form the nonionized protein from the basis species and of the ionization reaction of the protein (if ‘`H+`’ is in the basis), then the standard Gibbs energy/RT of the reaction to form the (possibly ionized) protein per residue. The per-residue values of ‘`logQstar`’ and ‘`Astar/RT`’ are also shown for the first protein. Equilibrium calculations are then performed, only if more than one protein is specified. This calculation applies the Boltzmann distribution to the calculation of the equilibrium degrees of formation of the residue equivalents of the proteins, then converts them to activities of proteins taking account of `loga.protein` and protein length. If the `protein` argument is numeric (indicating rownumbers in `thermo$protein`), the values of
‘Astar/RT’ are compared with the output of `affinity`, and those of the equilibrium degrees of formation of the residues and the chemical activities of the proteins with the output of `diagram`. If the values in any of these tests are are not `all.equal` an error is produced indicating a bug.

`MP90.cp` takes protein (name of protein) and `T` (one or more temperatures in °C) and returns the additive heat capacity (J mol⁻¹) of the unfolded protein using values of heat capacities of the residues taken from Makhatadze and Privalov, 1990. Those authors provided values of heat capacity at six points between 5 and 125 °C; this function interpolates (using `splinefun`) values at other temperatures.

### References


### See Also

`ionize.aa` for an example that compares `MP90.cp` with heat capacities calculated in CHNOSZ at different temperatures and pHs. The functions for interacting with the database of amino acid compositions of proteins are documented at `iprotein`, and examples of relative stability calculations can be found on the `protein` help page.

### Examples

```r
## example for chicken lysozyme C
# index in thermo$protein
ip <- iprotein("LYSC_CHICK")
# amino acid composition
ip2aa(ip)
# length and chemical formula
protein.length(ip)
protein.formula(ip)
# formula, Gibbs energy, average oxidation state of carbon
protein.info(ip)
# as above, now with charge and Gibbs energy of ionized protein at pH 7
# basis("CHNOS+")
protein.info(ip)
# group additivity for thermodynamic properties and HKF equation-of-state
# parameters of non-ionized protein
aa2eos(ip2aa(ip))
# calculation of standard thermodynamic properties
# (subcrt uses the species name, not ip)
subcrt("LYSC_CHICK")
# affinity calculation, protein identified by ip
affinity(iprotein=ip)
# affinity calculation, protein loaded as a species
```
species("LYSC_CHICK")
affinity()
# NB: subcrt() only shows the properties of the non-ionized 
# protein, but affinity() uses the properties of the ionized 
# protein if the basis species have H+

## these are all the same
protein.formula("P53_PIG")
protein.formula(iprotein("P53_PIG"))
protein.formula(ip2aa(iprotein("P53_PIG")))

## steps in calculation of chemical activities of two proteins 
## in metastable equilibrium, after Dick and Shock, 2011
protein <- iprotein(c("CSG_METVO", "CSG_METJA"))
# clear out amino acid residues loaded by the example above
# ( in affinity(iprotein=ip) )
data(thermo)
# load supplemental database to use "old" [Met] sidechain group
add.obigt()
# set up the basis species to those used in DS11
basis("CHNOS+")
# note this yields logaH2 = -4.657486
swap.basis("O2", "H2")
# demonstrate the steps of the equilibrium calculation
protein.equil(protein, loga.protein=-3)
## we can also look at the affinities
## (Reaction 7, Dick and Shock, 2011)
# A/2.303RT for protein at unit activity (A-star for the protein)
a <- affinity(iprotein=protein[1], loga.protein=0)
Astar.protein <- a$svalues[[1]]
# divide affinity by protein length (A-star for the residue)
pl <- protein.length(protein[1])
Astar.residue <- a$svalues[[1]]/pl  # 0.1893, Eq. 11
# A/2.303RT per residue corresponding to protein activity of 10^-3
loga.residue <- log10(pl*10^-3)
Aref.residue <- Astar.residue - loga.residue  # 0.446, after Eq. 16
# A-star of the residue in natural log units (A/RT)
log(10) * Astar.residue  # 0.4359, after Eq. 23

## using protein.formula: average oxidation state of
## carbon of proteins from different organisms
# get amino acid compositions of microbial proteins
# generated from the RefSeq database
file <- system.file("extdata/refseq/protein_refseq.csv.xz", package="CHNOSZ"
ip <- add.protein(read.aa(file))
# only use those organisms with a certain 
# number of sequenced bases
ip <- ip[as.numeric(thermo$protein$abbrv[ip]) > 100000]
pf <- protein.formula(thermo$protein[ip, ])
zc <- ZC(pf)
# the organism names we search for
# " " matches all organisms


"Nitro", "Desulfo", "Chloro", "Geo", "Methano",
"Thermo", "Pyro", "Sulfo", "Buchner", ""

tps <- thermo$protein$ref[ip]
plot(0, 0, xlim=c(1, 15), ylim=c(-0.3, -0.05), pch="",
ylab="average oxidation state of carbon in proteins",
xlab="", xaxt="n", mar=c(6, 3, 1, 1))
for(i in 1:length(terms)) {
  it <- grep(terms[i], tps)
  zct <- zc[it]
  points(jitter(rep(i, length(zct))), zct, pch=20)
}
terms[15] <- paste("all", length(ip))
axis(1, 1:15, terms, las=2)
title(main=paste("Average Oxidation State of Carbon: ",
  "Total Protein per taxID in NCBI RefSeq", sep="\n"))

---

**Experimental Data for Protein Abundances and Localizations**

**Description**

Get abundance data from a protein expression experiment and add the proteins to the working instance of CHNOSZ. Retrieve the amino acid compositions of proteins with localizations and abundances taken from the YeastGFP project.

**Usage**

```r
stress(condition, organism)
yeastgfp(location, exclusive = TRUE)
read.expr(file, idcol, abundcol, filter=NULL)
```

**Arguments**

- `condition` character, name of condition of stress response experiment
- `organism` character, organism in stress response experiment
- `location` character, name of subcellular location (compartment)
- `exclusive` logical, report only proteins exclusively localized to a compartment?
- `file` character, name of file with sequence IDs and abundance data
- `idcol` character, name of the column with sequence IDs
- `abundcol` character, name of the column with abundances
- `filter` list, optional filters to apply
Details

read.expr, yeastgfp and stress all interact with data files stored in extdata/abundance to retrieve identities and possibly abundances of proteins in certain conditions.

stress is the simplest of these functions since the source of its data, stress.csv, lists proteins without any abundance data. condition indicates the name of the stress response experiment (column name of stress.csv, e.g. 'low.C') and organism indicates the organism ('Eco' or 'Sce').

The yeastgfp function returns the identities and abundances of proteins with the requested subcellular localization(s) (specified in location) using data from the YeastGFP project that is stored in extdata/abundance/yeastgfp.csv.xz. The default value of exclusive (FALSE) tells the function to grab all proteins that are localized to a compartment even if they are also localized to other compartments. If exclusive is TRUE, only those proteins that are localized exclusively to the requested compartments are identified, unless there are no such proteins, then the non-exclusive localizations are used (applies to the 'bud' localization).

read.expr reads a file (CSV format) that contains protein sequence names or IDs and protein abundance data. idcol and abundcol are either the names of the columns holding the sequence IDs and protein abundances, or numeric values indicating the column numbers where these data are found. The column indicated by abundcol might not actually be abundance (it is likely to be abundance ratios). The data can be filtered to only include records that contain the term in the named argument filter, the name of which indicates the column to apply the filter to.

Value

Each of these functions returns a list with elements named protein (names of proteins) and abundance (counts or concentrations without any conversion from the units in the data file). For stress, the abundance value is all 1's. For yeastgfp, if location is NULL, the function returns the names of all known locations, and if the length of location is >1, the protein and abundance values are lists of the results for each location.

References


**See Also**

*more.aa* for getting the amino acid compositions of proteins whose names are returned by these functions.

**Examples**

```r
## overall oxidation state of proteins exclusively localized
ty <- yeastgfp("cytoplasm")
aa <- more.aa(y$protein, "Sce")
aaavg <- asum(aa, average=TRUE)
ZC(protein.formula(aaavg))
# the average composition weighted by abundance
waaavg <- asum(aa, abundance=y$abundance, average=TRUE)
ZC(protein.formula(waaavg))

## read.expr using one of the provided data files,
## from Ishihama et al., 2008
file <- system.file("extdata/abundance/ISR+08.csv.xz", package="CHNOSZ")
# read all protein names and abundances in ID and emPAI columns
# (emPAI - exponentially modified protein abundance index)
expr <- read.expr(file, "ID", "emPAI")
# scatter plot of average oxidation state and emPAI
aa <- more.aa(expr$protein, "Eco")
pf <- protein.formula(aa)
zc <- ZC(pf)
# note we specify ylim here that excludes some high-emPAI values
plot(zc, expr$abundance, xlab=expr.property("ZC"), ylim=c(0, 90), ylab="emPAI", main="Proteins in E. coli cytosol\nAbundance vs oxidation state of carbon")
legend("topleft", pch=1, legend="Ishihama et al., 2008")
# what if we just want kinases?
# "description" is the name of the column where we search for "kinase"
expr.kinase <- read.expr(file, "ID", "emPAI", list(description="kinase"))

## read.expr using a different data file,
## from Anderson and Anderson, 2003
file <- system.file("extdata/abundance/AA03.csv", package="CHNOSZ")
# look for proteins described as "Complement"
read.expr(file, "name", "log10(pg/ml)", list(description="Complement"))

## speciation diagram for ER.to.Golgi proteins
## proteins as a function of logF02, after Dick, 2009
# add old parameters for [Met] sidechain to database
add.obigt()
y <- yeastgfp("ER.to.Golgi")
```
# don't use those with NA abundance
ina <- is.na(y$abundance)
# get the amino acid compositions of the proteins
aa <- more.aa(y$protein[!ina], "Sce")
# add proteins to thermo$protein
ip <- add.protein(aa)
# use logarithms of activities of proteins such
# that total activity of residues is unity
pl <- protein.length(ip)
logact <- unitize(rep(1, length(ip)), pl)
# load the proteins
basis("CHNOS+")
a <- affinity(02=c(-80, -73), iprotein=ip, loga.protein=logact)
# make a speciation diagram
e <- equilibrate(a, normalize=TRUE)
diagram(e, ylim=c(-4.9, -2.9))
# where we are closest to experimental log activity
logf02 <- rep(logf02, length(ip))
abline(v=logf02[1], lty=3)
# scale experimental abundances such that
# total activity of residues is unity
logact.expt <- unitize(log@y(abundance[!ina]), pl)
# plot experimental log activity
points(logf02, logact.expt, pch=16)
text(logf02+0.5, logact.expt, y$protein[!ina])
# add title
title(main=paste("ER.to.Golgi; points - relative abundances",
               "from YeastGFP. Figure after Dick, 2009", sep="\n"))
# restore default thermodynamic database
data(thermo)

# examples using stress() #

# predominance fields for overall protein compositions
# induced and repressed in an/aerobic carbon limitation
# (experiments of Tai et al., 2005)
# the activities of ammonium and sulfate
# are similar to the non-growth-limiting concentrations
# used by Boer et al., 2003
basis(c("glucose", "H2O", "NH4+", "hydrogen", "SO4-2", "H4+"),
c(-1, 0, -1.3, 999, -1.4, -7))
# the names of the experiments in thermo$stress
expt <- c("Clim.aerobic.down", "Clim.aerobic.up",
           "Clim.anaerobic.down", "Clim.anaerobic.up")
# here we use abundance to indicate that the protein
# compositions should be summed together in equal amounts
for(i in 1:length(expt)) {
  p <- stress(expt[i], "Sce")
aa <- more.aa(p$protein, "Sce")
aa <- aasum(aa, average=TRUE, protein=expt[i])
  add.protein(aa)
### Examples using yeastGFP()

#### Oxygen fugacity - activity of H2O predominance

#### diagrams for proteologs for 23 YeastGFP localizations

# need old properties of [Met] to reproduce this example
data(thermo)
add.obigt()

# arranged by decreasing metastability:
# order of this list of locations is based on the
# (dis)appearance of species on the current set of diagrams
names <- c("vacuole", "early.Golgi", "ER", "lipid.particle",
    "cell.periphery", "ambiguous", "Golgi", "mitochondrion",
    "bud", "actin", "cytoplasm", "late.Golgi",
    "endosome", "nucleus", "vacuolar.membrane", "punctate.composite",
    "peroxisome", "ER.to.Golgi", "nucleolus", "spindle.pole",
    "nuclear.periphery", "bud.neck", "microtubule")

# define the system
basis("CHNOS+)

# get protein names and abundances in each location
gfp <- yeastGFP(names)

# get amino acid compositions of proteins
aa <- more.aa(gfp$protein, "Sce")

# calculate average amino acid compositions
for(i in 1:length(names)) {
    avga <- aasum(aa[[i]], gfp$abundance[[i]], average=TRUE, protein=names[i])
    add.protein(avga)
}

species(names, "Sce")
a <- affinity(H2O=c(-5, 0, 256), O2=c(-80, -66, 256))

# setup the plot
layout(matrix(c(1, 1, 2:7), byrow=TRUE, nrow=4), heights=c(0.7, 3, 3, 3))
par(mar=c(0, 0, 0, 0))

plot.new()
text(0.5, 0.5, paste("Subcellular proteins of S. cerevisiae,",
    "after Dick, 2009
"), cex=1.5)
text(0.5, 0.2, describe.basis(ibasis=c(1, 3, 4, 6), oneline=TRUE), cex=1.5)
opar <- par(mar=c(3, 4, 1, 1), xpd=TRUE)
fill <- heat.colors(length(names))
inames <- 1:length(names)
for(i in 1:length(nloc)) {
  diagram(a, normalize=TRUE, names=names[inames], groups=as.list(inames),
          fill=fill[inames], cex.axis=0.75, cex.names=1)
  label.plot(letters[i])
  title(main=paste(length(inames), "locations"))
  # take out the stable species
  inames <- inames[-(1:nloc[i])]
}
# return to plot defaults
layout(matrix(1))
par(opar)
# reset thermodynamic database
data(thermo)

---

**Plots and Optima of Objective Functions**

**Description**

Calculate values of an objective function from logarithms of activities of chemical species and (for some objectives) reference logarithms of activity. Make line or contour plots showing the values of the objective function and the positions of the optima (minimum or maximum).

**Usage**

```r
revisit(eout, objective = "CV", loga2 = NULL, loga0 = NULL,
       ispecies = NULL, col = par("fg"), yline = 2, ylim = NULL,
       cex = par("cex"), lwd = par("lwd"), mar = NULL, side = 1:4,
       xlim = NULL, labcex = 0.6, pch = 1, main = NULL, plot.it = NULL,
       add = FALSE, plot.optval = TRUE, style.2D = "contour")
```

**Arguments**

- `eout` list, output from `equilibrate`, containing logarithms of activities of species
- `objective` character, name of `objective` function
- `loga2` numeric vector, reference values of logarithm of activities
- `loga0` numeric vector, logarithm of activities to calculate activity ratios
- `ispecies` numeric, which species to consider
- `col` character, color to use for points or lines
- `yline` numeric, margin line for y-axis label
**Details**

`revisit` is used to calculate the variation in the equilibrium logarithms of chemical activity (supplied in `eout`) or to compare the calculated values with reference (e.g. measured) values (`logaR`). Usually, the output of `equilibrate` is used as the value for `eout`. The type of calculation is indicated by `objective`, giving the name of an `objective` function. Generally, `loga2` is expressed in base-10 logarithms. However, if `loga0` (base 10) is supplied, it is used to calculate the base-2 log ratio (`log2(a1/a0)`); these calculated values are then compared with values in `logaR` interpreted as base-2 logarithms.

Internally, the list of logarithms of chemical activities in `eout$loga.equil` is passed as `logaQ` to the objective function. If the objective function has an argument `aQ` instead of `logaQ`, the activities instead of their logarithms are passed to the function. Generally, `loga2` must be a numeric vector with length equal to that of `loga1` (i.e., number of species). However, if a single numeric value is supplied for `logaR`, it is recycled to the length of `loga1`.

For calculations at a single condition (0-D, no variation), with the ‘qqr’ objective, a quantile-quantile plot (`qqnorm`) is shown. For ‘rmsd’ and other objective functions having reference values (`logaR`), a scatter plot is shown with a smooth line calculated using `loess.smooth`. The line can be suppressed using `lwd=NULL`. Otherwise, no plot is made for 0-D calculations for the other objective functions.

If `plot.it` is TRUE, and `eout` is the output from `equilibrate`, and the number of variables is 1 or 2, the results are plotted — a line diagram in 1 dimension or a contour plot in 2 dimensions. `style.2D` can be set to `image` to fill the plot with colors instead of the `contour` plot that is the default.

If `plot.optval` is TRUE, the location of the optimum (or optima) is indicated by a dashed vertical line(s) on a 1-D plot or a point(s) marked by an asterisk on a 2-D plot. Also, on 2-D plots, the locations of the optima at each grid line perpendicular to the `x` and `y` axes are plotted. These points follow major ridges or valleys, and are plotted as dashed lines colored green for the `x` and blue for the `y` values returned by `extremes`.
An alternative source for the `eout` argument is any list of numeric values, each element of which corresponds to a different observation (such as a single species), all having the same dimensions (as vectors, matrices or higher-dimensional arrays). In this case, plotting is disabled, since the names of the variables are not in the input.

`revisit` is a partial anagram of `diversity`, which was the provisional name of the function but was changed in CHNOSZ-0.9. While the `diversity` function (in `vegan`) operates on a matrix with (biological) species on the columns, `revisit` operates on a list with (chemical) species as the elements of the list. The name of the `H` output value is the conventional symbol for the Shannon diversity index, which was the first target statistic to be implemented in `revisit`.

Value

`revisit` returns a list containing at least an element named `H`, giving the calculated values of the objective function. For 1 or 2 dimensions of variability of chemical conditions, the output also contains the elements `ixopt` and `iyopt` (1-D and 2-D) and `iyopt` and `yopt` (2-D) indicating the positions and values of the optimum. For calculations in more than two dimensions, the output contains `iopt` which is a matrix.

`optimum.index` returns the index (or indices) of the optimal values in `z` using `array(...)`, `arr.ind=TRUE`.
`extremes` returns `x` values that are the column numbers where the optimum is found for each row, and `y` values that are the row numbers where the optimum is found for each column. The `optimum` attribute of the objective function indicates whether minimal or maximal values are used.

See Also

`findit` for gridded search of chemical activities, temperature and/or pressure that optimize the objective function.

Examples

```r
# example of defining a new objective function
# count the species with logarithms of activity greater than loga2
count <- function(loga1, loga2) rowSums(loga1 > loga2)
# set the attribute indicating the type of optimum
attr(count, "optimum") <- "maximal"
# equilibrate a system of amino acids
basis("CHNOS")
species(aminoacids("")
a <- affinity(O2=c(-80, -60))
e <- equilibrate(a)
# make a plot
r <- revisit(e, "count", -5)
title(main="amino acids with metastable log activities > -5")

# can also make a 2-D plot
a <- affinity(O2=c(-74, -60), H2O=c(-3, 3))
e <- equilibrate(a)
r <- revisit(e, "count", -5, style.2D="image", plot.optval=FALSE)
title(main="amino acids with metastable log activities > -5")
```
revisit

```r
# 'revisit' calculations for amino acids
opar <- par(mfrow=c(2, 2))
basis("CHNOS+")
species(aminoacids(""))
# chemical affinities as a function of logarithm of oxygen fugacity
a <- affinity(O2=c(-85, -60))
# shows the equilibrium abundances of the amino acids
e <- equilibrate(a)
diagram(e)
mtitle(c("20 amino acids", "balanced on CO2"))
# show a legend with input constraints
db <- describe.basis(ibasis=3)
dp <- describe.property("T", 25)
legend("bottomright", c(dp, db))
# default is to plot coefficient of variation
r <- revisit(e)
# show a title with the optimal conditions
mincv <- format(r$optimum, digits=3)
t1 <- paste("minimum coeff of variation," , mincv, "at:" )
# the logfO2 that minimized the C.V.
basis("O2", r$x)
t2 <- describe.basis(ibasis=5)
mtitle(c(t1, as.expression(t2)))
# chemical affinities as a function of two other variables
a <- affinity(NH3=c(-10, 10, 40), T=c(0, 80, 40))
diagram(a, fill="heat")
# show a legend with input constraints
db <- describe.basis(ibasis=5)
legend("bottomright", as.expression(db))
# contour plot of the CV
e <- equilibrate(a)
r <- revisit(e)
# show a title with the optimal conditions
mincv <- format(r$optimum, digits=3)
t1 <- paste("minimum coeff of variation," , mincv, "at:" )
# the logaNH3 and T that minimized the C.V.
basis("NH3", r$x)
db <- describe.basis(ibasis=3)
dp <- describe.property("T", r$y)
t2 <- substitute(list(dp, db), list(dp=dp[[1]], db=db[[1]]))
mtitle(c(t1, as.expression(t2)))
par(opar)
```

## calculations for proteins in Pelagibacter ubique
# using grep.file, read.fasta, add.protein
f <- system.file("extdata/fasta/HTCC1062.faa.xz", package="CHNOSZ")
# what proteins to select (set to "" for all proteins)
w <- "ribosomal"
# locate entries whose names contain w
j <- grep.file(f, w)
# get the amino acid compositions of these proteins
aa <- read.fasta(f, j)
# add these proteins to CHNOSZ's inventory
sideeffects

Side effects of functions in CHNOSZ

Description

Some functions in the package access thermodynamic data and system definitions contained in the thermo data object, as well as modify the object. This help topic should help users understand the major side effects, but does not contain a comprehensive description of these interactions (the code is the ultimate reference).

