

Package ‘CHNOSZ’

August 23, 2011

Title Chemical Thermodynamics and Activity Diagrams

Version 0.9-7

Date 2011-08-23

Author Jeffrey M. Dick <jmdick@asu.edu>

Maintainer Jeffrey M. Dick <jmdick@asu.edu>

Depends R (>= 2.10.0), utils

Suggests xtable

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/>

Repository CRAN

Date/Publication 2011-08-23 05:10:01

R topics documented:

CHNOSZ-package	3
affinity	6
anim.TCA	12
basis	14
buffer	16
diagram	20
eos	29
EOSregress	32
eqdata	35
equil	37
examples	38
extdata	39
findit	44
get.expr	47
get.protein	48
info	54
ionize	56
makeup	60
protein	62
revisit	68
species	71
subcrt	73
taxonomy	80
thermo	82
transfer	89
util.args	92
util.array	93
util.blast	95
util.character	97
util.data	99
util.fasta	102
util.formula	104
util.list	107
util.misc	108
util.plot	112
util.seq	115
util.stat	116
util.units	117
water	119

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](#), [cgl](#)), [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.react](#), [equil.boltz](#)).
- 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. You should receive four warning messages when the examples are finished; don't worry, they are an expected result.
- 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.
- To learn how to update the thermodynamic database, look at its documentation in [thermo](#).

Compatibility

The package depends on 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). However, without accessing the compressed data files in `extdata` it should be possible to run CHNOSZ under R versions 2.4.0 or greater (availability of the `'stringsAsFactors'` argument to `data.frame`).

Acknowledgements

This package was made possible by the fearless leadership of the late Professor Harold C. Helgeson. He took no heed of disciplinary boundaries, and encouraged the author to pursue unconventional problems. Hal, and those in his research group, are in some way responsible for many parts of this package, such as the aqueous equations of state used here, the thermodynamic parameters for many species, and actual computer code (`H2O92D.f`, borrowed from the SUPCRT92 program). Work on this project at U.C. Berkeley (through 2008) was supported by research grants from the U.S. National Science Foundation and Department of Energy. In 2009–2011, research projects involving this package were supported by the National Science Foundation under grant EAR-0847616 (<http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0847616>).

Examples

```

### Getting Started

## standard thermodynamic properties of species
subcrt("H2O")
subcrt("alanine")
# names of proteins have an underscore
subcrt("LYSC_CHICK")
# custom temperature range
T <- seq(0,500,100)
subcrt("H2O",T=T,P=1000)
# temperature - pressure grid
P <- seq(1000,4000,1000)
subcrt("H2O",T=T,P=P,grid="P")

## information about species
# query the database using formulas
info("C6H12O6")
info("SiO2")
# query using names
info("quartz")
si <- info(c("glucose","mannose"))
# show the equations of state parameters
info(si)
# approximate matches for names or formulas
info("acid ")
info("SiO2 ")

## standard thermodynamic properties of reactions
# fermentation example
info(c("fructose","ethanol"))
subcrt(c("fructose","C2H5OH","CO2"),c(-1,2,2))
# weathering example -- also see transfer()
subcrt(c("k-feldspar","H2O","H+","kaolinite","K+","SiO2"),
  c(-2,-1,-2,1,2,4))
# partial reaction auto-completion is possible
basis(c("SiO2","H2O","K+","H+","O2"))
subcrt(c("k-feldspar","kaolinite"),c(-2,1))

## chemical affinities
# basis species
basis(c("CO2","H2O","O2"))
# species of interest
species(c("CH4","C2H4O2","CO2"))
# chemical affinities of formation reactions
# take off $values for complete output
affinity()$values
affinity(O2=c(-90,-60,5))$values

```

Description

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.