Details

When the package is loaded, .onAttach creates a list object named thermo that is placed in an environment named ‘CHNOSZ’. Some functions in CHNOSZ have side effects that modify the contents of thermo; all such changes can be reverted, and the object restored to its original state, by calling data(thermo).

The ‘CHNOSZ’ environment is not (as of CHNOSZ 1.0.0) attached, rather the thermo object is accessed in functions using get (as in get("thermo")), assign(assign("thermo", thermo, "CHNOSZ")) and occasionally with(with(as.environment("CHNOSZ"), ...)).
In the functions in the package, the greatest number of accessions are to the thermodynamic database (\code{thermo$obigt}), followed by the basis and species definitions (\code{thermo$basis} and \code{thermo$species}). For example, \code{info} can be used to look up thermodynamic data in \code{thermo$obigt} by the name or chemical formula of a species. As another example, \code{subcrt} attempts to balance unbalanced chemical reactions with the user-defined basis species in \code{thermo$basis}.

Some functions modify the thermodynamic database or system definition in \code{thermo}. These are “side effects”, since the functions have an effect on the state of the program that persists beyond the lifetime of the objects returned by the functions. In the code, side effects can be recognized by assignment to the ‘thermo’ object in the ‘CHNOSZ’ environment, \code{i.e. assign("thermo", thermo, “CHNOSZ")} (the unquoted \code{thermo} here refers to the object that was manipulated internally by a function and is now being assigned to the environment).

Side effects are not highly desirable in functional programming languages such as R. The reason this design is adopted in CHNOSZ is that interactive use of \code{basis} and \code{species} appeared to the author, in the early stages of developing the package and of learning R, to be facilitated by not requiring users to assign the results of these functions to objects. Instead, using side effects, the program “remembers” the results of these function calls. Experience has shown that this design is usable (especially for new users), and is adaptable to many usage scenarios, but the dependence on side effects probably should be eliminated in the future.

The two \textit{major} side effects, that most users will encounter, are the basis and species definitions. These functions and a few other modifications (writing) and accessions (reading) of data objects are listed below. The names of objects in this table refer to the components of the \code{thermo} object; for example, one can type \code{thermo$opt} at the command line to access all of the contents of the \code{opt} component, including those not listed in the table.

<table>
<thead>
<tr>
<th>object</th>
<th>writer</th>
<th>reader</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>obigt</td>
<td>\code{mod.obigt}</td>
<td>\code{info}</td>
<td>thermodynamic database</td>
</tr>
<tr>
<td>basis</td>
<td>\code{basis}</td>
<td>\code{species,subcrt}</td>
<td>basis definition</td>
</tr>
<tr>
<td>species</td>
<td>\code{species}</td>
<td>\code{affinity}</td>
<td>species definition</td>
</tr>
<tr>
<td>\code{opt$T.units}</td>
<td>\code{T.units}</td>
<td>\code{convert}</td>
<td>units</td>
</tr>
<tr>
<td>\code{opt$water}</td>
<td>–</td>
<td>\code{water}</td>
<td>formulation for properties of water</td>
</tr>
<tr>
<td>\code{opt$Tr,Pr}</td>
<td>–</td>
<td>\code{GHS}</td>
<td>reference temperature and pressure</td>
</tr>
<tr>
<td>\code{opt$state}</td>
<td>–</td>
<td>\code{info}</td>
<td>physical state</td>
</tr>
<tr>
<td>\code{opar}</td>
<td>\code{thermo.plot.new}</td>
<td>–</td>
<td>graphical parameters</td>
</tr>
</tbody>
</table>

Beginning with CHNOSZ version 1.0.0, the “superassignment” operator (\code{<<-}) is no longer used in functions. However, if \textit{you} wish to alter something in \code{thermo} in an interactive session, it is recommended to use the \code{<<-} operator, instead of \code{<-}. This way, your changes to the \code{thermo} object occur in the ‘CHNOSZ’ environment, which is where the functions in CHNOSZ expect to find it, rather than being saved to the global environment. An example of changing \code{thermo$opt$water} in this manner can found in the help page for \code{water}.

**See Also**


\floatfoot
Examples

```r
data(thermo) # side effect: reset the system definition
basis() # NULL
basis("CHNOS") # side effect: define the basis species
basis() # not NULL
data(thermo) # side effect: reset the system definition
basis() # NULL
```

---

**species**

### Species of Interest

**Description**

Define the species of interest in a system; modify their physical states and logarithms of activities.

**Usage**

```r
species(species = NULL, state = NULL, delete = FALSE, index.return = FALSE)
species.basis(species)
```

**Arguments**

- `species`: character, names or formulas of species to add to the species definition; numeric, rownumbers of species to modify or delete
- `state`: character, physical states; numeric, logarithms of activities or fugacities
- `delete`: logical, delete the species identified by numeric values of `species` (all species if that argument is missing)?
- `index.return`: logical, return the affected rownumbers of `thermo$species` instead of its contents?

**Details**

After defining the `basis` species of your system you can use `species` to identify for the program the species of interest. A species is operationally a combination of a name and state, which are columns of the thermodynamic database in `thermo$obigt`. The function operates on one or more character values of `species`. For each first match of `species` (optionally restricted to a state among 'aq', 'cr', 'gas', 'liq') to the name of a species or a formula or abbreviation in the thermodynamic database, a row is added to `thermo$species`.

The data frame in `thermo$species` holds the identifying characteristics of the species as well as the stoichiometric reaction coefficients for the formation of each of the species from the basis species, and reference settings for the logarithms of activities or fugacities used in calculations of `affinity`. The default values for logarithms of activities are -3 for aqueous ('aq') species and 0 for others.

If `state` is `NULL` (the default), species in any state can be matched in the thermodynamic database. If there are multiple matches for a species, the one that is in the state given by `thermo$opt$state` is chosen, otherwise the matching (or n'th matching duplicate) species is used. Note that the states of species representing phases of minerals that undergo phase transitions are coded as 'cr1', 'cr2',
'cr', ..., (phases with increasing temperature). If state is 'cr' when one of these minerals is matched, all the phase species are added.

To modify the logarithms of activities of species (logarithms of fugacities for gases) provide one or more numeric values of species referring to the rownumbers of the species dataframe, or species NULL, to modify all currently defined species. The values in state, if numeric, are interpreted as the logarithms of activities, or if character are interpreted as states to which the species should be changed. If species is numeric and delete is TRUE, the rows representing these species are deleted from the dataframe; if the only argument is delete and it is TRUE, all the species are removed.

species.basis is the function used by species to calculate the coefficients of reactions to form the species from the basis species. It accepts one or more species formulas or indices, and produces an error if either the basis species are not defined, or they do not contain all of the elements in any of the species.

Value

With no arguments or when adding species, species returns the value of thermo$species, unless index.return is TRUE, when the function returns the rownumbers of thermo$species having the new species. With 'delete=TRUE', the value is the definition that existed prior the deletion; with 'delete=TRUE' and 'species' not NULL, the number of species remaining after the selected ones have been deleted, or NULL if no species remain.

See Also

Use info to search the thermodynamic database without adding species to the system. basis is a prerequisite for species.

Examples

```r
# set up the basis species
basis("CHNOS")
# show the formation reactions of some species
ispecies <- info(c("glutamic acid","phenylalanine"))
species.basis(ispecies)
# add, modify, delete species
species(c("CO2","NH3"))  # aqueous species
species(c("CO2","NH3"),"gas")  # gases
# delete the first couple of species
species(1:2,delete=TRUE)
# modify the logarithms of activities (actually
# fugacities) of the remaining species
species(1:2,c(-2,-5))
# set the species to aqueous
species(1:2,"aq")
# delete all the species (returns the existing species
# definition, then deletes the species)
sd <- species(delete=TRUE)

# changing the elements in the basis definition
# causes species to be deleted
```
subcrt

Properties of Species and Reactions

Description

Calculate the standard molal thermodynamic properties of one or more species or a reaction between species as a function of temperature and pressure. Import or export thermodynamic data in SUPCRT format.

Usage

```
subcrt(species, coeff = 1, state = NULL, property = c("logK","G","H","S","V","Cp"), T = seq(273.15,623.15,25), P = "Psat", grid = NULL, convert = TRUE, exceed.Ttr = FALSE, logact = NULL, action.unbalanced = "warn", IS = 0)
```

Arguments

- `species` character, name or formula of species, or numeric, rownumber of species in thermobigt
- `coeff` numeric, reaction coefficients on species
- `state` character, state(s) of species
- `property` character, property(s) to calculate
- `T` numeric, temperature(s) of the calculation
- `P` numeric, pressure(s) of the calculation, or character, ‘Psat’
- `grid` character, type of P×T grid to produce (NULL, the default, means no gridding)
- `exceed.Ttr` logical, calculate Gibbs energies of phases of minerals and of other species beyond their transition temperatures?
- `logact` numeric, logarithms of activities of species in reaction
- `convert` logical, are input and output units of T and P those of the user (TRUE) (see T.units), or are they Kelvin and bar (FALSE)?
- `action.unbalanced` character ‘warn’ or NULL, what action to take if unbalanced reaction is provided
- `IS` numeric, ionic strength(s) at which to calculate apparent molal properties, mol kg⁻¹
subcrt calculates the standard molal thermodynamic properties of species and reactions as a function of temperature and pressure. For each of the species (a formula or name), optionally identified in a given state, the standard molal thermodynamic properties and equations-of-state parameters are retrieved via info (except for H$_2$O liquid). The standard molal properties of the species are computed using equations-of-state functions for aqueous species (hkf), crystalline, gas, and liquid species (cgl) and liquid or supercritical H$_2$O (water).

T and P denote the temperature and pressure conditions for the calculations and should generally be of the same length, unless P is ‘Psat’ (the default; see water). Argument grid if present can be one of T × P or P × T grid. The propertys to be calculated can be one or more of those shown below:

<table>
<thead>
<tr>
<th>Property</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>rho</td>
<td>g cm$^{-3}$</td>
</tr>
<tr>
<td>logK</td>
<td>dimensionless</td>
</tr>
<tr>
<td>G</td>
<td>(cal</td>
</tr>
<tr>
<td>H</td>
<td>(cal</td>
</tr>
<tr>
<td>S</td>
<td>(cal</td>
</tr>
<tr>
<td>V</td>
<td>cm$^3$ mol$^{-1}$</td>
</tr>
<tr>
<td>Cp</td>
<td>(cal</td>
</tr>
<tr>
<td>E</td>
<td>cm$^3$ K$^{-1}$</td>
</tr>
<tr>
<td>kT</td>
<td>cm$^3$ bar$^{-1}$</td>
</tr>
</tbody>
</table>

(Note that E and kT can only be calculated for aqueous species and only if the option (thermo$opt$water) for calculations of properties using water is set to IAPWS. On the other hand, if the water option is ‘SUPCRT’ (the default), E and kT can be calculated for water but not for aqueous species. (This is not an inherent limitation in either formulation, but it is just a matter of implementation.))

Depending on the units currently defined (E.units) the values of G, H, S and Cp are returned using calories or Joules as the unit of energy, but only if convert is TRUE. Likewise, the input values of T and P are interpreted to have the units specified through T.units and P.units, but setting convert to FALSE forces subcrt to treat them as Kelvin and bar, respectively.

A chemical reaction is defined if coeff is given. In this mode the standard molal properties of species are summed according to the stoichiometric coefficients, where negative values denote reactants. Reactions that do not conserve elements are permitted; subcrt prints the missing composition needed to balance the reaction and produces a warning but computes anyway. Alternatively, if the basis species of a system were previously defined, and if the species being considered are within the compositional range of the basis species, an unbalanced reaction given in the arguments to subcrt will be balanced automatically, without altering the coefficients on the species identified in the arguments (unless perhaps they correspond to basis species), and without a warning. However, if a reaction is unbalanced and action.unbalanced is set to NULL, no warnings are generated and no attempt is made to balance the reaction.

Minerals with phase transitions (denoted by having states ‘cr1’, ‘cr2’ etc.) can be defined generically by ‘cr’ in the state argument. As of CHNOSZ-0.9-6, to consider phase transitions the species must be character, not numeric. subcrt in this case simultaneously calculates the requested properties of all the phases of each such mineral (phase species) and, using the values of the transition temperatures calculated from those at P = 1 bar given in the thermodynamic database together with functions of the entropies and volumes of transitions (see dPdTtr), determines the
stable phase of the mineral at any grid point and substitutes the properties of this phase at that point for further calculations. If individual phase species of minerals are specified (by ‘cr1’, ‘cr2’ etc. in state), and exceed.Ttr is FALSE (the default), the Gibbs energies of these minerals are assigned values of NA at conditions beyond their transition temperature, where another of the phases is stable. If you set exceed.Ttr to TRUE while investigating the properties of phases of minerals identified generically (by ‘cr’), the function will identify the stable phase on the basis not of the transition temperatures but of the calculated Gibbs energies of the phase species. This is not generally advised, since the computed standard molal properties of a phase species lose their physical meaning beyond the transition temperatures of the phase.

If logact is provided, the chemical affinities of reactions are calculated. logact indicates the logarithms of activities (fugacities for gases) of species in the reaction; if there are fewer values of logact than number of species those values are repeated as necessary. If the reaction was unbalanced to start, the logarithms of activities of any basis species added to the reaction are taken from the current definition of the basis species. Columns appended to the output are logQ for the log10 of the activity product of the reaction, and A for the chemical affinity, in the units set by E.units. Note that affinity provides related functionality but is geared toward the properties of formation reactions of species from the basis species and can be performed in more dimensions. Calculations of chemical affinity in subcrt can be performed for any reaction of interest; however, they are currently limited to constant values of the logarithms of activities of species in the reactions, and hence of logQ, across the computational range.

If IS is set to a single value other than zero, nonideal is used to calculate the apparent properties (G, H, S and Cp) of charged aqueous species at the given ionic strength. To perform calculations at a single P and T and for multiple values of ionic strength, supply these values in IS. Calculations can also be performed on a P-IS, T-IS or P,T-IS grid. Values of logK of reactions calculated for IS not equal to zero are consistent with the apparent Gibbs energies of the charged aqueous species.

subcrt is modeled after the functionality of the SUPCRT92 package (Johnson et al., 1992). Certain features of SUPCRT92 are not available here, for example, calculations as a function of density of H2O instead of pressure, or calculations of temperatures of univariant curves (i.e. for which logK is zero), or calculations of the molar volumes of quartz and coesite as a function of temperature (taken here to be constant). The informative messages produced by SUPCRT92 when temperature or pressure limits of the equations of state are exceeded generally are not reproduced here. However, NAs may be produced in the output of subcrt if the requisite thermodynamic or electrostatic properties of water can not be calculated at given conditions. Specifically, NAs are produced for calculations at ‘Psat’ when the temperature exceeds the critical temperature of H2O.

For calculations below 273.16 K, the pressure should be set to 1, as Psat is not defined in these conditions.

Value

For subcrt, a list of length two or three. If the properties of a reaction were calculated, the first element of the list (named ‘reaction’) contains a dataframe with the reaction parameters; the second element, named ‘out’, is a dataframe containing the calculated properties. Otherwise, the properties of species (not reactions) are returned: the first element, named ‘species’, contains a dataframe with the species identification; the second element, named ‘out’, is itself a list, each element of which is a dataframe of properties for a given species. If minerals with phase transitions are present, a third element (a dataframe) in the list indicates for all such minerals the stable phase at each grid point.
Warning

Comparison of the output of subcrt with that of SUPCRT92 indicates the two give similar values of properties for neutral aqueous species up to 1000 °C and 5000 bar. Changes were made in CHNOSZ 0.9-9 to improve the calculation of the $g$- and $f$-functions (Shock et al., 1992) for the partial derivatives of the omega parameter which are used by the hkf function, so thermodynamic properties of charged aqueous species over the temperature range 0 to 1000 °C are now mostly consistent with SUPCRT92. Note, however, that while SUPCRT92 limits calculations at Psat to 350 °C (“beyond range of applicability of aqueous species equations”), there is no corresponding limit in place in subcrt (or hkf), so that inapplicable calculations will be performed up to the critical temperature (373.917 °C) without any warning. It is probably for this reason that there is a noticeable jump in the value of logK at Psat shown in the one of the examples (demos("NaCl")).

A known bug is misidentification of the stable polymorph of some minerals at high temperature; an example of this bug is shown in the stopifnot at the end of the skarn example below.

References


See Also

demos("nonideal") runs an example using the IS argument. info can be used to find species in the thermodynamic database. makeup is used by subcrt for parsing formulas to check mass balance of reactions.

Examples

```r
## properties of species
subcrt("water")
# calculating at different temperatures
subcrt("water", T=seq(0, 100, 10))
# calculating at even increments
subcrt("water", T=seq(500, 1000, length.out=10),
    P=seq(5000, 10000, length.out=10))
```
calculating on a temperature-pressure grid
subcrt("water", T=c(500, 1000), P=c(5000, 10000), grid="P")

# to calculate only selected properties
subcrt("water", property=c("G", "E"))

# the properties of multiple species
subcrt(c("glucose", "ethanol", "CO2"))

# properties of reactions
subcrt(c("H2O", "H+", "K-feldspar", "kaolinite", "K+", "SiO2"),
        c(-1, -2, -2, 1, 2, 4))
subcrt(c("glucose", "ethanol", "CO2"), c(-1, 2, 2))

# to specify the states
subcrt(c("glucose", "ethanol", "CO2"), c(-1, 2, 2), c("aq", "aq", "gas"))

auto balancing reactions
# the basis species must first be defined
basis(c("CO2", "H2O", "NH3", "H2S", "O2"))
subcrt(c("glucose", "ethanol"), c(-1, 3))

# a bug in CHNOSZ <0.9 caused the following
# to initiate an infinite loop
basis(c("H2O", "H2S", "O2", "H+"))
subcrt(c("HS-", "SO4-2"), c(-1, 1))

# note the next one is a non-reaction
# (products same as reactants)
subcrt("H2O", 1)

# but this one auto-balances into a reaction
# (water to steam)
subcrt("H2O", 1, "gas")

# properties of a species and a formation
# reaction for that species
subcrt("C2H5OH")  # species
basis("CHNOS")
subcrt("C2H5OH", 1)  # reaction

# properties of mineral phases
# properties of phase species
subcrt(c("pyrrhotite", "pyrrhotite"), state=c("cr1", "cr2"))

# properties of the stable phases
subcrt("pyrrhotite")

# phase transitions in a reaction
subcrt(c("pyrite", "pyrrhotite", "H2O", "H2S", "O2"), c(-1, 1, -1, 1, 0.5))

# these produce messages about problems with the calculation
# Psat is not defined above the critical point
subcrt("alanine", T=seq(0, 5000, by=1000))

# above the T, P limits for the H2O equations of state
subcrt("alanine", T=c(2250, 2251), P=c(30000, 30001), grid="T")

# heat capacity of Fe(cr)
# compare calculated values of heat capacity with
# values from Robie and Hemingway, 1995, (from which
# the parameters in the database were derived)
# set the units
# we set pressure here otherwise subcrt goes for P_sat (saturation vapor pressure of H2O above 100 degrees C) which can not be calculated above the critical point of H2O (~647 K)
s <- subcrt("Fe", T=seq(300, 1800, 20), P=1)
    xlab=axis.label("T"), ylab=axis.label("Cp"))

# add points from RH95
RH95 <- read.csv(system.file("extdata/cpetc/RH95.csv", package="CHNOSZ"))
points(RH95[,1], RH95[,2])
title(main=paste("Heat capacity of Fe(cr)\n", 
    "(points - Robie and Hemingway, 1995)"))

# reset the units to default values
T.units("C")
E.units("cal")

## Skarn example after Johnson et al., 1992
P <- seq(500, 5000, 500)
# this is like the temperature specification used in the example by Johnson et al., 1992
# we use this one to avoid warnings at 0 deg C, 5000 bar
T <- seq(0, 1000, 100)

# the results are not identical to SUPCRT92, at least because CHNOSZ doesn't have volume changes for quartz, and has updated parameters for species e.g. Cu+ from Shock et al., 1997
# the following is to help detect unintended changes to the code across revisions; the code *should* be fixed sometime so that the last 1 becomes a 3
stopifnot(all.equal(s$state$chalcopyrite[1:11],
    c(1, 1, 1, 1, 1, 1, 1, 2, 3, 3, 3)))

## Standard Gibbs energy of reactions with HCN and formaldehyde, after Schulte and Shock, 1995 Fig. 1
rxn1 <- subcrt(c("formaldehyde","HCN","H2O","glycolic acid","NH3"),
    c(-1,-1,-2,1,1),P=300)
rxn2 <- subcrt(c("formaldehyde","HCN","H2O","glycine"),
    c(-1,-1,-1,1),P=300)
plot(x=rxn1$sout$T,rxn1$sout$G/1000,type="l",ylim=c(-40,-10),
    xlab=axis.label("T"),ylab=axis.label("DG0r","k"))
lines(rxn1$sout$T,rxn2$sout$G/1000)

# write the reactions on the plot
text(150, -14, describe.reaction(rxn1$reaction, iname=c(1,2,4)))
text(200, -35, describe.reaction(rxn2$reaction, iname=c(1,2)))
title(main=paste("Standard Gibbs energy of reactions", 
    "after Schulte and Shock, 1995",sep="\n"))
## Calculation of chemical affinities

# after LaRowe and Helgeson, 2007, Fig. 3 (a): reduction of nicotinamide
# adenine dinucleotide (NAD) coupled to oxidation of glucose
# list the available NAD species

info("NAD ")

# oxidation of glucose (C6H12O6)
basis(c("glucose", "H2O", "NH3", "CO2", "H+"), c(-3, 0, 999, -3, -7))
s <- subcrt(c("NAD(ox)-", "NAD(red)-2"), c(-12, 12), logact=c(0, 0), T=T)
# LH07's diagrams are shown per mole of electron (24 e- per 12 NAD)
A <- s$out$A/24/1000

plot(x=T, y=A, xlab=axis.label("T"), ylab=axis.label("A", prefix="k"), type="l")
text("NAD(ox)-/NAD(red)-2 = 1", x=53, y=median(A), srt=21)
# different activity ratio
s <- subcrt(c("NAD(ox)-", "NAD(red)-2"), c(-12, 12), logact=c(1, 0), T=T)
A <- s$out$A/24/1000
lines(x=T, y=A)
text("NAD(ox)-/NAD(red)-2 = 10", x=55, y=median(A), srt=24)
# different activity ratio
s <- subcrt(c("NAD(ox)-", "NAD(red)-2"), c(-12, 12), logact=c(0, 1), T=T)
A <- s$out$A/24/1000
lines(x=T, y=A)
text("NAD(ox)-/NAD(red)-2 = 0.1", x=52, y=median(A), srt=18)
# print the reaction and chemical conditions on the plot
text(0, 5.3, describe.reaction(s$s$reaction, iname=c(1, 2)), adj=0)
text(0, 5.1, describe.basis(oneline=TRUE, ibasis=c(1, 2, 4, 5)), adj=0)
# label the plot
title(main=paste("Reduction of NAD coupled to oxidation of glucose", 
"after LaRowe and Helgeson, 2007", sep="\n"))

## Subzero (degrees C) calculations

# uncomment the following to try IAPWS95 instead of SUPCRT92
# thermoOpt$water <= "IAPWS95"
# the limit for H2O92D.f (from SUPCRT92) is currently -20 deg C
# but we go to -30 knowing properties will become NA
sb <- subcrt(c("H2O", "Na++"), T=seq(-30, 10), P=1)$out
# start plot with extra room on right
par(mar=c(5, 4, 4, 4))
# plot G
plot(sb$water$T, sb$water$G, ylim=c(-63000, -56000), xlab=axis.label("T"), ylab=axis.label("DG0"))
points(sb$Na++$T, sb$Na++$G, pch=2)
# add Cp
# change y-axis
par("usr"=c(par("usr")[1:2], -100, 25))
axis(4)
mtext(axis.label("Cp"), side=4, line=3)
points(sb$water$T, sb$water$Cp, pch=16)
points(sb$Na++$T, sb$Na++$Cp, pch=17)
legend("topleft", pch=c(16, 1, 17, 2), legend=c("H2O Cp", "H2O G", "Na++ Cp", "Na++ G"))
H2O <- expr.species("H2O")
Na <- expr.species("Na++")
degC <- expr.units("C")
swap.basis

Swap Basis Species

Description

Swap the basis species defining a chemical system. One basis species is replaced by a new one with a different chemical formula.

Usage

```r
basis.matrix(basis = get("thermo")$basis)
element.mu(basis = get("thermo")$basis, T = 25)
basis.logact(emu, basis = get("thermo")$basis, T = 25)
swap.basis(species, species2)
```

Arguments

- `basis`: dataframe, a basis definition
- `T`: numeric, temperature in Kelvin
- `emu`: numeric, chemical potentials of elements
- `species`: character, names or formulas of species, or numeric, indices of species
- `species2`: character or numeric, a species to swap in to the basis definition

Details

To change the basis definition, specify the names or formulas of the new basis species in the first argument. When the basis definition is changed, any species of interest that were present are deleted, unless the new basis definition has exactly the same elements as before. In that case, the species are kept and the activities of the new basis species are set so that the chemical potentials of the elements at 25 °C and 1 bar are unchanged.

See Also

`basis` for defining the basis species, a prerequisite for swapping. Tests (using `test_that`) showing common error conditions are in `inst/tests`.

Examples

```r
## swapping basis species
# start with a preset basis definition
b1 <- basis("CHNOS+")
# swap H2(aq) for O2(gas)
(b2 <- swap.basis("O2", "H2"))
```
# the logarithm of activity calculated for H2
# is equal to the one calculated from the equilibrium constant
# for H2O = H2 + 0.5O2
logK <- subcrt(c("oxygen","H2","H2O"), c(-0.5,-1,1), T=25)$out$logK
# the equilibrium value of logaH2
# (for logaH2O = 0 and logfO2 = -80)
(logaH2 <- -logK + 40)
stopifnot(all.equal(logaH2, b2$logact[5]))
# put O2 back in
b3 <- swap.basis("H2", "oxygen")
# we have returned to starting point
stopifnot(all.equal(b1$logact, b3$logact))

## demonstrating the interconvertibility between
## chemical potentials of elements and logarithms
## of activities of basis species at high temperature
basis("CHNOS+")
b11 <- basis$logact
emu <- element.mu(T=100)
b12 <- basis.logact(emu, T=100)
# note that basis.logact produces a named array
stopifnot(all.equal(b11, as.numeric(b12)))

## swapping basis species while species are defined
## and using numeric species indices
basis("MgCHNOPS+")
# load species whose names contain "ATP"
species(info.approx("ATP "))
# swap in CO2(g) for CO2(aq)
icO2g <- info("CO2", "gas")
swap.basis("CO2", iCO2g)
a1 <- affinity()
# swap in CH4(g) for CO2(g)
ich4g <- info("CH4", "gas")
swap.basis(iCO2g, ich4g)
a2 <- affinity()
# the equilibrium fugacity of CH4 is *very* low
# swap in CO2(aq) for CH4(g)
icO2a <- info("CO2", "aq")
swap.basis(ich4g, iCO2a)
a3 <- affinity()
# swapping the basis species didn't affect the affinities
# of the formation reactions of the species, since
# the chemical potentials of the elements were unchanged
stopifnot(all.equal(a1$values, a2$values))
stopifnot(all.equal(a1$values, a3$values))
Description

Read data from NCBI taxonomy files, traverse taxonomic ranks, get scientific names of taxonomic nodes.

Usage

getnodes(taxdir)
getranks(id, taxdir, nodes=NULL)
parent(id, taxdir, rank=NULL, nodes=NULL)
allparents(id, taxdir, nodes=NULL)
getnames(taxdir)
sclename(id, taxdir, names=NULL)

Arguments

taxdir character, directory where the taxonomy files are kept.
id numeric, taxonomic ID(s) of the nodes of interest.
nodes dataframe, output from getnodes (optional).
rank character, name of the taxonomic rank of interest.
names dataframe, output from getnames (optional).

Details

These functions provide a convenient way to read data from NCBI taxonomy files (i.e., the contents of taxdump.tar.gz, which can be downloaded from ftp://ftp.ncbi.nih.gov/pub/taxonomy/).

The taxdir argument is used to specify the directory where the nodes.dmp and names.dmp files are located. getnodes and getnames read these files into data frames. getrank returns the rank (species, genus, etc) of the node with the given taxonomic id. parent returns the taxonomic ID of the next-lowest node below that specified by the id in the argument, unless rank is supplied, in which case the function descends the tree until a node with that rank is found. allparents returns all the taxonomic IDs of all nodes between that specified by id and the root of the tree, inclusive. sname returns the scientific name of the node with the given id.

The id argument can be of length greater than 1 except for allparents. If getrank, parent, allparents or sclename need to be called repeatedly, the operation can be hastened by supplying the output of getnodes in the nodes argument and/or the output of getnames in the names argument.

Examples

```r
## get information about Homo sapiens from the
## packaged taxonomy files
taxdir <- system.file("extdata/taxonomy",package="CHNOSZ")
# H. sapiens' taxonomic id
idl <- 9006
# that is a species
getranks(idl,taxdir)
# the next step up the taxonomy
id2 <- parent(idl,taxdir)
```
print(id2)
# that is a genus
getrank(id2,taxdir)
# that genus is "Homo"
sciname(id2,taxdir)
# we can ask what phylum is it part of?
id3 <- parent(id1,taxdir,"phylum")
# answer: "Chordata"
sciname(id3,taxdir)
# H. sapiens' complete taxonomy
id4 <- allparents(id1,taxdir)
sciname(id4,taxdir)

## the names of the organisms in the supplied taxonomy files
taxdir <- system.file("extdata/taxonomy",package="CHNOSZ")
id5 <- c(83333,4932,9606,186497,243232)
sciname(id5,taxdir)
# these are not all species, though
# (those with "no rank" are something like strains,
# e.g. Escherichia coli K-12)
getrank(id5,taxdir)
# find the species for each of these
id6 <- sapply(id5,function(x) parent(x,taxdir=taxdir,rank="species"))
stopifnot(unique(getrank(id6,taxdir))=="species")
# note that the K-12 is dropped
sciname(id6,taxdir)

## the complete nodes.dmp and names.dmp files are quite large,
## so it helps to store them in memory when performing multiple queries
## (this doesn't have a noticeable speed-up for the small files
## we use in this example)
taxdir <- system.file("extdata/taxonomy",package="CHNOSZ")
nodes <- getnodes(taxdir=taxdir)
# all of the node ids in this file
id7 <- nodes$id
# all of the non-leaf nodes
id8 <- unique(parent(id7,nodes=nodes))
names <- getnames(taxdir=taxdir)
sciname(id8,names=names)

---

**thermo**

*Thermodynamic Database and System Definition*

**Description**

The core data files provided with CHNOSZ are in the `data` directory of the package. These `*.csv` files are used to build the `thermo` data object on loading the package. Additional (extra) data files, supporting the examples and vignettes, are documented separately at `extdata`

The `thermo` object holds the thermodynamic database of properties of species, some thermodynamic constants and operational parameters for functions in CHNOSZ, the properties of elements,
references to literature sources of thermodynamic data, compositions of chemical activity buffers, and amino acid compositions of proteins. The \texttt{thermo} object also holds intermediate data used in calculations, in particular the definitions of basis species and species of interest input by the user, and the calculated properties of \texttt{water} so that subsequent calculations at the same temperature-pressure conditions can be accelerated.

The \texttt{thermo} object is a \texttt{list} composed of \texttt{data.frames} or lists each representing a class of data. The object is created in an environment named \texttt{thermo}; see \texttt{sideeffects} for details. It is created upon loading the package, through a call to \texttt{data(thermo)} in \texttt{.onAttach}. At any time, the user can restore the data object to its initial state by calling \texttt{data(thermo)}. This is sometimes a useful command to use during an interactive session, when previous definitions of basis species and species of interest are longer desired.

The function \texttt{add.obigt} is available to update the thermodynamic database in use in a running session. For example, one can run \texttt{add.obigt("mydata.csv")} after loading the package, and the data in that file will be added to the database. The format of this file must be the same as the \texttt{OBIGT.csv} file provided with CHNOSZ. Although changes made using \texttt{add.obigt} are lost when the current \texttt{R} session is closed, the data can always be restored the next time as long as the user has the \texttt{mydata.csv} (or other) file available.

The first example below shows how to find the installation locations of \texttt{OBIGT.csv} and other *.csv files. Making changes to these files is not recommended, because incompatible changes can leave the package unusable; also, the files will be overwritten whenever the package is installed (or updated). Instead, use these files as templates for creating your own database files.

**Usage**

\texttt{data(thermo)}

**Format**

The items in the \texttt{thermo} object are documented below.

- \texttt{thermo$opt} List of operational parameters

\begin{verbatim}
Tr numeric Reference temperature (K)
Pr numeric Reference pressure (bar)
Theta numeric \(\Theta\) in the revised HKF equations of state (K)
Psi numeric \(\Psi\) in the revised HKF equations of state (bar)
cutoff numeric Cutoff below which values are taken to be zero (see \texttt{makeup})
E.units character The user’s units of energy (‘\texttt{cal}’ (default) or ‘\texttt{J}’) 
T.units character The user’s units of temperature (‘\texttt{C}’ (default) or ‘\texttt{K}’) 
P.units character The user’s units of pressure (‘\texttt{bar}’ (default) or ‘\texttt{MPa}’) 
state character The default physical state for searching species (‘\texttt{aq}’ by default)
water character Computational option for properties of water (‘\texttt{SUPCRT}’ (default) or ‘\texttt{IAPWS}’) 
online logical Allow online searches of protein composition? Default (\texttt{NA}) is to ask the user.
G.tol numeric Absolute difference between tabulated and calculated value of G above which \texttt{checkGHS} produces a message
Cp.tol numeric Absolute difference between tabulated and calculated value of Cp above which \texttt{checkEOS} produces a message
V.tol numeric Absolute difference between tabulated and calculated value of V above which \texttt{checkEOS} produces a message
\end{verbatim}
thermo$element Dataframe containing the thermodynamic properties of elements taken from Cox et al., 1989 and Wagman et al., 1982. The standard molal entropy ($S(Z)$) at 25 °C and 1 bar for the element of charge ($Z$) was calculated from $S(H_2,g) + 2S(Z) = 2S(H^+)$, where the standard molal entropies of $H_2,g$ and $H^+$ were taken from Cox et al., 1989. The mass of $Z$ is taken to be zero. Accessing this data frame using mass or entropy will select the first entry found for a given element; i.e., values from Wagman et al., 1982 will only be retrieved if the properties of the element are not found from Cox et al., 1989.

<table>
<thead>
<tr>
<th>element character</th>
<th>Symbol of element</th>
</tr>
</thead>
<tbody>
<tr>
<td>state character</td>
<td>Stable state of element at 25 °C and 1 bar</td>
</tr>
<tr>
<td>source character</td>
<td>Source of data</td>
</tr>
<tr>
<td>mass numeric</td>
<td>Mass of element (in natural isotopic distribution; referenced to a mass of 12 for $^{12}$C)</td>
</tr>
<tr>
<td>s numeric</td>
<td>Entropy of the compound of the element in its stable state at 25 °C and 1 bar (cal K$^{-1}$ mol$^{-1}$)</td>
</tr>
<tr>
<td>n numeric</td>
<td>Number of atoms of the element in its stable compound at 25 °C and 1 bar</td>
</tr>
</tbody>
</table>

thermo$obigt

This dataframe is a thermodynamic database of standard molal thermodynamic properties and equations of state parameters of species. OBIGT is an acronym for ‘Organobiogeotherm’, which refers to a software package produced by Harold C. Helgeson and coworkers at the Laboratory of Theoretical Geochemistry and Biogeochemistry at the University of California, Berkeley. (There may be an additional meaning for the acronym: “One BIG Table” of thermodynamic data.)

As of CHNOSZ version 0.7, the data in OBIGT.csv represent 179 minerals, 16 gases, and 294 aqueous (largely inorganic) species taken from the data file included in the SUPCRT92 distribution (Johnson et al., 1992), an additional 14 minerals, 6 gases, and 1049 aqueous organic and inorganic species from the SLOP98.DAT file (Shock et al., 1998), and approximately 50 other minerals, 175 crystalline organic and biochemical species, 220 organic gases, 300 organic liquids, 650 aqueous inorganic, organic, and biochemical species, and 40 organic groups taken from the recent literature. Each entry is referenced to one or two literature sources listed in thermo$refs. Use browse.refs to display the references in a browser window.

Note the following additional modifications:

- Use corrected values of $a_2$ and $a_4$ for [-CH2NH2] (were incorrectly listed as zero in Table 6 of Dick et al., 2006).
- The standard molal thermodynamic properties and equations of state parameters of the aqueous electron are zero except for the standard molal entropy at 25 °C and 1 bar, which is the opposite of that for the element of charge ($Z$, see above).
- The properties and parameters of some reference unfolded proteins used by Dick et al., 2006 are included here. Their names have dashes, instead of underscores, so that they are not confused with proteins whose properties are generated at runtime.
- The standard molal Gibbs energies and enthalpies of formation of the elements and entropies at 25 °C and 1 bar of aqueous metal-amino acid (alanate or glycinate) complexes reported by Shock and Koretsky, 1995 were recalculated by adding to their values the differences in the corresponding properties between the values for aqueous alanate and glycinate used by Shock and Koretsky, 1995, and those used by Amend and Helgeson, 1997b and Dick et al., 2006.
The standard molal properties and equations-of-state parameters of four phase species (see below) of Fe(cr) were generated from heat capacity data given by Robie and Hemingway, 1995.

These modifications are indicated in OBIGT.csv by having ‘CHNOSZ’ as one of the sources of data. Note also that some data appearing in the SLOP98.DAT file (Shock et al., 1998) were corrected or modified as noted in that file, and are indicated in OBIGT.csv by having ‘SLOP98’ as one of the sources of data.

In order to represent thermodynamic data for minerals with phase transitions, the different phases of these minerals are represented as phase species that have states denoted by ‘cr1’, ‘cr2’, etc. The standard molar thermodynamic properties at 25 °C and 1 bar (T_r and P_r) of the ‘cr2’ phase species of minerals were generated by first calculating those of the ‘cr1’ phase species at the transition temperature (T_tr) and 1 bar then taking account of the volume and entropy of transition (the latter can be retrieved by combining the former with the Clausius-Clapeyron equation and values of (dP/dT) of transitions taken from the SUPCRT92 data file) to calculate the standard molar enthalpy of the ‘cr2’ phase species at T_tr. The standard molar properties of the ‘cr2’ phase species at T_tr and 1 bar calculated in this manner were combined with the equations-of-state parameters of the species to generate values of the standard molar properties at 25 °C and 1 bar. This process was repeated as necessary to generate the standard molar properties of phase species represented by ‘cr3’ and ‘cr4’, referencing at each iteration the previously calculated values of the standard molar properties of the lower-temperature phase species (i.e., ‘cr2’ and ‘cr3’). A consequence of tabulating the standard molar thermodynamic properties of the phase species is that the values of (dP/dT) and ∆H° of phase transitions can be calculated using the equations of state and therefore do not need to be stored in the thermodynamic database. However, the transition temperatures (T_tr) generally can not be assessed by comparing the Gibbs energies of phase species and are tabulated in the database.

The identification of species and their standard molal thermodynamic properties at 25 °C and 1 bar are located in the first 12 columns of thermo$obigt:

<table>
<thead>
<tr>
<th>name</th>
<th>character</th>
<th>Species name</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbrv</td>
<td>character</td>
<td>Species abbreviation</td>
</tr>
<tr>
<td>formula</td>
<td>character</td>
<td>Species formula</td>
</tr>
<tr>
<td>state</td>
<td>character</td>
<td>Physical state</td>
</tr>
<tr>
<td>ref1</td>
<td>character</td>
<td>Primary source</td>
</tr>
<tr>
<td>ref2</td>
<td>character</td>
<td>Secondary source</td>
</tr>
<tr>
<td>date</td>
<td>character</td>
<td>Date of data entry (formatted as in SUPCRT92)</td>
</tr>
<tr>
<td>G</td>
<td>numeric</td>
<td>Standard molal Gibbs energy of formation from the elements (cal mol$^{-1}$)</td>
</tr>
<tr>
<td>H</td>
<td>numeric</td>
<td>Standard molal enthalpy of formation from the elements (cal mol$^{-1}$)</td>
</tr>
<tr>
<td>S</td>
<td>numeric</td>
<td>Standard molal entropy (cal mol$^{-1}$ K$^{-1}$)</td>
</tr>
<tr>
<td>Cp</td>
<td>numeric</td>
<td>Standard molal isobaric heat capacity (cal mol$^{-1}$ K$^{-1}$)</td>
</tr>
<tr>
<td>V</td>
<td>numeric</td>
<td>Standard molal volume (cm$^3$ mol$^{-1}$)</td>
</tr>
</tbody>
</table>

The meanings of the remaining columns depend on the physical state of a particular species. If it is aqueous, the values in these columns represent parameters in the revised HKF equations of state (see hkf), otherwise they denote parameters in a general equations for crystalline,
gas and liquid species (see \textit{cgl}). The names of these columns are compounded from those of the parameters in each of the equations of state (for example, column 13 is named \textit{a1.a}). Scaling of the values by orders of magnitude is adopted for some of the parameters, following common usage in the literature.

Columns 13-20 for aqueous species (parameters in the revised HKF equations of state):

\begin{verbatim}
  a1 numeric  $a_1 \times 10$ (cal mol$^{-1}$ bar$^{-1}$)
  a2 numeric  $a_2 \times 10^{-2}$ (cal mol$^{-1}$)
  a3 numeric  $a_3$ (cal K mol$^{-1}$ bar$^{-1}$)
  a4 numeric  $a_4 \times 10^{-4}$ (cal mol$^{-1}$ K)
  c1 numeric  $c_1$ (cal mol$^{-1}$ K$^{-1}$)
  c2 numeric  $c_2 \times 10^{-4}$ (cal mol$^{-1}$ K)
  omega numeric $\omega \times 10^{-5}$ (cal mol$^{-1}$)
  Z numeric Charge
\end{verbatim}

Columns 13-20 for crystalline, gas and liquid species ($C_p = a + bT + cT^{-2} + dT^{-0.5} + eT^2 + fT^\lambda$).

\begin{verbatim}
  a numeric  $a$ (cal K$^{-1}$ mol$^{-1}$)
  b numeric  $b \times 10^3$ (cal K$^{-2}$ mol$^{-1}$)
  c numeric  $c \times 10^{-5}$ (cal K mol$^{-1}$)
  d numeric  $d$ (cal K$^{-0.5}$ mol$^{-1}$)
  e numeric  $e \times 10^5$ (cal K$^{-3}$ mol$^{-1}$)
  f numeric  $f$ (cal K$^{-\lambda-1}$ mol$^{-1}$)
  lambda numeric $\lambda$ (exponent on the $f$ term)
  T numeric Temperature of phase transition or upper temperature limit of validity of extrapolation (K)
\end{verbatim}

• \texttt{thermo$\textbackslash$sourc}e Dataframe of references to sources of thermodynamic data.

\begin{verbatim}
  key character  Source key
  author character  Author(s)
  year character  Year
  citation character  Citation (journal title, volume, and article number or pages; or book or report title)
  URL character  URL
\end{verbatim}

• \texttt{thermo$\textbackslash$buffers} Dataframe which contains definitions of buffers of chemical activity. Each named buffer can be composed of one or more species, which may include any species in the thermodynamic database and/or any protein. The calculations provided by \texttt{buffer} do not take into account phase transitions of minerals, so individual phase species of such minerals must be specified in the buffers.

\begin{verbatim}
  name character  Name of buffer
  species character  Name of species
  state character  Physical state of species
  logact numeric  Logarithm of activity (fugacity for gases)
\end{verbatim}
- **thermoprotein** Data frame of amino acid compositions of selected proteins. Many of the compositions were taken from the SWISS-PROT/UniProt online database (Boeckmann et al., 2003) and the protein and organism names usually follow the conventions adopted there. In some cases different isoforms of proteins are identified using modifications of the protein names; for example, 'M005.M' and M005.N proteins of 'YEAST' denote the mitochondrial and nuclear isoforms of this protein. See `iprotein` to search this data frame by protein name, and other functions to work with the amino acid compositions.

| protein | character | Identification of protein |
| organism | character | Identification of organism |
| ref | character | Reference key for source of compositional data |
| abbrv | character | Abbreviation or other ID for protein |
| chains | numeric | Number of polypeptide chains in the protein |
| Ala...Tyr | numeric | Number of each amino acid in the protein |

- **thermo$groups** This is a dataframe with 22 columns for the amino acid sidechain, backbone and protein backbone groups ([Ala],[Tyr],[AABB],[UPBB]) whose rows correspond to the elements C, H, N, O, S. It is used to quickly calculate the chemical formulas of proteins that are selected using the `iprotein` argument in `affinity`.

- **thermo$basis** Initially NULL, reserved for a dataframe written by `basis` upon definition of the basis species. The number of rows of this dataframe is equal to the number of columns in "..." (one for each element).

| ... | numeric | One or more columns of stoichiometric coefficients of elements in the basis species |
| ispecies | numeric | Rownumber of basis species in `thermo$obigt` |
| logact | numeric | Logarithm of activity or fugacity of basis species |
| state | character | Physical state of basis species |

- **thermo$species** Initially NULL, reserved for a dataframe generated by `species` to define the species of interest. The number of columns in "..." is equal to the number of basis species (i.e., rows of `thermo$basis`).

| ... | numeric | One or more columns of stoichiometric coefficients of basis species in the species of interest |
| ispecies | numeric | Rownumber of species in `thermo$obigt` |
| logact | numeric | Logarithm of activity or fugacity of species |
| state | character | Physical state of species |
| name | character | Name of species |

- **thermo$water** The properties calculated with `water` at multiple T, P points (minimum of 26) are stored here so that repeated calculations at the same conditions can be done more quickly.

- **thermo$Psat** The values of Psat calculated with `water.SUPCRT` at multiple T points (minimum of 26) are stored here.

- **thermo$water2** The properties calculated with `water.SUPCRT` at multiple T, P points (minimum of 26) are stored here.
References


See Also

add.obigt for thermodynamic data from local .csv files.

Examples

```r
## where are OBIGT.csv and the other data
## files on your installation?
system.file("data",package="CHNOSZ")

## exploring thermo$obigt
# what physical states there are
unique(thermo$obigt$state)
# formulas of ten species at random
n <- nrow(thermo$obigt)
thermo$obigt$formula[runif(10)*n]
```
Mass Transfer Calculations

Description

Simulate a mass transfer process such as mineral weathering or sequential formation of proteins.

Usage

```r
transfer(nsteps = 500, dmode = "coupled", devmax = 0.1, plot = NULL,
         ibalance = 1, fmode = "one", buffers = NULL, alphamax = -2,
         alphastart = -10, T = 25, P = "Psat", do.title = TRUE, beta = 0)
draw.transfer(t, ylim = c(-10, 1), ylimbasis = c(-12, -2),
               logprogress = FALSE)
feldspar(which = "closed", plot.it = FALSE)
apc(which = "open", basis = "CO2", plot.it = FALSE)
```

Arguments

- `nsteps` numeric, maximum number of steps to run simulation.
- `dmode` character, destruction mode.
- `devmax` numeric, maximum deviation of logarithm of activity of basis species in any step.
- `plot` numeric, which basis species to use as plotting and coupling variables.
- `ibalance` numeric, which basis species is the primary conservant.
- `fmode` character, formation mode.
- `buffers` list, basis species to be buffered during the simulation.
- `alphamax` numeric, maximum value of the destruction exponent.
- `alphastart` numeric, initial value of the destruction exponent.
- `T` numeric, temperature.
- `P` numeric, pressure.
- `do.title` logical, plot a title?
- `beta` numeric, alpha + beta = buffer transfer exponent.
- `t` list, the output of `transfer`.
- `ylim` numeric, y-axis limits.
- `ylimbasis` numeric, y-axis limits for the logarithms of activities of basis species.
- `logprogress` logical, put reaction progress on a logarithmic scale?
- `which` character, type of system to simulate.
- `plot.it` logical, summarize the results using `draw.transfer`?
- `basis` character, type of basis definition to use.
Details

The transfer function calculates a reaction path that is generated by incrementally reacting a starting composition into an aqueous system. Before calling this function, set up a system and define the starting material using \texttt{species}.

At each step, a small amount ($10^\alpha$) of the starting composition is provisionally reacted and a relatively more stable product may be formed. The amount of product formed is such that the activity of the primary conservant (the basis species given in \texttt{ibalance}) is not changed. The changes in the activities of the other basis species are calculated, and the process is iterated until \texttt{nsteps} is reached or the value of $\alpha$ is driven to a very low value (\texttt{logpresent}, which is a constant set in the code to $-50$).

If at a given step the most stable product is different from the one before, either the previous products are ignored (for \texttt{dmode} equal to ‘none’, i.e. an open system) or the reaction of the starting material is coupled to that of the existing products (for \texttt{dmode} equal to ‘coupled’, i.e. a closed system) through a secondary conservation constraint. The basis species that are candidates for the secondary conservation are identified in \texttt{iplot}.

The initial value of alpha is given by \texttt{alphastart}. After successful steps, the function increases the value of alpha by 1, and after failed steps decreases the value of alpha by 1. One condition that can lead to a failed step is that the logarithm of activity of any basis species changes by more than \texttt{devmax}. Therefore, throughout the simulation the value of $\alpha$ dynamically adjusts based on the \texttt{devmax} set by the user.

\texttt{buffers} is a list with elements \texttt{basis} indicating the basis species to be buffered and \texttt{buffer} naming the buffers to use for that basis species. If this argument is given, at each step the activity of the basis species in \texttt{in the buffer} is calculated. The difference between this activity and the current activity of the basis species in the system is then multiplied by 10 raised to the $(\alpha + \beta)$ and this quantity added to the current activity of the basis species in the system. As a result, the value of $\beta$ modifies the strength of the buffer relative to the incremental reaction progress.

\texttt{draw.transfer} is used to plot the logarithms of activities of basis species, and logarithms of activities (moles for solid species, molalities for aqueous species) of the minerals or proteins as a function of reaction progress, or logarithm of reaction progress if \texttt{logprogress} is set to \texttt{true}. The y-limits of the plots can be set using \texttt{ylim} and \texttt{ylimbasis}.

\texttt{feldspar} and \texttt{apc} encode examples for feldspar weathering and reactions among proteins in the anaphase-promoting complex of yeast.

Value

\texttt{transfer} returns a list containing information about the conditions at each step: \texttt{basis}, data frame of the logarithms of activities of basis species, \texttt{species}, data frame of the logarithms of activities (moles for solids) of species, \texttt{alphas}, numeric vector of the values of the destruction exponent, \texttt{dmodes}, character vector of the destruction mode, \texttt{istables}, numeric vector of the index of the most stable product, \texttt{myaffs}, list of the affinities of the formation reactions of species, \texttt{didwork}, logical vector indicating whether the steps succeeded or failed.

References

Examples

```r
## react potassium feldspar in a closed system
## after Steinmann et al., 1994 and Helgeson et al., 1969
basis(c("Al+3", "H4SiO4", "K+", "H2O", "H+", "O2"), c(0, -6, -6, 0, 0, 0))
species(c("K-feldspar", "muscovite", "pyrophylite", "kaolinite", "gibbsite"))
a <- affinity(H4SiO4=c(-6, -2), "K+"=c(-3, 8))
diagram(a, fill="heat")
basis("pH", 4)
species(1:5, c(-4, rep(-999, 4)))
tr <- transfer(550, dmode="coupled", plot=c(2, 3), devmax=0.2)
# plot the output from transfer
draw.transfer(tr)
# reset the plot layout
layout(matrix(11))

## can also run the calculation above with
## feldspar("closed")
## or an example for proteins with
## apc("closed")
```

util.affinity

**Functions to Work With Chemical Affinities**

**Description**

The “guts” of the affinity calculations: compute affinities or other thermodynamic properties of formation reactions of species on multidimensional arrays with dimensions corresponding to the ranges of intensive variables requested by the user; calculate affinities of ionization reactions of proteins.

**Usage**

```r
energy(what, vars, vals, lims, T=get("thermo")$opt$Tr, P="Psat", IS=0,
       sout=NULL, exceed.Ttr=FALSE, transect = FALSE)
energy.args(args)
A.ionization(iprotein, vars, vals, T=get("thermo")$opt$Tr,
P="Psat", pH=7, transect=FALSE)
```

**Arguments**

- `what`: character, name of property to calculate
- `vars`: character, names of variables over which to calculate a property
vals list of numeric, values for each variable
lims list of numeric, limits of the values for each variable
\( T \) numeric, temperature. Default is to take the temperature from thermo\$opt\$Tr, which corresponds to 25 °C
\( P \) numeric, pressure, or character "Psat" (default), which denotes 1 bar or the saturation vapor pressure of \( \text{H}_2\text{O} \) above 100 °C (see water)
\( \text{IS} \) numeric, ionic strength; default is 0 mol kg\(^{-1}\)
sout list, output from subcrt function
exceed.Ttr logical, allow subcrt to compute properties for phases beyond their transition temperature?
transect logical, perform calculations on a transect instead of a grid?
args list, defines the variables over which to calculate properties
iprotein numeric, rownumber in thermo\$protein
\( \text{pH} \) numeric, pH

Details

energy is the engine for the calculations of chemical affinity. Given \( n \) (which can be zero, one, or more) names of basis species and/or ‘\( T \)’, ‘\( P \)’, or ‘\( \text{IS} \)’ as the vars, it calculates the property given in what on an \( n \)-dimensional grid or transect for each of the values (vals) of the corresponding variable. The limits for each variable given in lims indicate the minimum and maximum value and, if a third value is supplied, the resolution, or number of points in the given dimension. If ‘\( T \)’, ‘\( P \)', and/or ‘\( \text{IS} \)’ are not among the vars, their constant values can be supplied in \( T \) (in Kelvin), \( P \) (in bar, or ‘Psat’), and \( \text{IS} \) (in mol kg\(^{-1}\)). sout, if provided, replaces the call to subcrt which can greatly speed up the calculations if this intermediate step is stored by other functions (e.g., transfer).

exceed.Ttr is passed to subcrt so that the properties of mineral phases beyond their transition temperatures can optionally be calculated.

The what argument of energy is analogous to the property argument of affinity.

energy.args is used by affinity to generate the argument list for energy. energy.args also has the job of converting ‘Eh’ to ‘pe’ as a function of temperature (see convert), and converting ‘pe’ and ‘pH’ to logarithms of activities of the electron and protein, respectively (i.e., negating the values).

In CHNOSZ version 0.9, energy gained a new argument ‘transect’ which is set to TRUE by energy.args when the length(s) of the variables is(are) greater than three. In this mode of operation, instead of performing the calculations on an \( n \)-dimensional grid, the affinities are calculated on an \( n \)-dimensional transect through chemical potential (possibly including \( T \) and/or \( P \) space.

A.ionization builds a list of values of \( A/2.303RT \) of the ionization reactions of proteins that are a function of \( T \), \( P \) and \( \text{pH} \) but are expanded to as many dimensions as defined in vars in order to be included by the calculations by energy. These calculations are invoked if proteins are in the species definition, and the basis species contain ‘\( \text{H}^+ \)’.
Value

For energy, a list the first element of which is sout (the results from subcrt) and the second element of which is a, which contains the calculated properties. The latter itself is a list, one element for each species of interest, which have dimensions that are the number of variables passed to the function.

For energy.args, a list with elements what, vars, vals, lims, T, P, IS that are appropriate for the corresponding arguments in energy.

See Also

In most cases, affinity is used interactively instead of these functions.

Examples

```r
basis("CHNOS")
species("acetic acid")
eargs <- energy.args(list(O2=c(-90, -60, 5), T=c(0, 100, 5)))
ea <- do.call(energy, eargs)
```

---

**Description**

Handle arguments referring to temperature, pressure, states, and equations of state.

**Usage**

```r
eos.args(eos, property = NULL, T = NULL, P = NULL)
TP.args(T = NULL, P = NULL)
state.args(state = NULL)
```

**Arguments**

- **eos** character, name of equation of state (one of ‘hkf’, ‘cgl’, ‘water’).
- **property** character, name(s) of thermodynamic properties.
- **T** numeric, temperature (K).
- **P** numeric, pressure (bar) (can also be character, ‘Psat’ in TP.args).
- **state** character, name(s) of states (e.g., ‘cr’, ‘aq’).
Details

The *args functions are used to normalize user-input arguments, which are case-insensitive. `eos.args` returns a list with elements named props, for all the properties available for the specified equations-of-state, prop for the lower-case version of property, and Prop, for the upper-case (of first letter) version of property. `eos.args` produces an error if one of the properties is not in the list of available properties. (See `water` and `subcrt` for the available properties for different species.) `TP.args` forces T and P to equal length. This function also looks for the keyword ‘Psat’ in the value of P and substitutes calculated values of the saturation vapor pressure (see `water`). `state.args` makes its argument lowercase, then transforms ‘a’, ‘c’, ‘g’, and ‘l’ to ‘aq’, ‘gas’, ‘cr’, and ‘liq’, respectively.

Value

A list is return by `eos.args` and `TP.args`, and character is returned by `state.args`.

Examples

```r
## argument processing
eoS.args("hkf",c("g","H","S","CP","V","kT","e"))
## produces an error because "Q" is not allowed in water
## Not run:
  eoS.args("hkf",c("G","H","S","Cp","V","kT","E","Q"))
## Not run:
  thermo$opt$water <- "IAPWS" # needed for p and n in next line
  eoS.args("water",c("p","u","cv","psat","rho","n","q","x","y","epsilon"))
  TP.args(c(273.15,373.15))
  TP.args(c(273.15,373.15),"Psat")
  TP.args(c(273.15,373.15),c(100,100,200,200))
  state.args(c("AQ","GAS"))
  state.args(c("a","l","liq"))
```
Arguments

- `l`: a list.
- `arr`: an array.
- `d`: numeric, what dimension to use.
- `i`: numeric, what slice to use.
- `value`: values to assign to the portion of an array specified by `d` and `i`.
- `affinity`: list, output from `affinity` function.

Details

`list2array` turns a list of arrays, each with the same dimensions, into a new array having one more dimension whose size is equal to the number of initial arrays.

`slicex` extracts or assigns values from/to the `i`th slice(s) in the `d`th dimension of an array. Values are assigned to an array if `value` is not NULL. This function works by building an expression containing the extraction operator (\[\]).

`slice.affinity` performs a slice operation on the `values` element of the `affinity` variable (which should be the output of `affinity`). This function is used e.g. by `anim.TCA` to extract slices that are the basis for frames of an animated stability diagram.

`dimsums` sums an array along the `d`th dimension using only the `i`th slices in that dimension. If `i` is NULL, all slices in that dimension are summed together. For matrices, `dimsums(x, 1)` has the same result as `colsums(x)` and `dimsums(x, 2)` has the same result as `rowsums(x)`.

In the examples below, the “stopifnot” tests fail unless ‘a’ and ‘b’ are both created as multiples of the starting matrix ‘x’. This behavior probably reflects the internal representation of these matrices in R.

Examples

```r
# start with a matrix
x <- matrix(1:12, ncol=3)
# pay attention to the following when # writing examples that test for identity!
identical(x, x) # FALSE
# create two matrices that are multiples of the first
a <- 1*x
b <- 2*a
# these both have two dimensions of lengths 4 and 3
dim(a) # 4 3
# combine them to make an array with three dimensions
c <- list2array(list(a, b))
# the third dimension has length 2
dim(c) # 4 3 2
# the first slice of the third dimension == a
stopifnot(identical(slicex(c, 3), a ))
# the second slice of the third dimension == b
stopifnot(identical(slicex(c, 3, 2), b ))
# 'slice' works just like the bracket operator
c11 <- slicex(c, 1)
```
```r
c12 <- slice(c, 1, 2)
c21 <- slice(c, 2, 1)
c212 <- slice(c, 2, 1:2)
stopifnot(identical(c11, c[1:1, ]))
stopifnot(identical(c12, c[2:2, ]))
stopifnot(identical(c21, c[1:1, ]))
stopifnot(identical(c212, c[1:2, ]))
# let us replace part of the array
d <- slice(c, 3, 2, value = a)
# now the second slice of the third dimension == a
stopifnot(identical(slice(d, 3, 2), a))
# and the sum across the third dimension == b
stopifnot(identical(dimSums(d, 3), b))
# taking the sum removes that dimension
dim(d)  # 4 3 2
dim(dimSums(d, 1))  # 3 2
dim(dimSums(d, 2))  # 4 2
dim(dimSums(d, 3))  # 4 3

# working with an 'affinity' object

basis("CHNOS+")
species("alanine")
a1 <- affinity(O2 = c(-80, -60))  # at pH=7
a2 <- affinity(O2 = c(-80, -60), pH = c(0, 14, 7))
# in the 2nd dimension (pH) get the 4th slice (pH=7)
a3 <- slice.affinity(a2, 2, 4)
stopifnot(all.equal(a1$svalues, a3$svalues))
```

---

**util.blast**

*Functions to Work with BLAST Output Files*

**Description**

Read and filter BLAST tabular output files, make taxonomic identifications of the BLAST hits using gi numbers, write trimmed-down BLAST files.

**Usage**

```r
read.blast(file, similarity = 30, evalue = 1e-5, max.hits = 1,
            min.length = NA, quiet = FALSE)
id.blast(blast, gi.taxid, taxid.names, min.taxon = 0,
         min.query = 0, min.phylum = 0, take.first = TRUE)
write.blast(blast, outfile)
def2gi(def)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>character, name of BLAST tabular output file</td>
</tr>
</tbody>
</table>
read.blast reads a BLAST (Altschul et al., 1997) tabular output file (such as generated using the -m 8 switch to the ‘blastall’ command), keeping only those hits with greater than or equal to similarity and less than or equal to evalue (expectation value). Furthermore, for each query sequence, only the top number of hits specified by max.hits are kept, and only hits with an alignment length of at least min.length are kept. One or more of these filters can be disabled by setting similarity, evalue and/or max.hits to NA.

id.blast takes a BLAST table (i.e., the output of read.blast) and finds the taxonomic ID, phylum and species name for each hit (subject sequence). The BLAST results are tied to taxids using gi.taxid, which is a list consisting of ‘gi’ and ‘taxid’ numeric vectors. Any subject sequence identifiers appearing in the BLAST file that do not match gi numbers in the gi.taxid list are dropped. The taxid.names dataframe lists the phylum and species names for each taxid.

id.blast furthermore performs three possible filtering steps, which are all disabled by default. If one or more of the arguments is set to a non-zero value, its operation is performed, in this order. Any taxon that does not initially make up at least the fraction of total hits given by min.taxon is removed. Any query sequence that does not have a single phylum making up at least the fraction of hits (for each query sequence) given by min.query is removed. Finally, any phylum that does not make up at least the fraction of total hits given by min.phylum is removed.

By default, for take.first equal to TRUE, id.blast performs a final filtering step (but min.query must be disabled). Only the first hit for each query sequence is kept.

write.blast takes a BLAST table (the output of read.blast) and writes to outfile a stripped-down BLAST file with empty values in the columns except for columns 1 (query sequence ID), 2 (hit sequence ID), 3 (similarity), 11 (E value). In the process, def2gi is used to extract the GI numbers for the hit sequences that are then kept in the second column. This function is used

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>similarity</td>
<td>numeric, hits above this similarity score are kept</td>
</tr>
<tr>
<td>evalue</td>
<td>character, hits below this E value are kept</td>
</tr>
<tr>
<td>max.hits</td>
<td>numeric, up to this many hits are kept for each query sequence</td>
</tr>
<tr>
<td>min.length</td>
<td>numeric, hits with at least this alignment length are kept</td>
</tr>
<tr>
<td>quiet</td>
<td>logical, produce fewer messages?</td>
</tr>
<tr>
<td>blast</td>
<td>dataframe, BLAST table</td>
</tr>
<tr>
<td>gi.taxid</td>
<td>list, first component is sequence identifiers (gi numbers), second is taxon ids (taxids)</td>
</tr>
<tr>
<td>taxid.names</td>
<td>dataframe, with at least columns ‘taxid’ (taxon id), ‘phylum’ (name of phylum), ‘species’ (name of species)</td>
</tr>
<tr>
<td>min.taxon</td>
<td>numeric, this taxon is kept if it makes up at least this fraction of total</td>
</tr>
<tr>
<td>min.query</td>
<td>numeric, query sequence is counted if a single phylum makes up this fraction of its hits</td>
</tr>
<tr>
<td>min.phylum</td>
<td>numeric, this phylum is kept if it makes up at least this fraction of total</td>
</tr>
<tr>
<td>take.first</td>
<td>logical, keep only first hit after all other filtering steps?</td>
</tr>
<tr>
<td>outfile</td>
<td>character, name of output file</td>
</tr>
<tr>
<td>def</td>
<td>character, FASTA defline(s)</td>
</tr>
</tbody>
</table>
to reduce the size of the example BLAST files that are packaged with CHNOSZ (see the ‘bison’ section in extdata).

def2gi extracts the GI number from a FASTA defline.

Value

read.blast returns a dataframe with as many columns (12) as the BLAST file. id.blast returns a dataframe with columns query (i.e., sequence id or gi number), similarity, evalue, taxid, phylum and species. write.blast invisible-y returns the results (that are also written to outfile).

References


See Also

The extdata/refseq directory contains the taxid_names.csv.xz file for microbial taxa, which can be used for the taxid.names in id.blast.

Examples

```r
## using def2gi
def <- "gi\[218295810\]|ref\[ZP_03496590.1\]"
stopifnot(all.equal(def2gi(def), "218295810"))

## process some of the BLAST output for proteins
## from Bison Pool metagenome (JGI, 2007)
tfile <- system.file("extdata/bison/gi.taxid.txt.xz", package="CHNOSZ")
gi.taxid <- scan(tfile, what=as.list(character(2)), flush=TRUE)
# read the file that connects names with the taxids
nfile <- system.file("extdata/refseq/taxid_names.csv.xz", package="CHNOSZ")
taxid.names <- read.csv(nfile)
# the BLAST files
sites <- c("N","S","R","Q","P")
bfile <- paste("extdata/bison/bison", sites, ".vs_refseq$7.blastp.xz", sep="")
for(i in 1:5) {
  file <- system.file(bfile[i], package="CHNOSZ")
  # read the blast file, with default filtering settings
  bl <- read.blast(file)
  # process the blast file -- get taxon names
  ib <- id.blast(bl, gi.taxid, taxid.names, min.taxon=2^-7)
  # count each of the phyla
  bd <- as.matrix(sapply(unique(ib$phylum), function(x) (sum(x==ib$phylum)))))
colnames(bd) <- sites[i]
  # make a matrix -- each column for a different file
  if(i==1) bardata <- bd else {
    bardata <- merge(bardata, bd, all=TRUE, by="row.names")
```
```r
rownames(bardata) <- bardata$Row.names
bardata <- bardata[, -1]
}
# normalize the counts
bardata[is.na(bardata)] <- 0
bardata <- t(t(bardata)/colSums(bardata))
# make a bar chart
bp <- barplot(as.matrix(bardata), col=rainbow(nrow(bardata)),
    xlab="location", ylab="fractional abundance")
# add labels to the bars
names <- substr(rownames(bardata), 1, 3)
for(i in 1:5) {
    bd <- bardata[, i]
    ib <- bd! = 0
    y <- (cumsum(bd) - bd/2)[ib]
    text(bp[i], y, names[ib])
}
title(main=paste("Phylum Classification of Protein Sequences",
    "in Part of the Bison Pool Metagenome", sep="\n"))
```

---

**util.character**

*Functions to Manipulate Character Objects*

**Description**

Convert between strings and character objects. Test for ability to become numeric.

**Usage**

```r
c2s(x, sep = " ")
s2c(x, sep = NULL, keep.sep = TRUE)
```

can.be.numeric(x)

**Arguments**

- **x**: character object to convert (s2c, c2s, axis.label), or object to be tested (can.be.numeric).
- **sep**: character, the separator to insert or separator(s) to match (c2s, s2c).
- **keep.sep**: logical, retain the separator in the output (TRUE) or discard it (FALSE) (s2c).

**Details**

c2s joins the elements of a character object into a character object of length 1 (a string), and s2c splits a string into elements of a character object of length \( n + 1 \), where \( n \) stands for the number of separators in the string. sep gives the separator to insert between successive items (in c2s) or the separator(s) to find in a string (in s2c). The default value of sep is a space (" ") in c2s. The default value for sep is NULL in s2c, indicating a separator at every position of x (the result in this case has length equal to nchar(x)). Argument keep. sep if TRUE (the default) instructs s2c to keep the
separating values in the output. The maximum length of the object returned by \texttt{s2c} is determined by \texttt{n}; the default value of \texttt{NULL} indicates an unrestricted length.

can.be.numeric returns a value of TRUE or FALSE for each element of \texttt{x}.

**Value**

\texttt{s2c}, \texttt{c2s} and \texttt{axis.label} return character values. \texttt{can.be.numeric} returns logical.

**Examples**

```r
## string to character
s2c("hello world")
s2c("hello world", sep=" ", keep.sep=FALSE)
s2c("3.141592", sep=\texttt{c(\texttt{\".\"},\texttt{\"9\")})
# character to string
c2s(\texttt{aminoacids()})
c2s(\texttt{aminoacids()}, sep="\")
```

**Description**

Add species to or alter properties of species in the thermodynamic database or in the buffer definition table. Show references for sources of thermodynamic data in a web browser. Check internal consistency of individual entries in database.

**Usage**

```r
add.obigt(file = \texttt{system.file("extdata/thermo/OBIGT-2.csv", package = "CHNOSZ"), force = FALSE, E.units = "cal")}
mod.obigt(\ldots)
today()
browse.refs(key=NULL)
checkEOS(eos, state, prop, ret.diff = FALSE)
checkGHS(ghs, ret.diff = FALSE)
check.obigt()
obigt2eos(obigt, state, fixGHS = FALSE)
RH2obigt(compound = NULL, state = "cr",
      file = \texttt{system.file("extdata/thermo/RH98_Table15.csv", package = "CHNOSZ")})
```

**Arguments**

- **file** character, path to a file
- **force** logical, force replacement of already existing species?
- **E.units** character, units of energy, ‘cal’ or ‘J’
... character or numeric, properties of species to modify in the thermodynamic database

key character, numeric, or list, containing reference key(s) for which to show URL(s) in browser

eos dataframe, containing equations-of-state parameters in the format of \texttt{thermo$obigt}

state character, physical state of species

prop character, property of interest (‘\texttt{cp}’ or ‘\texttt{v}’)

ret.diff logical, return the difference between calculated and tabulated values?

ghs dataframe, containing G, H and S, in the format of \texttt{thermo$obigt}

obigt dataframe, in the format of \texttt{thermo$obigt}

fixGHS logical, calculate one of missing G, H, or S?

compound character, name of compound(s) in group additivity calculation

Details

\texttt{add.obigt} reads data from the specified file and adds it to \texttt{thermo$obigt}. Set \texttt{force} to \texttt{TRUE} to replace species that exist in the thermodynamic database (each unique combination of a name and a state in the database is considered to be a unique species). \texttt{force}, if not specified, reverts to \texttt{TRUE} if the file is left at its default. Given the default setting of \texttt{E.units}, the function does not perform any unit conversions. If \texttt{E.units} is set to ‘\texttt{J}’, then the thermodynamic parameters are converted from units of Joules to calories, as used in the CHNOSZ database.

\texttt{mod.obigt} changes one or more of the properties of species or adds species to the thermodynamic database. These changes are lost if you reload the database by calling \texttt{dataHthermoI} or if you quit the R session without saving it. The name of the species to add or change must be supplied as the first argument of \ldots or as a named argument (named ‘name’). When adding new species, a chemical formula should be included along with the values of any of the thermodynamic properties. The formula is taken from the ‘formula’ argument, or if that is missing, is taken to be the same as the ‘name’ of the species. An error results if the formula is not valid (i.e. can not be parsed by \texttt{makeup}). Additional arguments refer to the name of the property(s) to be updated and are matched to any part of compound column names in \texttt{thermo$obigt}, such as ‘\texttt{z}’ or ‘\texttt{T}’ in ‘\texttt{z.T}’. Unless ‘state’ is specified as one of the properties, its value is taken from \texttt{thermo$opt$state}. When adding species, properties that are not specified become \texttt{NA} (except for ‘state’). The values provided should be in the units specified in the documentation for the \texttt{thermo} data object, including any order-of-magnitude scaling factors.

today returns the current date in the format adopted for \texttt{thermo$obigt} (inherited from SUPCRT-format data files) e.g. ‘\texttt{13.May.12}’ for May 13, 2012.

\texttt{change} is a wrapper function to \texttt{mod.obigt} and \texttt{mod.buffer}. The name provided in the argument refers to the name or numeric index of the species to update or add using \texttt{mod.obigt}, unless the name begins with an underscore character, in which case the remaining part of the name (after the underscore) is passed to \texttt{mod.buffer}. The arguments in \ldots are sent without change to the subordinate function.

\texttt{browse.refs} with default arguments uses \texttt{browseURL} to display the sources of thermodynamic data in \texttt{thermo$refs}, with the URLs in that table showing as hyperlinks in the browser. Otherwise, if \texttt{key} is character, the URLs of those reference keys are opened in the browser. If \texttt{key} is numeric, the values refer to the species in those rows of \texttt{thermo$obigt}, and the URLs for each listed reference
(thermo$obigt$ref1, thermo$obigt$ref2) are opened. If key is a list, it is interpreted as the result of a call to subcrt, and the reference URLs for each species involved in the calculation are opened.

cHECKeOS compares heat capacity and volume calculated from equation-of-state parameters with reference (tabulated) values at 25 °C and 1 bar and prints a message and returns the calculated value if tolerance is exceeded. The Helgeson-Kirkham-Flowers equations of state parameters are in eos, which is a data frame with columns (and column names) in the same format as thermo$obigt. The property can be one of ‘Cp’ or V. The code only distinguishes between states of ‘aq’ and all others. The default tolerances, given in thermo$opt$Cp.tol and thermo$opt$V.tol, are 1 cal/K.mol for Cp and 1 cm³/mol for V. If ret.diff is TRUE, the differences are returned irrespective of their values, and no messages are printed.

cHECKghS compares G (standard molal Gibbs energy of formation from the elements) calculated from H (standard molal enthalpy of formation) and S (standard molal entropy) with reference (tabulated) values of G at 25 °C and 1 bar. A message is printed and the calculated difference is returned if it exceeds the value given in thermo$opt$G.tol, which has a default value of 100 cal/mol. The calculation requires that G, H and S, and the chemical formula of the species all be present. checkEOS and checkGHs are used by info when retrieving species parameters from the database.

cHECK.obigt is a function to check self-consistency of each entry (with itself) in the thermodynamic database, using checkEOS and checkGHs. The function checks data in both thermo$obigt and extdata/thermo/OBIGT-2.csv. The output is a table listing only species that exceed at least one of the tolerance limits, giving the name of the database (OBIGT or OBIGT-2), the species index (rownumber in the database), species name and state, and DCp, DV and DG, for the calculated differences (only those above the tolerances are given). This function is used to generate the file found at extdata/thermo/obigt_check.csv.

obigt2eos takes a data frame in the format of thermo$obigt of one or more rows, removes scaling factors from equations-of-state parameters, and applies new column names depending on the state. This function is used by both info and subcrt when retrieving entries from the thermodynamic database. If fixGHs is TRUE a missing one of G, H or S for any species is calculated from the other two and the chemical formula of the species (used by subcrt when retrieving database entries).

RH2oobigt implements a group additivity algorithm for standard molal thermodynamic properties and equations of state parameters of crystalline and liquid organic molecules from Richard and Helgeson, 1998. The names of the compounds and their physical state are searched for in the indicated file, that also contains chemical formulas and group stoichiometries; the names of the groups are stored in the column names of this file, and must be present in thermo$obigt. The default file (extdata/thermo/RH98_Table15.csv) includes data taken from Table 15 of Richard and Helgeson, 1998 for high molecular weight compounds in ‘cr’ystalline and ‘liq’uid states. An error is produced if any of the compound-state combinations is not found in the file, if any of the group names for a given compound-state combination is not found in thermo$obigt, or if the chemical formula calculated from group additivity (with the aid of i2A and as.chemical.formula) is not identical to that listed in the file.

Value

The values returned (invisible-y) by mod.obigt are the rownumbers of the affected species.

Warning

add.obigt affects the order of entries in thermo$obigt; therefore, it should be called before any
basis or species definition. Also, there is no attempt made to keep the order of physical states in the database (aq-cr-liq-gas); the function simply adds new rows to the end of thermoObigt. As a result, retrieving the properties of an added aqueous species using info requires an explicit state="aq" argument to that function if a species with the same name is in one of the cr, liq or gas states.

References


See Also

The default supplementary thermodynamic database for add.obigt (extdata/thermo/OBIGT-2.csv) is used in some of the example calculations in the help page for diagram and also in anim.TCA. mod.buffer for updating the list of available buffers.

Examples

```r
## modify an existing species with fake properties
ialanine <- mod.obigt("alanine", state="cr", G=0, H=0, S=0)
# we have made the values of G, H, and S inconsistent
# with the elemental composition of alanine so the following
# produces a message about that
info(ialanine)
## add a species
icl20 <- mod.obigt("Cl2O", G=20978)
info(icl20)
# add a species with a name that is distinct from the formula
mod.obigt("buckministerfullerene", formula="C60", state="cr")
## Not run:
## marked dontrun because they open pages in a browser
## show the contents of thermo$refs
browse.refs()
# Internet needed for the following examples:
# open URL for Helgeson et al., 1998
browse.refs("HOK+98")
# open two URLs for alanine
browse.refs(info("alanine"))
# open three URLs for species in the reaction
s <- subcr(c("O2","O2"),c("gas","aq"),c(-1,1))
browse.refs(s)
## End(Not run)
```

## calculate thermodynamic properties of organic compounds
## using group additivity, after Richard and Helgeson, 1998
RH2obigt()
Functions to Express Chemical Formulas and Properties

Description

Generate expressions suitable for axis labels and plot legends describing chemical species, properties and reactions.

Usage

 expr.species(species, state = "", log = "", value=NULL)
 expr.property(property)
 expr.units(property, prefix = "", per = "mol")
 axis.label(label, units = NULL, basis = get("thermo")$basis, prefix = "")
 describe.basis(basis = get("thermo")$basis, ibasis = 1:nrow(basis), digits = 1,
 oneline = FALSE)
 describe.property(property, value, digits = 1, oneline = FALSE, ret.val = FALSE)
 describe.reaction(reaction, iname = numeric(), states = NULL)

Arguments

species character, formula of a chemical species
state character, designation of physical state
log character, designation of physical state (for logarithm of activity or fugacity)
value numeric, logarithm of activity or fugacity of species, or value of other property
property character, description of chemical property
prefix character, prefix for units
per character, denominator in units
label character, description of species, condition or property
units character, description of units
basis data frame, definition of basis species
ibasis numeric, which basis species to include
digits numeric, number of digits to show after decimal point
oneline logical, make descriptions occupy a single line?
ret.val logical, return only the value with the units?
reaction data frame, definition of reaction
iname numeric, show names instead of formulas for these species
states character, if ‘all’, show states for all species
Details

The expr.* functions create expressions using the plotmath syntax to describe the names and states and logarithms of activity or fugacity of chemical species, conditions including temperature and pressure and chemical properties such as Gibbs energy and volume. expr.species takes as input the formula of a single chemical species and constructs an expression including subscripted coefficients, and a suffixed designation of physical state (italicized, in parentheses) if provided. If log designates a physical state (as in thermo$obigt$state), the expression includes a 'log' prefix, followed by 'f' for fugacity of gaseous species, or 'a' for activity of species in all other states.

expr.property accepts a description in property that indicates the chemical property of interest. Uppercase letters are italicized, and lowercase letters are italicized and subscripted. Other specific characters are parsed as follows (case-sensitive):

‘D’ Delta
‘A’ bold A (chemical affinity)
‘p’ subscript italic P (isobaric heat capacity)
‘φ’ degree sign (standard-state property)

A ‘φ’ gets interpreted as a degree sign only if it does not immediately follow a number (so that e.g. ‘2.303’ can be included in an expression).

Every other character that is one of the letters or LETTERS in the description of the property is italicized in the expression; other characters such as numerals or mathematical operators are shown without any special formatting. Special cases for the property argument (‘logk’, ‘Eh’, ‘pH’, ‘pe’, ‘IS’ and ‘ZC’) are interpreted as simple expressions, and are not parsed according to the above rules.

expr.units returns an expression for the units, based on one or more characters appearing in the property:

‘A’, ‘G’, ‘H’ energy
‘Cp’, ‘S’ energy per Kelvin
‘V’ volume
‘E’ volume per Kelvin
‘P’ pressure
‘T’ temperature
‘Eh’ electrical potential
‘IS’ ionic strength

If none of those characters appears in the property, the expression is an empty character (no units). If a prefix is given, it is added to the expression. The denominator of the units (default ‘mol’) is taken from the per argument; it is applied to all units except for ‘P’, ‘T’, ‘Eh’, and ‘IS’.

axis.label accepts a generic description of a label. If this matches the chemical formula of one of the basis species in the basis argument, the expression for the label is generated using expr.species with log set to the physical state of the basis species. Otherwise, the expression is built by combining the output of expr.property with expr.units (or the value in units, if it is supplied), placing a comma between the two. This function is used extensively in diagram and also appears in many of the examples.
describe.basis makes an expression summarizing the basis species definition (logarithms of activity or fugacity of the basis species) provided in basis; only the basis species identified by ibasis are included.

describe.property makes an expression summarizing the properties supplied in property, along with their values. The expressions returned by both functions consist of a property, an equals sign, and a value (with units where appropriate); the expressions have a length equal to the number of property/value pairs. If oneline is TRUE, the property/value pairs are combined into a single line, separated by commas. The number of digits shown after the decimal point in the values is controlled by digits. If ret.val is TRUE, only the values and their units are returned; this is useful for labeling plots with values of temperature.

describe.reaction makes an expression summarizing a chemical reaction. The reaction data frame can be generated using subcrt. Based on the sign of their reaction coefficients, species are placed on the reactant (left) or product (right) side of the reaction, where the species with their coefficients are separated by plus signs; the two sides of the reaction are separated by an equals sign. Coefficients equal to 1 are not shown. Chemical formulas of species include a designation of physical state if states is ‘all’. Names of species (as provided in reaction) are shown instead of chemical formulas for the species identified by iname.

Examples

```r
# show descriptions of species and properties on a plot
plot(0, 0, xlim=c(1,5), ylim=c(1,5), xlab="function", ylab="example")
text0 <- function(...) text(..., adj=0)
  # species
text0(1, 1, expr.species("CO2"))
text0(1, 2, expr.species("CO2", state="aq"))
text0(1, 3, expr.species("CO2", state="aq", log="aq"))
text0(1, 4, expr.species("CO2", log="aq"))
text0(1, 5, expr.species("CO2", log="aq", value=-3))
  # properties
text0(2, 1, expr.property("A"))
text0(2, 2, expr.property("DV"))
text0(2, 3, expr.property("DG0f"))
text0(2, 4, expr.property("DCp0,r"))
text0(2, 5, expr.property("T"))
  # units
text0(3, 1, expr.units("A", prefix="k"))
text0(3, 2, expr.units("DV"))
text0(3, 3, expr.units("DG0f", prefix="k"))
text0(3, 4, expr.units("DCp0,r"))
text0(3, 5, expr.units("T"))
  # axis.label
text0(4, 1, axis.label("DG0f"))
text0(4, 2, axis.label("T"))
text0(4, 3, axis.label("pH"))
text0(4, 4, axis.label("Eh"))
text0(4, 5, axis.label("IS"))
# describe.basis
basis("CHNO+S")
dbasis <- describe.basis(oneline=TRUE, digits=0)
```
util.fasta

Functions for Reading FASTA Files and Downloading from UniProt

Description

Search the header lines of a FASTA file, read protein sequences from a file, count numbers of amino acids in each sequence, and download sequences from UniProt.

Usage

grep.file(file, pattern = "", y = NULL, ignore.case = TRUE, startswith = "">", lines = NULL, grep = "grep")
read.fasta(file, i = NULL, ret = "count", lines = NULL, ihead = NULL, start=NULL, stop=NULL, type="protein", id = NULL)
count.aa(seq, start=NULL, stop=NULL, type="protein")
uniprot.aa(protein, start=NULL, stop=NULL)

Arguments

file character, path to FASTA file
pattern character, pattern to search for in header lines
y character, term to exclude in searching sequence headers
ignore.case logical, ignore differences between upper- and lower-case?
startswith character, only lines starting with this expression are matched
lines list of character, supply the lines here instead of reading them from file
grep character, name of system ‘grep’ command
i numeric, line numbers of sequence headers to read
ret character, specification for type of return (count, sequence, or FASTA format)
ihead numeric, which lines are headers
start numeric, position in sequence to start counting
Details

grep.file returns the line numbers of header lines in a FASTA file. Matching header lines are identified having the search term pattern and optionally a term to exclude in y. The ignore.case option is passed to grep, which does the work of finding lines that match. Only lines that start with the expression in startswith are searched; the default setting reflects the format of the header lines in a FASTA file. If y is NULL and a supported operating system is identified, the operating system’s `grep` function (or other specified in the grep argument) is applied directly to the file instead of R’s `grep`. This avoids having to read the file into R using `readLines`. If the lines from the file were obtained in a preceding operation, they can be supplied to this function in the lines argument.

read.fasta is used to retrieve entries from a FASTA file. To read only selected sequences pass the line numbers of the header lines to the function in `i` (they can be identified using e.g. grep.file). The function returns various formats depending on the value of `ret`. The default ‘count’ returns a data frame of amino acid counts (the data frame can be given to add.protein in order to add the proteins to thermo$protein), ‘seq’ returns a list of sequences, and ‘fas’ returns a list of lines extracted from the FASTA file, including the headers (this can be used e.g. to generate a new FASTA file with only the selected sequences). Similarly to grep.file, this function utilizes the OS’s `grep` on supported operating systems in order to identify the header lines as well as ‘cat’ to read the file, otherwise `readLines` and R’s `substr` are used to read the file and locate the header lines. If the line numbers of the header lines were previously determined, they can be supplied in `ihead`. Optionally, the lines of a previously read file may be supplied in `lines` (in this case no file is needed so `file` should be set to ""). When `ret` is ‘count’, the names of the proteins in the resulting data frame are parsed from the header lines of the file, unless `id` is provided.

count.aa counts the occurrences of each amino acid or nucleic-acid base in a sequence (seq). For amino acids, the columns in the returned data frame are in the same order as thermo$protein. Letters are matched without regard for case. A warning is generated if any character in seq, excluding spaces, is not one of the single-letter amino acid or nucleobase abbreviations. start and/or stop can be provided to count a fragment of the sequence (extracted using `substr`). If only one of start or stop is present, the other defaults to 1 (start) or the length of the sequence (stop).

uniprot.aa returns a data frame of amino acid composition, in the format of thermo$protein, retrieved from the protein sequence if it is available from UniProt (http://uniprot.org; The UniProt Consortium, 2012). The protein argument corresponds to the ‘Entry name’ on the UniProt search pages.

Value

grep.file returns a numeric vector. read.fasta returns a list of sequences or lines (for `ret` equal to ‘seq’ or ‘fas’, respectively), or a data frame with amino acid compositions of proteins (for `ret` equal to ‘count’) with columns corresponding to those in thermo$protein.
References


See Also

*nucleic.formula* for an example of counting nucleobases in a DNA sequence. When computing relative abundances of many proteins that might be found with grep.file and read.fasta, consider using the iprotein argument of *affinity* to speed things up; for an example see the help page for *revisit*.

Examples

```r
## reading a protein FASTA file
# the path to the file
file <- system.file("extdata/fasta/EF-Tu.aln", package="CHNOSZ")
# read the sequences, and print the first one
read.fasta(file, ret="seq")[[1]]
# count the amino acids in the sequences
aa <- read.fasta(file)
# compute lengths (number of amino acids)
protein.length(aa)

## Not run:
# download amino acid composition of a protein
# start at position 2 to remove the initiator methionine
aa <- uniprot.aa("ALAT1_HUMAN", start=2)
# add it to thermo$protein
ip <- add.protein(aa)
# now it's possible to calculate some properties
protein.length(ip)
protein.formula(ip)
subcrt("ALAT1_HUMAN", c("cr", "aq"), c(-1, 1))
# the amino acid composition can be saved for future use
write.csv(aa, "saved.aa.csv", row.names=FALSE)
# in another R session, the protein can be loaded without using uniprot.aa()
aa <- read.csv("saved.aa.csv")
add.protein(aa)

## count amino acids in a sequence
count(aa("GGS GG"))
# warnings are issued for unrecognized characters
astest <- count(aa("WhatAmI MadeOf?"))
# there are 3 "A" (alanine)
stopifnot(astest[1, "A"]==3)

## End(Not run)
```
util.formula

Functions to Work with Chemical Formulas

Description

Calculate the standard molal entropy of elements in a compound; calculate the standard molal Gibbs energy or enthalpy of formation, or standard molal entropy, from the other two; list coefficients of selected elements in a chemical formula; calculate the average oxidation state of carbon. Create a stoichiometric matrix for selected species.

Usage

as.chemical.formula(makeup, drop.zero = TRUE)
generate.formula(formula)
mass(formula)
entropy(formula)
GHS(formula, G = NA, H = NA, S = NA, T = get("thermo")$opt$Tr)
ZC(formula)
i2A(formula)

Arguments

makeup numeric, object returned by makeup
drop.zero logical, drop elements with a coefficient of zero?
formula character, chemical formulas, or numeric, rownumbers in thermo$obigt
G numeric, standard molal Gibbs energy of formation from the elements
H numeric, standard molal enthalpy of formation from the elements
S numeric, standard molal molal entropy
T numeric, temperature in Kelvin

Details

generate.formula accepts a numeric or character argument; the character argument can be mixed i.e. include quoted numbers. as.numeric is tested on every value; numeric values are then interpreted as species indices in the thermodynamic database (rownumbers of thermo$obigt) and the chemical formulas for those species are returned. Values that can not be converted to numeric are returned as-is.

i2A returns a stoichiometric matrix representing the elemental composition of the formulas, e.g., those returned by generate.formula. Each column corresponds to an element that is present in at least one of the formulas; some element counts will be zero if not all formula have the same elements.

If a matrix is passed to either generate.formula or i2A it is returned unchanged.

as.chemical.formula makes a character string representing a chemical formula from a vector of coefficients with names corresponding to the elements (e.g., the output of makeup) or from a stoichiometric matrix (output of i2A). Each elemental symbol is written followed by its coefficient;
negative coefficients are signed. Any coefficients equal to 1 are not explicitly written, and any charge (indicated by makeup as ‘Z’) is shown as a signed number at the end of the formula. If the formula is uncharged, and the last element has a negative coefficient, +0 is shown at the end of the formula to indicate a charge of zero.

The remaining functions documented here accept vectors of chemical formulas, species indices, or a mixture of both, or stoichiometric matrices with elements on the columns. They do so by passing the supplied formula argument through both get_formula and i2A.

mass and entropy return the sums of masses or entropies of elements in each of the formulas. The masses are calculated using the masses of the elements in their natural isotopic distribution, and the entropies, in cal K\(^{-1}\) mol\(^{-1}\), are calculated using the entropies of the compounds of the pure elements in their stable states at 25 °C and 1 bar. The properties of the elements used by this function are taken from thermo$element.

GHS computes one of the standard molal Gibbs energy or enthalpy of formation from the elements, or standard molal entropy, from values of the other two. The formula, G, H and S arguments must all have the same length. The entropies of the elements (S\(e\)) in each formula are calculated using entropy. The equation in effect can be written as \(\Delta G^o = \Delta H^o - T \Delta S^o\), where \(\Delta S^o = S - S_e\) and \(T\) is the temperature given in \(T\) (defaults to 298.15 K) (note that \(G\) and \(H\) in the arguments correspond respectively to \(\Delta G^o\) and \(\Delta H^o\) in the equation). For each formula, if one of \(G\), \(H\), or \(S\) is NA, its value is calculated from the other two. Otherwise, the values are returned unchanged. Units of cal mol\(^{-1}\) (\(DG\), \(DH\)) and cal K\(^{-1}\) mol\(^{-1}\) (\(S\)) are assumed.

ZC returns the average oxidation state of carbon (Z\(_C\)) calculated from ratios of the elements in the chemical formulas. The equation used is \(Z_C = \frac{Z_{-n_H+2(n_O+n_S)+3n_N}}{n_C}\), where the \(n\) refer to the number of the indicated element in the formula (Dick and Shock, 2011). The result is NaN for any formula that does not contain carbon. Elements other than those shown in the equation are not included in the calculation, and produce a warning.

Value

mass, entropy, and ZC return numeric values. as.chemical.formula returns a character object. GHS returns a matrix with column names ‘G’, ‘H’ and ‘S’, and i2A returns a matrix with column names corresponding to the elements in the formulas.

References


See Also

makeup, used by mass and entropy, and ZC and i2A for counting the elements in a formula (the latter two make use of the count.zero argument), group.formulas (and by extension protein.formula) use the stoichiometric matrices created by i2A, as does run.wjd. protein.formula has an example of computing ZC for proteins compiled from the RefSeq database.

Examples
## mass and entropy from chemical formulas

```r
mass("H2O")
entropy("H2O")
mass("-1") # electron
entropy("-1")
```

## three ways to get the formula of alanine

```r
iA <- info("alanine")
info(iA)$formula
as.chemical.formula(makeup(iA))
get.formula(iA)
```

## converting among Gibbs energy, enthalpy, entropy

```r
# calculate the value of G from H and S
GHS("H2O", H=water("H"), S=water("S"))[1,]
# that not quite equal to the value from water("G");
# probably using different entropies of the elements

## average oxidation states of carbon

```r
stopifnot(ZC("CO2") == 4)
stopifnot(ZC("CH4") == -4)
stopifnot(ZC("CHNOSZ") == 7)
si <- info(info("LYSC_CHICK"))
stopifnot(si$formula == "C613H959N193O185S10")
stopifnot(all.equal(ZC(si$formula), 0.0163132137031))
```

## calculate the chemical formulas, then

```r
ZC of all of the proteins in CHNOSZ' database
pf <- protein.formula(thermo$protein)
range(mass(pf))
# use na.rm=TRUE because we have a "protein" with a formula of H2O
range(ZC(pf), na.rm=TRUE)
```

---

### util.list

#### Functions to Work with Lists

**Description**

Combine lists or perform arithmetic operations on elements of lists.

**Usage**

```r
lsub(x, y)
lsum(x, y)
pprod(x, y)
which.pmax(elts, na.rm = FALSE, pmin = FALSE)
```
Arguments

- **x**: list
- **y**: list (lsub, lsum), or numeric (pprod)
- **elts**: list, numeric vectors for which to find maximum values (in parallel) (which.pmax).
- **na.rm**: logical, remove missing values?
- **pmin**: logical, find minimum values instead of maximum ones?

Details

- `lsub` subtracts the elements of list `y` from the respective ones in list `x`. `lsum` sums the respective elements of lists `x` and `y`. `pprod` multiplies each element of list `x` by the respective numeric value in `y`.
- `which.pmax` takes a list of equal-length numeric vectors (or objects that can be coerced to numeric) in `elts` and returns the index of the vector holding the maximum value at each position. If `na.rm` is TRUE, values of NA are removed; if `pmin` is TRUE the function finds locations of the minimum values instead.

Value

- `lsub`, `lsum` and `pprod` return lists.

util.matrix  
Functions for Various Matrix Operations

Description

Find rows of a matrix that form invertible (linearly independent) combinations.

Usage

invertible.combs(A, nmax=20)

Arguments

- **A**: A matrix, with at least as many rows as columns.
- **nmax**: The maximum number of rows to consider.

Details

Given a matrix `A`, with number of rows equal to or greater than the number of columns, return the combinations of row numbers that constitute invertible square matrices. Consider only the first `nmax` rows of the original matrix (to save time for large systems).
Examples

```r
## what combinations of the 20 common amino acids have
## a linearly independent stoichiometry with five elements?
# the names of the amino acids
aanames <- aminoacids(""")
# their species indices
iaa <- suppressMessages(info(aanames))
# the full stoichiometric matrix
A <- i2A(iaa)
# the invertible combinations
ica <- invertible.combs(A)
stopifnot(nrow(ica)==6067)
# that's a bit less than 40% of all possible combinations
nrow(ica) / ncol(combn(20, 5))
# count the occurrences of each amino acid
counts <- table(ica)
names(counts) <- aminoacids(1)
(sc <- sort(counts))
# the two sulfur-containing ones show up most frequently
stopifnot(tail(names(sc), 2)==c("C", "M"))
```

---

**util.mis**

Functions for Miscellaneous Tasks

Description

Calculate $dP/dT$ and temperature of phase transitions, calculate heat capacities of unfolded proteins using an equation from the literature, calculate non-ideal contributions to apparent standard molal properties, identify a conserved basis species, scale logarithms of activity to a desired total activity, calculate Gibbs energy of transformation of a system.

Usage

```r
dpdttr(x)
ttr(x, P = 1, dpdT = NULL)
nonideal(species, proptable, IS, T)
which.balance(species)
unitize(logact = NULL, length = NULL, logact.tot = 0)
```

Arguments

- `x`: numeric index of a mineral phase (`dpdttr`, `ttr`)
- `P`: numeric, pressure (bar)
- `dpdT`: numeric, values of $(dP/dT)$ of phase transitions (`ttr`)
- `species`: Names or indices of species for which to calculate nonideal properties (`nonideal`), or dataframe, species definition such as that in `thermo$species` (`which.balance`)
proptable list of dataframes of species properties
T numeric, temperature (K) (lines.water, nonideal)
IS numeric, ionic strength(s) used in nonideal calculations, mol kg\(^{-1}\)
logact numeric, logarithms of activity
length numeric, numbers of residues
logact.tot numeric, logarithm of total activity

Details
dPdTtr returns values of \((dP/dT)_{Ttr}\), where \(Ttr\) represents the transition temperature, of the phase transition at the high-\(T\) stability limit of the \(x\)th species in thermo$obigt$ (no checking is done to verify that the species represents in fact one phase of a mineral with phase transitions). \(dPdTtr\) takes account of the Clapeyron equation, \((dP/dT)_{Ttr}=\Delta S/\Delta V\), where \(\Delta S\) and \(\Delta V\) represent the changes in entropy and volume of phase transition, and are calculated using subcrt at \(Ttr\) from the standard molal entropies and volumes of the two phases involved. Using values of \(dPdT\) calculated using \(dPdTtr\) or supplied in the arguments, \(Ttr\) returns as a function of \(P\) values of the upper transition temperature of the mineral phase represented by the \(x\)th species.

nonideal takes a list of dataframes (in proptable) containing the standard molal properties of the identified species. For those species whose charge (determined by the number of \(Z\) in their makeup) is not equal to zero, the values of IS are combined with Alberty’s (2003) equation 3.6-1 (Debye-Huckel equation) and its derivatives, to calculate apparent molal properties at the specified ionic strength(s) and temperature(s). The lengths of IS and \(T\) supplied in the arguments should be equal to the number of rows of each dataframe in proptable, or one to use single values throughout. The apparent molal properties that can be calculated include \(g\), \(h\), \(s\) and \(cp\); any columns in the dataframes of proptable with other names are left untouched. A column named \(loggam\) (logarithm of gamma, the activity coefficient) is appended to the output dataframe of species properties.

which.balance returns, in order, which column(s) of species all have non-zero values. It is used by diagram and transfer to determine a conservant (i.e. basis species that are conserved in transformation reactions) if none is supplied by the user.

spearman calculates Spearman’s rank correlation coefficient for \(a\) and \(b\).

unitize scales the logarithms of activities given in logact so that the logarithm of total activity of residues is equal to zero (i.e. total activity of residues is one), or to some other value set in logact.tot. length indicates the number of residues in each species. If logact is NULL, the function takes the logarithms of activities from the current species definition. If any of those species are proteins, the function gets their lengths using protein.length.

Value
Numeric returns are made by dPdTtr, Ttr, spearman, mod.obigt Functions with no (or unspecified) returns are thermo.plot.new, thermo.postscript, label.plot and water.lines.

References
See Also

For some of the functions on which these utilities depend or were modeled, see `paste`, `substr`, `tolower`, `par` and `text`.

Examples

```r
## properties of phase transitions
si <- info("enstatite")
# (dP/dT) of transitions
dPdTtr(si) # first transition
dPdTtr(si+1) # second transition
# temperature of transitions (Ttr) as a function of P
ttr(si,P=c(1,10,100,1000))
ttr(si,P=c(1,10,100,1000))

## scale logarithms of activity
# suppose we have two proteins whose lengths are 100 and 200; what are the logarithms of activity of the proteins
# that are equal to each other and that give a total activity of residues equal to unity?
logact <- c(-3,-3) # could be any two equal numbers
length <- c(100,200)
logact.tot <- 0
loga <- unitize(logact,length,logact.tot)
# the proteins have equal activity
stopifnot(identical(loga[1],loga[2]))
# the sum of activity of the residues is unity
stopifnot(isTRUE(all.equal(sum(10*loga*length),1)))
## now, what if the activity of protein 2 is ten
## times that of protein 1?
logact <- c(-3,-2)
loga <- unitize(logact,length,logact.tot)
# the proteins have unequal activity
stopifnot(isTRUE(all.equal(loga[2]-loga[1],1)))
# but the activities of residues still add up to one
stopifnot(isTRUE(all.equal(sum(10*loga*length),1)))
```

Description

Initialize a new plot window using preset parameters, open a postscript file for plotting, add an axis or title to a plot, generate labels for plot axes and for identification of subplots and physical and chemical conditions, add stability lines for water to a diagram.
Usage

thermo.plot.new(xlim, ylim, xlab, ylab, cex = par("cex"),
    mar = NULL, lwd = par("lwd"), side = c(1,2,3,4),
    mgp = c(1.5, 0.3, 0), cex.axis = par("cex"), col = par("col"),
    yline = NULL, axs = "i", do.box = TRUE, ticks = NULL, las = 1,
    xline = NULL)
thermo.axis(lab = "x-axis", side = 1, line = 1.5, cex = par("cex"),
    lwd = par("lwd"), T = NULL, col = par("col"))
label.plot(x, xfrac = 0.95, yfrac = 0.9, cex = 1, paren = TRUE,
    adj = 1)
water.lines(xaxis = "pH", yaxis = "Eh", T = 298.15, P = "Psat",
    which = c("oxidation","reduction"), logaH2O = 0, lty = 2,
    col = par("fg"), xpoints = NULL)
mttitle(main, ...)
residualsplot(residuals, property = "Cp", model = "big")

Arguments

xlim numeric, limits of the $x$-axis
ylim numeric, limits of the $y$-axis
xlab character, $x$-axis label
ylab character, $y$-axis label
cex numeric, character expansion factor for labels
mar numeric, width (number of lines) of margins on each side of plot
lwd numeric, line width
side numeric, which sides of plot to draw axes
mgp numeric, sizes of margins of plot
cex.axis numeric, character expansion factor for names of axes
col character, line color
yline numeric, margin line on which to plot $y$-axis name
axs character, setting for axis limit calculation
do.box logical, draw a box around the plot?
ticks numeric, same effect as side (retained for backwards compatibility)
las numeric, style for axis labels
xline numeric, margin line on which to plot $x$-axis name
lab character, description of axis label
line numeric, margin line to plot axis label
T numeric, temperature (K)
x character, label to place on plot
xfrac numeric, fractional location on $x$-axis for placement of label
yfrac numeric, fractional location on $y$-axis for placement of label
paren logical, add parentheses around label text?
adj numeric, parameter for text alignment
xaxis character, description of x-axis
yaxis character, description of y-axis
p numeric, pressure (bar)
which character, which of oxidation/reduction lines to plot
logah2o numeric, logarithm of the activity of H2O
lty numeric, line type
xpoints numeric, points to plot on x axis
main character, text for plot title
... further arguments passed to mtext
residuals numeric, named vector of residuals to plot
property character, name of property
model character, name of model to use in plot title

Details

thermo.plot.new sets parameters for a new plot, creates a new plot using plot.new, and adds axes and major and minor ticks to the plot. Plot parameters (see par) including cex, mar, lwd, mgp and axs can be given, as well as a numeric vector in ticks identifying which sides of the plot receive tick marks. yline, if present, denotes the margin line (default par('mgp')[1]) where the y-axis name is plotted. The very first time thermo.plot.new is called, it stores the existing value of par(no.readonly=TRUE) in thermo$opar so that the plot device can be reset to the previous state later on.

water.lines plots lines representing the oxidation and reduction stability limits of water on yaxis-xaxis diagrams, where yaxis can be 'Eh' or 'O2', and xaxis can be 'pH' or 'T', which controls which lines ('oxidation', 'reduction', or both (the default)) are drawn, logah2o (default 0) denotes the logarithm of the activity of water, lty (default 2) the line type, col (default par('fg'), the foreground color), and xpoints an optional list of points on the x axis to which to restrict the plotting (default of NULL refers to the axis limits).

label.plot adds identifying text to the plot; the value given for x is made into a label like (a). The location of the label is controlled by xfrac and yfrac (the fractional locations along the respective axes) as well as adj (the text alignment parameter, see text).

thermo.axis is used to add axes and axis labels to plots, with some default style settings (rotation of numeric labels) and conversions between oxidation-reduction scales (called by thermo.plot.new). It also adds minor tick marks.

mtitle can be used to add a multi-line title to a plot. It loops over each element of main and places it on a separate margin line using mtext. This function exists to facilitate using expressions in multiline titles (see revisit for an example.)

residualsplot produces a barchart with options useful for plotting residuals of group contribution models for thermodynamic properties. It plots horizontal bars stacked with largest on top. The names of the residuals argument (i.e., the names of model species) are plotted across from each respective bar. The axis title is taken from the property (probably 'Cp' or 'V'), and the plot title includes the model name. See the 'xadditivity' vignette for examples of these plots.
Side Effects

These functions create or modify a plot.

Description

Get name of calling function; send messages to standard output; use multiple processors if parallel package is loaded.

Usage

caller.name(n)
msgout(..., domain = NULL, appendLF = FALSE)
palply(X, FUN, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>numeric, number of frame to go up</td>
</tr>
<tr>
<td>...</td>
<td>equivalent to the same argument in message or parLapply</td>
</tr>
<tr>
<td>domain</td>
<td>equivalent to the same argument in message</td>
</tr>
<tr>
<td>appendLF</td>
<td>equivalent to the same argument in message</td>
</tr>
<tr>
<td>X</td>
<td>vector, argument for lapply or parLapply</td>
</tr>
<tr>
<td>FUN</td>
<td>function, argument for lapply or parLapply</td>
</tr>
</tbody>
</table>

Details

caller.name returns the name of the calling function n frames up (i.e., for n equal to 2, the caller of the function that calls this one). If called interactively, returns character().

msgout is a variation of message in base R. It writes output to stdout instead of stderr and has a default setting of appendLF of FALSE. This function is used throughout CHNOSZ to generate informative messages, that will show up both in an interactive session and in Sweave output, but that are suppressed while running the test_that testing scripts to make it easier to watch their progress.

palply is a wrapper function to run parLapply if length of X > 100 and package parallel is loaded, otherwise it runs lapply. Note that parLapply is called with methods set to FALSE. If lots of package startup messages are created when running makeCluster (which is called by palply), it can probably be stopped by adding a test for interactive sessions around any library commands in the Rprofile.

See Also

palply is used in read.fasta and grep.file, count.aa and protein.length, affinity when the iprotein argument is given, equil.boltzmann and equil.reaction, and in revisit.
Examples

caller.name()  # character(0)
afun <- function() caller.name()
afun()  # character(0)
bfun <- function() afun()
bfun()  # "bfun"

msgout("h3llo w0rld\n")

---

util.seq  

*Functions to Work with Sequence Data*

Description

Return one- or three-letter abbreviations of amino acids; count nucleotides in nucleic acid sequences, calculate DNA and RNA complements of nucleic acid sequences.

Usage

aminoacids(nchar=1, which=NULL)
nucleic.formula(nucleic = NULL)
nucleic.complement(nucleic = NULL, type="DNA")

Arguments

- **nchar**: numeric, 1 to return one-letter, 3 to return three-letter abbreviations for amino acids
- **which**: character, which amino acids to name
- **nucleic**: data frame, counts of nucleic-acid bases
- **type**: character, target type of nucleic acid (DNA or RNA)

Details

aminoacids returns the one-letter abbreviations (nchar=’1’) or the three-letter abbreviations (nchar=’3’) or the names of the neutral amino acids (nchar=’ ‘) or the names of the amino acids with ionized side chains (nchar=’Z’). The output includes 20 amino acids in alphabetic order by 1-letter abbreviation (the order used in thermo$protein), unless which is provided, indicating the desired amino acids (either as 1- or 3-letter abbreviations or names of the neutral amino acids).

nucleic.formula returns a string representation of the chemical formula for each nucleic-acid composition contained in nucleic. The names of the bases are indicated by the column names of nucleic. At present, the formula is computed as the sum of the chemical formulas of the bases themselves, with no contribution from polymerization (dehydration) or phosphorylation.

See Also

countNaa for counting amino acids or nucleic-acid bases in a sequence; protein.formula for calculating the chemical formulas of proteins.

Examples

```r
## count nucleobases in a sequence
bases <- countNaa("ACCGGGTTT", type="DNA")
# the DNA complement of that sequence
DNA.comp <- nucleic.complement(bases)
# the RNA complement of the DNA complement
RNA.comp <- nucleic.complement(DNA.comp, type="RNA")
# the formula of the RNA complement (bases only)
nucleic.formula(RNA.comp)  # C40H42N32011
```

util.units  

Functions to Convert Units

Description

These functions to convert values between units and set the user’s preferred units.

Usage

```r
P.units(units = NULL)
T.units(units = NULL)
E.units(units = NULL)
convert(value, units, T = get("thermo")$opt$Tr,
P = get("thermo")$opt$Pr, pH = 7, logaH2O = 0)
convert(value, units)
outvert(value, units)
```

Arguments

- `units` character, name of units to set or convert to/from
- `value` numeric, value(s) to be converted
- `T` numeric, temperature (Kelvin), used in ‘G’-‘logK’, ‘pe’-‘Eh’ and ‘logf02’-‘E0’ conversions
- `P` numeric, pressure (bar), used in ‘logf02’-‘E0’ conversions
- `pH` numeric, pH, used in ‘logf02’-‘E0’ conversions
- `logaH2O` numeric, logarithm of activity of water, used in ‘logf02’-‘E0’ conversions
Details

The units settings are used by subcrt, affinity, and diagram to accept input in or convert output to the units desired by the user. The settings, which can be queried or changed with T.units, E.units and P.units, refer to the units of temperature (K or C), energy (cal or J), and pressure (bar, MPa). (The first value in each of those pairs refers to the default units).

The actual units conversions are handled by convert, through which values are transformed into destination units (names not case sensitive). The possible conversions and settings for the units argument are shown in the following table. Note that ‘Eh’ and ‘E0’ both stand for the value of Eh (oxidation-reduction potential in volts); they have different names so that one can choose to convert between Eh and either ‘pe’ or ‘logf02’.

<table>
<thead>
<tr>
<th>property</th>
<th>units</th>
<th>setting of units argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>temperature</td>
<td>°C, K</td>
<td>C, K</td>
</tr>
<tr>
<td>pressure</td>
<td>bar, MPa</td>
<td>bar, MPa</td>
</tr>
<tr>
<td>energy</td>
<td>cal, J</td>
<td>cal, J</td>
</tr>
<tr>
<td>energy</td>
<td>cal, cm³ bar</td>
<td>calories, cm³bar</td>
</tr>
<tr>
<td>energy</td>
<td>cal, [none]</td>
<td>G, logK</td>
</tr>
<tr>
<td>oxidation potential</td>
<td>volt, [none]</td>
<td>Eh, pe</td>
</tr>
<tr>
<td>oxidation potential</td>
<td>volt, [none]</td>
<td>E0, logf02</td>
</tr>
</tbody>
</table>

envert and outvert are wrappers for convert that handle the conditional conversion of values from or to the current units settings. envert converts the value to the units in the argument, and outvert converts the value from the units in the argument, only if they are different than the current setting; otherwise the value is returned unchanged.

Value

Character return for T.units, P.units and E.units; numeric returns by the other functions.

Examples

```r
## examples using convert
# temperature (Kelvin) to degrees C
convert(273.15, "C")
# temperature (degrees C) to Kelvin
convert(100, "K")
# Gibbs energy (cal mol⁻¹) to/from logK
convert(1000, "logK")
convert(1000, "logK", T=373.15)
convert(1, "G")
# Eh (volt) to pe
convert(-1, "pe")
convert(-1, "pe", T=373.15)
# logf02 to E0 (volt)
convert(-80, "E0")
convert(-80, "E0", pH=5)
convert(-80, "E0", pH=5, logaH2O=-5)
```
# calorie to/from joule
convert(10, "J")
convert(10, "cal")
# cm3bar to calories
convert(10, "calories")

## examples showing unit settings
T.units("C")
T1in <- envert(25, "C")  # no conversion
T1out <- outvert(313.15, "K")  # K to C
T.units("K")
T2in <- envert(298.15, "C")  # K to C
T2out <- outvert(-233.15, "C")  # C to K
# these are the same temperature (25 deg C)
stopifnot(all.equal(T1in, T2in))
# these are numerically equivalent (40 deg C / 40 K)
stopifnot(all.equal(T1out, T2out))
T.units("C")  # return to default

---

### Properties of Water

**Description**

Calculate thermodynamic and electrostatic properties of water.

**Usage**

```r
water(property = NULL, T = get("thermo")$opt$Tr, P = "Psat")
water.props(formulation = get("thermo")$opt$water)
water.SUPCRT92(property, T = 298.15, P = 1)
water.IAPWS95(property, T = 298.15, P = 1)
rho.IAPWS95(T = 298.15, P = 1)
water.AW90(T = 298.15, rho = 1000, P = 0.1)
```

**Arguments**

- **property**: character, name(s) of property(s) to calculate
- **T**: numeric, temperature (K)
- **P**: numeric, pressure (units of bar, except MPa for `water.AW90`), or 'Psat' for vapor-liquid saturation
- **formulation**: character, name of formulation for which to return names of available properties
- **rho**: numeric, density (kg m\(^{-3}\))
Details

These functions compute the thermodynamic (Gibbs energy and its derivatives) and electrostatic (dielectric constant and its derivatives) properties of liquid or supercritical H$_2$O using equations of state taken from the literature. The wrapper function water accepts two major computational alternatives. The default option (i.e., `thermoDoptDwater` equal to ‘SUPCRT92’) is retrieve thermodynamic and electrostatic properties as a function of temperature and pressure using a FORTRAN subroutine taken from the SUPCRT92 software package (Johnson et al., 1992). If `thermoDoptDwater` is set to ‘IAPWS95’, the thermodynamic properties are calculated using a (slower) implementation in R code of the IAPWS-95 formulation (Wagner and Pruss, 2002) and electrostatic properties are calculated using the equations of Archer and Wang, 1990.

The allowed properties for water are one or more of those listed below, depending on the computational option; availability is shown by an asterisk. The names of properties in the arguments are not case sensitive. Note that some of the properties that can actually be calculated using the different formulations are not implemented here. Except for rho, the units are those used by Johnson and Norton, 1991.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Units</th>
<th>IAPWS95</th>
<th>SUPCRT92</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Helmholtz energy</td>
<td>cal mol$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>G</td>
<td>Gibbs energy</td>
<td>cal mol$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>S</td>
<td>Entropy</td>
<td>cal K$^{-1}$ mol$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>U</td>
<td>Internal energy</td>
<td>cal mol$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>H</td>
<td>Enthalpy</td>
<td>cal mol$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Cv</td>
<td>Isochoric heat capacity</td>
<td>cal K$^{-1}$ mol$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Cp</td>
<td>Isobaric heat capacity</td>
<td>cal K$^{-1}$ mol$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Speed</td>
<td>Speed of sound</td>
<td>cm s$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>alpha</td>
<td>Coefficient of isobaric expansivity</td>
<td>K$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>beta</td>
<td>Coefficient of isothermal compressibility</td>
<td>bar$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>diel</td>
<td>Dielectric constant</td>
<td>dimensionless</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>visc</td>
<td>Dynamic viscosity</td>
<td>g cm$^{-1}$ s$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>tcond</td>
<td>Thermal conductivity</td>
<td>cal cm$^{-1}$ s$^{-1}$ K$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>tdiff</td>
<td>Thermal diffusivity</td>
<td>cm$^2$ s$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>Prndtl</td>
<td>Prandtl number</td>
<td>dimensionless</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>visck</td>
<td>Kinematic viscosity</td>
<td>cm$^2$ s$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>albe</td>
<td>Isochoric expansivity</td>
<td>bar K$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>ZBorn</td>
<td>Z Born function</td>
<td>dimensionless</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>YBorn</td>
<td>Y Born function</td>
<td>K$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>QBorn</td>
<td>Q Born function</td>
<td>bar$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>daldT</td>
<td>Isobaric temperature derivative</td>
<td>K$^{-2}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>XBorn</td>
<td>X Born function</td>
<td>K$^{-2}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>NBorn</td>
<td>N Born function</td>
<td>bar$^{-2}$</td>
<td>*</td>
<td>NA</td>
</tr>
<tr>
<td>UBorn</td>
<td>U Born function</td>
<td>bar$^{-1}$ K$^{-1}$</td>
<td>*</td>
<td>NA</td>
</tr>
<tr>
<td>V</td>
<td>Volume</td>
<td>cm$^3$ mol$^{-1}$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>rho</td>
<td>Density</td>
<td>kg cm$^3$</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Psat</td>
<td>Saturation vapor pressure</td>
<td>bar</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>E</td>
<td>Isobaric expansivity</td>
<td>cm$^3$ K$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
<tr>
<td>kT</td>
<td>Isothermal compressibility</td>
<td>cm$^3$ bar$^{-1}$</td>
<td>NA</td>
<td>*</td>
</tr>
</tbody>
</table>
water

de\text{.}dT \quad \text{Temperature derivative of dielectric constant} \quad K^{-1} \quad * \quad \text{NA}
de\text{.}dP \quad \text{Pressure derivative of dielectric constant} \quad \text{bar}^{-1} \quad * \quad \text{NA}
P \quad \text{Pressure} \quad \text{bar} \quad * \quad \text{NA}

water\text{.}props \text{ returns the names of the available properties listed in this table, reflecting the current setting of thermo$\text{opt}$\text{water}.

water\text{.}SUPCRT92 \text{ interfaces to the FORTRAN subroutine taken from the SUPCRT92 package (H2O92D.F) for calculating properties of water. These calculations are based on data and equations of Levelt-Sengers et al., 1983, Haar et al., 1984, and Johnson and Norton, 1991, among others (see Johnson et al., 1992). A value of } P \text{ set to 'psat' refers to one bar below 100 °C, otherwise to the vapor-liquid saturation pressure at temperatures below the critical point ('psat' is not available at temperatures above the critical point). water\text{.}SUPCRT92 \text{ provides a limited interface to the FORTRAN subroutine; some functions provided there are not made available here (e.g., using variable density instead of pressure, or calculating the properties of steam). The properties of steam in CHNOSZ, as in SUPCRT92, are calculated using general equations for crystalline, gaseous and liquid species (cg1). The IAPWS-95 formulation also has provisions for computing the properties of steam, but these are currently not used by CHNOSZ.}

water\text{.}IAPWS95 \text{ is a wrapper (a function of temperature and pressure) around IAPWS95 (a function of temperature and density), rho.IAPWS95 and water.AW90. rho.IAPWS95 \text{ implements a root-finding technique (using uniroot) to determine the values of density for the given temperature and pressure. (Note that the 'P' property in water\text{.}IAPWS95 is calculated by first computing a density with rho.IAPWS95 for the given T and P, then calculating the P given the T and rho.) water.AW90 \text{ provides values of the static dielectric constant (diel) calculated using equations given by Archer and Wang, 1990. water.IAPWS95 \text{ contains routines for calculating the Born functions as finite-difference derivatives of the static dielectric constant with respect to temperature and pressure. For compatibility with geochemical modeling conventions, the values of Gibbs energy, enthalpy and entropy output by IAPWS95 are converted by water.IAPWS95 to the triple point reference state adopted in SUPCRT92 (Johnson and Norton, 1991; Helgeson and Kirkham, 1974).}

The stated temperature limits of validity of calculations in water\text{.}SUPCRT92 \text{ are from the greater of 0 °C or the melting temperature at pressure, to 2250 °C (Johnson et al., 1992). Valid pressures are from the greater of zero bar or the melting pressure at temperature to 30000 bar (water\text{.}SUPCRT92). The present functions do not check these limits and will attempt calculations for any range of input parameters, but may return NA for properties that fail to be calculated at given temperatures and pressures and/or produce warnings or even errors when problems are encountered.}

Starting with version 0.9-9.4, a check for minimum pressure (in valTP function in H2O92D.f) has been bypassed so that properties of H2O can be calculated using water\text{.}SUPCRT92 at temperatures below the 0.01 °C triple point. A primary check is still enforced (Tbtm), giving a minimum valid temperature of 253.15 K.

Value

For water and water\text{.}SUPCRT92 a data frame the number of rows of which corresponds to the number of input temperature, pressure and/or density values. water.AW90 returns a numeric vector with length corresponding to the number of temperature values.
References


See Also

`uniroot` is the root finder used in water to back out values of the density (rho) from those of T and P when the `iapws` option is set in `thermo$opt$water`. Equations of state for species other than water are coded in `hkf` and `cgl`.

Examples

```r
## calculations along saturation curve
T <- seq(273.15, 623.15, 25)
# liquid density, from SUPCRT92
water("rho", T=T, P="Psat")
# values of the saturation pressure, Gibbs energy
water(c("Psat", "G"), T=T, P="Psat")
# derivatives of the dielectric constant (Born functions)
water(c("QBorn", "YBorn", "XBorn"), T=T, P="Psat")
# now at constant pressure
water(c("QBorn", "YBorn", "XBorn"), T=T, P=2000)

## comparing the formulations
T <- convert(c(25, 100, 200, 300), "K")
# use IAPWS-95 (experimental for now)
thermo$opt$water <- "IAPWS95"
```
water(water.props(), T=T)
# use SUPCRT92 (the default)
thermo$opt$water <- "SUPCRT92"
water(water.props(), T=T)

## functions of temperature, density
# calculate density at 500 K, 500 bar
rho <- rho.IAPWS95(T=500, P=500)
# calculate pressure (~ 50 MPa) at this density
IAPWS95("P", T=500, rho=rho)
# calculate dielectric constant
water.IAW90(T=500, rho=rho, P=50)

## calculating Q Born function
# after Table 22 of Johnson and Norton, 1991
thermo$opt$water <- "SUPCRT92"
T <- rep(c(375, 400, 425, 450, 475), each=5)
P <- rep(c(250, 300, 350, 400, 450), 5)
w <- water("QBorn", T=convert(T, "K"), P=P)
# the rest is to make a neat table
w <- as.data.frame(matrix(w[[1]], nrow=5))
colnames(w) <- T[1:5]+5
rownames(w) <- P[1:5]
print(w)

---

**Gibbs Energy Minimization by Steepest Descent**

**Description**

Find the quantities of chemical species, subject to constant elemental bulk composition of the system, that minimize the Gibbs energy of the system.

**Usage**

```r
wjd(  
  A = matrix(c(  
    1,2,2,0,0,1,0,0,0,1,  
    0,0,0,1,2,1,1,0,0,0,  
    0,0,1,0,0,0,1,1,2,1),ncol=3,  
    dimnames=list(NULL,c("H","N","O"))),  
  G0.RT = c(  
    -10.021,-21.096,-37.986,-9.846,-28.653,  
  Y = c(0.1,0.35,0.5,0.1,0.35,0.1,0.1,0.1,0.1,0.1),  
  P = 51,  
  nlambda = 101,  
  imax = 10,  
  Gfrac = 1e-7
)```


```r
element.potentials(w, plot.it=FALSE, iplot=1:ncol(w$A))
is.near.equil(w, tol=0.01, quiet=FALSE)
guess(
  A = matrix(c(
    1,2,0,0,1,0,0,0,1,
    0,0,1,2,1,1,0,0,0,
    0,0,1,0,0,1,1,2,1),ncol=3,
    dimnames=list(NULL,c("H","N","O"))),
  B = c(2,1,1), method="stoich", minX = 0.001, iguess = 1, ic = NULL
)
run.wjd(ispecies, B = NULL, method = "stoich", Y = run.guess(ispecies, B, method),
P=1, T=25, nlambda=101, imax = 10, Gfrac = 1e-7, tol = 0.01)
run.guess(ispecies, B = NULL, method = "stoich", iguess = NULL)
equil.potentials(w, tol=0.01, T=25)
```

**Arguments**

- **A**
  matrix, chemical formulas of the species (elements on columns)

- **G0/RT**
  numeric, the Gibbs energies / RT, at a single temperature (length equal to number of species)

- **Y**
  numeric, initial solution, a positive set of values (numbers of moles, length equal to number of species)

- **P**
  numeric, pressure in atmospheres

- **nlambda**
  numeric, number of values of fractional distance change (\(\lambda\)) tested at each step.

- **imax**
  numeric, maximum number of iterations

- **Gfrac**
  numeric, Gibbs energy change of system, as a fraction of total system energy in previous step, below which iterations will stop

- **w**
  list, output from `wjd`

- **plot.it**
  logical, make a plot?

- **iplot**
  numeric, which elements for which to make plots

- **tol**
  numeric, maximum difference in chemical potentials that counts as equilibrium

- **quiet**
  logical, don’t output messages?

- **B**
  numeric, numbers of moles of the elements

- **method**
  character, method used for generating an initial solution

- **minX**
  numeric, minimum mole number for ‘central’ method

- **iguess**
  numeric, which guess to return

- **ic**
  numeric, which combination(s) of variable species to use (NULL for all)

- **ispecies**
  numeric, species indices (rownumbers of `thermoSobigt`)

- **T**
  numeric, temperature in °C
Details

wjd implements the steepest descent algorithm for Gibbs energy minimization in a closed system described by White et al., 1958. “Gibbs energy” (G) referred to here is the same as the “free energy” (F) used by White et al., 1958. wjd itself is independent of other functions or datasets in CHNOSZ, but run.wjd and run.guess are provided to make access to the thermodynamic database in CHNOSZ easier.

The default values of $A$, $G\Theta$.RT, $Y$ and $P$ correspond to the example problem described by White et al., 1958 for gases in the H, N, O system at 3500 K. Note that for this, and for any other equilibrium problem that can be simulated using this function, the mole quantities in $Y$ must all be positive numbers. Operationally, this vector defines not only the “initial solution” but also the bulk composition of the system; it is not possible to define the bulk composition using mole numbers of elements alone. The dimnames attribute in the default value for $A$ gives the names of the elements; this attribute is not necessary for the function to operate, but is used in the examples below to help label the plots.

White et al., 1958 describe in detail the computation of the direction of steepest descent by means of Lagrange multipliers. They propose an iterative solution to the energy minimization problem, where at each step the mole numbers of species are recomputed and a new steepest descent direction calculated from there. However, the authors only give general guidelines for computing the value of $\lambda$, that is, the fraction of the total distance the system actually moves in the direction of steepest descent from the current point at each iteration. The two constraints given for determining the value of $\lambda$ are that all mole numbers of species are positive, and the Gibbs energy of the system actually decreases (the minimum point is not passed). In the code described here, at each iteration the minimum value of lambda, not exceeding unity, that violates the first condition is computed directly (a value of one is assigned if the mole numbers remain positive through the entire range). With the default setting of nlambda, 101 values of $\lambda$ at even intervals from 0 to this maximum permissible value are tested for the second condition at each iteration, and the highest conforming value is selected. If a value of 0 occurs, it means that the algorithm has reached an endpoint independently of the iteration and convergence tests ($\rho$ and $Gfr$; see below). If this occurs, the value of nlambda might have to be increased depending on the user's needs.

The number of iterations is controlled by imax and Gfrac. The maximum number of iterations is set by imax; it can even be zero, though such a setting might only be useful in combination with element.potentials to characterize the initial state of a minimization problem. Within the limit of imax, the iterations continue until the magnitude of the change in total Gibbs energy of the system, as a fraction of the system’s energy in the previous iteration, is lower than the value specified in Gfrac. For the first example below, the default setting of Gfrac causes the algorithm to stop after six iterations.

Using the output of wjd, provided in the argument w, element.potentials calculates the chemical potentials of the elements in the system. It does so by combining the values of $G\Theta$.RT of species with the inverses of stoichiometric matrices of combinations of species whose elemental compositions are linearly independent from each other. These possible combinations are constructed using the function invertible.combs. The value returned by element.potentials is a matrix, with each column corresponding to a different element and each row to a different combination of species. The entries in the matrix are the chemical potentials of the elements divided by $RT$. If plot.it is set to TRUE, the chemical potentials of the elements are plotted as a function of species combination number, with as many plots as elements, unless iplot is set to another value (e.g. ‘c1, 3’) for only elements 1 and 3). In the first example below, the number of unique combinations of species is 120, but only 76 of these combinations provide stoichiometric independence.
There is no guarantee that the algorithm will converge on a global (or even be close to a local) minimum. However, some tests are available to help assess the likelihood that a solution is close to equilibrium. A necessary condition of equilibrium is that the chemical potentials of the elements be independent of the particular combination of species used to compute them. is.near.equil compares the chemical potentials for each element, computed using element.potentials, with the value of tol. If, for each element, the range of potentials/RT (difference between minimum and maximum) is less than \(\text{tol} \times \frac{\Delta S}{T} \) is TRUE, otherwise the function returns FALSE, and prints a message unless quiet is TRUE. The default value of \(\text{tol} \times \frac{\Delta S}{T} \) corresponds to an energy of 0.01 * 1.9872 * 298.15 = ca. 6 cal/mol at 25 °C.

One of the constraints of the algorithm coded in \textit{wjd} is that the initial solution, and all iterations, require positive (non-zero) numbers of moles of every species. Often, when investigating an equilibrium problem, the stoichiometric constraints are expressed most readily in terms of bulk composition – numbers of moles of each element. guess is a function to make initial guesses about the numbers of moles of all species in the system subject to the positivity constraints. Its system-specific arguments are the stoichiometric matrix \(a\) (as defined above for \textit{wjd}) and the bulk composition vector \(b\), giving the number of moles of each element, in the same order that the elements appear in \(a\). The first method available in \textit{guess} generates the ‘central’ solution of the system of linear equations using the \textit{xranges} function from \textit{limSolve}. The central solution is the mean of ranges of unknowns. The inequality constraint, or minimum number of moles of any species, is given by \(\text{min}X\).

The second method for \textit{guess} ‘stoich’ (and the default for \textit{run.guess} and \textit{run.wjd}) is to test successive combinations of species whose elemental compositions are linearly independent. The linearly independent combinations tested are all those from \textit{invertible.combs} if \textit{ic} is NULL, or only those identified by \textit{ic}. Each combination is referred to as the ‘variable’ species; the moles of all ‘other’ species are set to a single value. This value is determined by the constraint that the greatest proportion, relative to the bulk composition in \(b\), of any element contributed by all the ‘other’ species is equal to a value in \textit{max.frac} (see code). The function tests nine hard-coded values of \(\text{max.frac}\) from 0.01 to 0.99, at each one solving for the moles of the ‘variable’ species that make up the difference in numbers of moles of elements. If the numbers of moles of all the ‘variable’ species is positive, the guess is accepted. The first accepted guess is returned if \textit{iguess} is 1 (the default); other values of \textit{iguess} indicate which guess to return. If \textit{iguess} is NULL, all results are returned in a list, with non-successful guesses indicated by NA. In the first example below, of the 76 combinations of species whose elemental compositions are linearly independent, 32 yield guesses following this method.

\textit{run.wjd} is a wrapper function to \textit{run.wjd}, provided the \textit{ispecies} in the thermodynamic database (\textit{thermo$\text{dbigt}$}), the chemical composition of the system in \(b\), and the pressure \(P\) and temperature \(T\); the latter are passed to \textit{subcrt} (with exceed.Ttr = TRUE) to generate the values of \$G_0$.RT for \textit{wjd}. Alternatively to \(b\), the initial guess of numbers of moles of species can be provided in \(Y\); otherwise as many combinations of \(Y\) as returned from \textit{run.guess} are tested until one is found that \textit{is.near.equil}. The function gives an error if none of the guesses in \(Y\) is near equilibrium, within the tolerance set by \textit{tol}.

\textit{run.guess} is a wrapper function to call \textit{guess} using the stoichiometric matrix \(a\) built from the \textit{ispecies} indices in the thermodynamic database.

\textit{equil.potentials} returns the average (\textit{colMeans}) of \textit{element.potentials(w)}, or NULL if \textit{is.near.equil(w, tol=tol)} is FALSE. The output of this function can be used as the \textit{emu} argument for \textit{basis.logact} to calculate the corresponding activities of the basis species.
Value

wjd returns a list with the problem definition and results: elements A, G0, RT, Y, and P are as supplied
in the arguments; the results are in X (final mole numbers of species), F, Y (Gibbs energy of the
system at initial conditions and after each iteration), lambda (value used for λ at each iteration),
and elements (matrix of moles of elements at initial conditions and after each iteration; iterations
on the columns and elements on the rows). The latter result is provided to assist in checking mass
balance (mostly for debugging more than theoretical reasons).

References


See Also

invertible.combs, used by element.potentials to find combinations of species that are compositionally independent.

Examples

```r
## run the function with default settings to reproduce
## the example problem in White et al., 1958
w <- wjd()
# the mole fractions are very close to those shown in the
# last column of Table III in the paper
print(w$x)
# the Gibbs energy of the system decreased,
# and by a smaller amount, at each iteration
print(diff(w$f$Y))
# there are 76 unique combinations of species that can be
# used to calculate the chemical potentials of the elements
stopifnot(nrow(invertible.combs(w$A)) == 76)
# what the scatter in those chemical potentials looks like
ep <- element.potentials(w, plot.it=TRUE)
# the differences in chemical potentials / RT are all less than 0.01
is.near.equil(w) # TRUE

## run the algorithm for only one iteration
w <- wjd(imax=1)
# the scatter in chemical potentials is much greater
ep <- element.potentials(w, plot.it=TRUE)
# and we're pretty far from equilibrium
is.near.equil(w) # FALSE

## test all of the guesses of initial mole quantities
## provided by guess() using default bulk composition (H2NO)
# 9 of them are not is.near.equil with the tolerance lowered to 0.0001
sapply(1:32, function(i)
  is.near.equil(wjd(Y=guess(method = "stoich", iguess=i)), tol=0.0001))
```
## using run.jwd(): 20 crystalline amino acids

```r
#i a <- info(ami(noacids(""), "cr")
# starting with one mole of each amino acid
w <- run.jwd(i a, Y=rep(1, 20), imax=20)
# the following is TRUE (FALSE if tol is left at default)
is.near.equilibrium(w, tol=0.012)
# in this assemblage, what are the amino acids
# in order of increasing abundance?
ami(noacids()[order(w))] 
# because the elements are redistributed among the species,
# the total number of moles of species does not remain constant
sum(w) # <20
```

## run.jwd with proteins: cell periphery of yeast

```r
# get the proteins in the requested location
y <- yeastGFP("cell.periphery")
# get the amino acid compositions of the proteins
aa <- more.aa(y$protein, "Sce")
# don't use those with NA abundance or sequence
ina <- is.na(y$abundance) | is.na(aa$chains)
aa <- aa[!ina,]
# let's try normalizing the proteins to single residues
# columns 6:25 are the actual amino acid counts
aa.625 <- aa[, 6:25]
aa[, 6:25] <- aa.625 / rowSums(aa.625)
# add proteins to thermo$protein
add.protein(aa)
# add proteins to thermo$obigt
iobigt <- info(paste(aa$protein, aa$organism, sep="_"))
# use equal initial abundances, with total equal to yeastGFP abundances
Y <- rep(mean(y$abundance[!ina]), length(y$abundance[!ina]))
# run the Gibbs energy minimization
w <- run.jwd(iobigt, Y=Y, imax=100)
# make a log-log plot
plot(log10(y$abundance[!ina]), log10(w), xlim=c(1.5, 5), ylim=c(1.5, 5))
# get the element potentials (tolerating "close enough" to equilibrium)
emu <- equi.potentials(w, tol=1e7)
# then the logarithms of activities of the basis species
basis("CHNOS")
bl <- basis.logact(emu)
# make a title and legend
```
```r
title(main="calculated vs observed abundances: yeast cell periphery")
basis(names(bl), bl)
legend("topleft", describe.basis(digits=2))
```
Index

*Topic datasets
  extdata, 39
  thermo, 94
*Topic extra
  anim.TCA, 12
  EOSregress, 30
eqdata, 33
examples, 38
transfer, 101
wjd, 139
*Topic package
  CHNOSZ-package, 3
*Topic primary
  affinity, 6
  basis, 14
  diagram, 19
  findit, 45
  info, 49
  revisit, 76
  species, 82
  subcrt, 84
*Topic protein
  ionize.aa, 51
  iprotein, 54
  more.aa, 59
  protein, 63
  protein.info, 67
  read.expr, 71
*Topic secondary
  buffer, 16
eos, 26
equilbrate, 35
IAPWS95, 47
makeup, 57
objective, 60
sideeffects, 80
swap.basis, 91
taxonomy, 92
water, 135

*Topic util
  util'affinity, 103
  util.args, 105
  util.array, 106
  util.blast, 108
  util.character, 111
  util.data, 112
  util.expression, 116
  util.fasta, 119
  util.formula, 122
  util.list, 124
  util.matrix, 125
  util.misc, 126
  util.plot, 128
  util.program, 131
  util.seq, 132
  util.units, 133
  .onAttach, 80, 95
[. 107

A.ionization, 52
A.ionization (util'affinity), 103
aa2eos (iprotein), 54
aaasum (iprotein), 54
add.obigt, 42, 95, 100
add.obigt (util.data), 112
add.protein, 120
add.protein (iprotein), 54
affinity, 3, 5, 6, 14, 17, 20–22, 35, 38, 52,
  69, 81, 82, 86, 99, 105, 107, 121,
  131, 134
agrep, 50
all.equal, 69
allparents (taxonomy), 92
aminoacids (util.seq), 132
anim (anim.TCA), 12
anim.TCA, 12, 42, 107, 115
apc, 63
apc (transfer), 101
array, 78, 107
as.chemical.formula, 114
as.chemical.formula (util.formula), 122
as.expression, 31
as.numeric, 122
assign, 80
attach, 80
attributes, 62
axis.label (util.expression), 116
balance, 20, 22
balance (equilibrate), 35
barchart, 130
basis, 3, 7, 8, 14, 17, 46, 58, 68, 81–83, 85, 86, 91, 99
basis.logact, 142
basis.logact (swap.basis), 91
basis.matrix (swap.basis), 91
browse.refs, 96
browse.refs (util.data), 112
browseURL, 113
buffer, 4, 5, 7–9, 15, 16, 17, 23, 98
buffers (thermo), 94
c2s (util.character), 111
caller.name (util.program), 131
can.be.numeric (util.character), 111
cgl, 3, 85, 98, 137, 138
cgl (eos), 26
caller (character), 131
check.obigt, 43, 50
check.obigt (util.data), 112
checkEOS, 50, 95
checkEOS (util.data), 112
checkGHS, 50, 95
checkGHS (util.data), 112
CHNOSZ-package, 3
colMeans, 142
colSums, 107
contour, 77
convert, 81, 104
convert (util.units), 133
count.aa, 131, 133
count.aa (util.fasta), 119
count.charge (makeup), 57
count.elements (makeup), 57
count.formulas (makeup), 57
CV (objective), 60
CVRMSD (objective), 60
data, 80, 113
data.frame, 5, 95
DDGmix (objective), 60
def2gi, 109
def2gi (util.blast), 108
demo, 38
demos, 22, 23, 46, 87
demos (examples), 38
describe.basis (util.expression), 116
describe.property (util.expression), 116
describe.reaction (util.expression), 116
DGinf (objective), 60
eDGmix (objective), 60
DGtr (objective), 60
diagram, 4, 5, 7, 14, 19, 37, 38, 42, 45, 69, 115, 117, 127, 134
dimSums (util.array), 106
diversity, 78
dPdTtr, 85
dPdTtr (util.misc), 126
draw.transfer (transfer), 101
E.units, 85, 86
E.units (util.units), 133
element (thermo), 94
element.mu (swap.basis), 91
element.potentials (wjd), 139
energy, 9
energy (util.affinity), 103
energy.args, 9
entropy, 58, 96, 123
entropy (util.formula), 122
envert (util.units), 133
environment, 80
eos, 26
eos.args (util.args), 105
EOScalc (EOSregress), 30
EOScoeffs (EOSregress), 30
EOSlab (EOSregress), 30
EOSplot (EOSregress), 30
EOSregress, 30, 41
EOStvar (EOSregress), 30
eqdata, 33
equilibrium.boltzmann, 4, 131
equilibrium.boltzmann (equilibrate), 35
equilibrium.potentials (wjd), 139
equilibrium.reaction, 4, 131
equilibrium.reaction (equilibrate), 35
equilibrate, 9, 20, 21, 35, 76, 77
INDEX

example, 38
examples, 4, 38
expr.property (util.expression), 116
expr.species (util.expression), 116
expr.units (util.expression), 116
expression, 117, 130
extdata, 13, 39, 56, 60, 72, 94, 110, 114, 115
extremes, 62
extremes (revisit), 76

feldspar (transfer), 101
find.tp (diagram), 19
findit, 4, 38, 45, 60, 62, 78

get, 31, 80
get.formula (util.formula), 122
get.objfun, 62
get.objfun (objective), 60
getnames (taxonomy), 92
getnodes (taxonomy), 92
getrank (taxonomy), 92
gfun (eos), 26
GHS, 81
GHS (util.formula), 122
grep, 120
grep.file, 131
grep.file (util.fasta), 119
grid, 85
group.formulas, 123
group.formulas (protein.info), 67
groups (thermo), 94
guess (wjd), 139

heat.colors, 13, 21
help.start, 4
hkf, 3, 85, 87, 97, 138
hkf (eos), 26

i2A, 58, 114
i2A (util.formula), 122
IAPWS95, 47, 137
id.blast, 40, 42
id.blast (util.blast), 108
image, 77
info, 15, 29, 49, 81, 83, 85, 87, 114, 115
install.packages, 4
interactive, 131
invertible.combs, 141–143
invertible.combs (util.matrix), 125

invisible, 22, 110, 114
ionize.aa, 41, 51, 63, 68, 69
ip2aa, 50, 52, 68
ip2aa (iprotein), 54
iprotein, 50, 54, 63, 68, 69, 99
is.near.equilibrium (wjd), 139

label.plot (util.plot), 128
lapply, 131
legend, 21
LETTERS, 117
letters, 117
library, 4, 131
list, 80, 95
list2array (util.array), 106
lm, 31, 32
loess.smooth, 77
log10, 36
logact, 77
logQP, 36
makeCluster, 131
makeup, 3, 15, 28, 57, 87, 95, 113, 122, 123, 127
mass, 58, 68, 96
mass (util.formula), 122
message, 131
mod.basis (basis), 14
mod.buffer, 113, 115
mod.buffer (buffer), 16
mod.obigt, 50, 81
mod.obigt (util.data), 112
more.aa, 41, 51, 59, 63, 73
MP90.cp (protein.info), 67
msgout (util.program), 131
mtext, 130
mtitle (util.plot), 128

names, 22
nonideal, 86
nonideal (util.misc), 126
nucleic.complement (util.seq), 132
nucleic.formula, 121
nucleic.formula (util.seq), 132

OBIGT (thermo), 94
obigt2eos (util.data), 112
objective, 46, 60, 76, 77
opt (thermo), 94
optimal.index, 46, 62
optimal.index (revisit), 76
outvert (util.units), 133

P.units, 8, 85
P.units (util.units), 133
palply, 37
palply (util.program), 131
par, 21, 128, 130
parent (taxonomy), 92
parLapply, 131
paste, 128
pearson (objective), 60
plot.findit (findit), 45
plot.new, 130
plotmath, 31, 117
png, 38
pprod (util.list), 124
preset.basis (basis), 14
preset.logact (basis), 14
protein, 3, 18, 23, 40, 50, 56, 63, 69
protein.basis, 52
protein.basis (protein.info), 67
protein.equil (protein.info), 67
protein.formula, 123, 133
protein.formula (protein.info), 67
protein.info, 42, 52, 56, 63, 67
protein.length, 131
protein.length (protein.info), 67
put.basis (basis), 14

qdnorm, 61, 77
qqr (objective), 60

rainbow, 22
read aa (iprotein), 54
read.blast, 40
read.blast (util.blast), 108
read.expr, 40, 41, 55, 56, 60, 63, 71
read.fasta, 3, 56, 131
read.fasta (util.fasta), 119
readLines, 120
refs (thermo), 94
residualsplot (util.plot), 128
revisit, 4, 37, 41, 46, 60, 62, 76, 121, 130, 131
RH2obigt, 43
RH2obigt (util.data), 112

rho.IAPWS95 (water), 135
richness (revisit), 76
RMSD (objective), 60
rowSums, 107
Rprofile, 131
run.guess (wjd), 139
run.wjd, 123
run.wjd (wjd), 139

s2c (util.character), 111
sciname (taxonomy), 92
SD (objective), 60
seq2aa (iprotein), 54
shannon (objective), 60
sideeffects, 80, 95
slice (util.array), 106
spearman (objective), 60
species, 3, 7, 8, 14, 15, 46, 55, 81, 82, 83, 99, 102

splinefun, 69
state.args (util.args), 105
stopifnot, 4, 87
stress (read.expr), 71
strip (diagram), 19
structure, 62
subcrt, 3, 5, 7, 8, 27–29, 41, 50, 52, 55, 58, 81, 84, 104–106, 114, 118, 134, 142

substitute, 31
subtr, 120, 128
swap.basis, 15, 91
Sweave, 131
system, 12

T.units, 8, 81, 84, 85
T.units (util.units), 133
taxonomy, 42, 92
test_that, 91, 131
text, 128, 130
thermo, 3, 4, 7, 8, 14, 16, 17, 28, 31, 39, 42, 43, 49, 50, 54, 55, 58, 60, 80, 82, 85, 94, 104, 113, 114, 117, 120, 122, 123, 126, 136, 138, 142
thermo.axis (util.plot), 128
thermo.plot.new, 81
thermo.plot.new (util.plot), 128
title, 21
today (util.data), 112
tolower, 128
TP.args (util.args), 105
INDEX

transfer, 4, 101, 104, 127
Ttr (util.misc), 126

uniprot.aa, 56
uniprot.aa (util.fasta), 119
unirroot, 37, 137, 138
unitize (util.misc), 126
util.affinity, 103
util.args, 105
util.array, 106
util.blast, 108
util.character, 111
util.data, 112
util.expression, 116
util.fasta, 41, 119
util.formula, 122
util.list, 124
util.matrix, 125
util.misc, 126
util.plot, 128
util.program, 131
util.seq, 132
util.units, 46, 133

water, 28, 29, 31, 41, 81, 85, 95, 99, 104, 106, 135
water.IAPWS95, 48
water.lines (util.plot), 128
which, 22
which.balance, 36
which.balance (util.misc), 126
which.pmax (util.list), 124
with, 80
wjd, 139
WP02.auxiliary (IAPWS95), 47
write.blast, 40
write.blast (util.blast), 108

xranges, 142

yeastgfp, 40
yeastgfp (read.expr), 71

ZC, 68
ZC (util.formula), 122