Usage

```
affinity(..., property=NULL, sout=NULL, do.phases=FALSE,
  return.buffer=FALSE, balance="PBB", quiet=FALSE,
  iprotein=NULL, loga.protein=-3)
energy(what, vars, vals, lims, T=thermo$opt$Tr, P="Psat", IS=0,
  sout=NULL, do.phases=FALSE, transect = FALSE)
energy.args(args, quiet = FALSE)
```

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.
return.buffer	logical. If TRUE, and a <code>buffer</code> 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 <code>buffer</code> to identify a conserved basis species (or 'PBB') in a chemical activity buffer. Default is 'PBB'.
iprotein	numeric, indices of proteins in <code>thermo\$protein</code> for which to calculate properties.
loga.protein	numeric, logarithms of activities of proteins identified in <code>iprotein</code> .
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 <code>thermo\$opt\$Tr</code> , which corresponds to 25 °C.
P	numeric, pressure, or character "Psat" (default), which denotes 1 bar or the saturation vapor pressure of H ₂ O above 100 °C (see water).

IS	numeric, ionic strength; default is 0 mol kg ⁻¹ .
sout	list, output from subcrt function.
do.phases	logical, allow subcrt to compute properties for stable phases?
transect	logical, perform calculations on a transect instead of a grid?
args	list, defines the variables over which to calculate properties.
quiet	logical, print fewer messages to the screen?

Details

The user calls `affinity` in order to calculate the chemical affinities of formation reactions of species of interest. This function itself calls `energy.args` and `energy` to perform the calculations (the user normally should not need to interact with either of these functions). The calculation of chemical affinities of formation reaction relies on the global definitions of the `basis` species and `species` of interest. After calculating the affinities, the user can go on to generate activity diagrams using `diagram` or to use them in other calculations.

The chemical affinity quantifies the driving force for a reaction to proceed in a forward or reverse direction. The expression for chemical affinity A is $A=RT \ln(K/Q)$ (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, ΔG_r° , by $\Delta G_r^\circ = -2.303RT \log K$, 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 $K = Q$.

`energy` is the workhorse of the `affinity` calculations. Given n (which can be zero, one, or more) names of basis species and/or 'T', 'P', or '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 'IS' are not among the `vars`, their constant values can be supplied in T (in Kelvin), P (in bar, or 'P_{sat}'), and 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`). `do.phases` is passed to `subcrt` so that the properties of stable mineral phases as a function of temperature can optionally be calculated.

`energy.args` is used by `affinity` to generate the argument list for `energy`. `affinity` passes the ... list as `args` to `energy.args`, which returns an argument list appropriate for `energy`. `energy.args` also has the job of converting 'Eh' to 'pe' as a temperature-dependent function (see `convert`), and converting 'pe' and 'pH' to logarithms of activities of the electron and protein, respectively (i.e., negating the values).

The `property` argument to `affinity` is analogous to the `what` argument of `energy`. 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 $c(\min, \max, \text{res})$ 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 `nuts` (on program start-up these are °C and bar, respectively).

If proteins are in the species definition (which are distinguished from other species by having an underscore character in the name), and the basis species are charged, and `thermooptionize` is TRUE, `affinity` calls `ionize` to calculate the properties of charged proteins. 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 global `species` definition. This approach takes advantage of the commutative properties of the protein group additivity algorithm (Dick et al., 2006), and 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`. The user can give the indices of desired proteins contained in `thermo$protein` and `affinity` will automatically add to the species list the amino acid *residues* and protein backbone group then calculate the properties of the reactions for the residues, and add them together to get those of the indicated proteins. The `iprotein` option is compatible with calculations for `ionized` 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`, `property` (name of the calculated property), `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, of length two (corresponding to minimum/maximum values) if either one is a variable of interest or length one otherwise), `xname` (the name of the first variable of interest, "" if none is present), `xlim` (if a first variable of interest is present, numeric of length 3 specifying the (minimum, maximum, resolution) of this variable), `yname` (the name of the second variable of interest, "" if none present), `ylim` (analogous to `xlim` but for a second variable of interest), `values`. The latter is itself a list, each element of which corresponds to a species of interest (names of the elements in this list are the character versions of the index of the species in `thermo$obigt`). The values for each species are a numeric value (if the number of variables of interest is zero) or an n -dimensional matrix otherwise. The values

of chemical affinity of formation reactions of the species of interest are returned in dimensionless (base 10) units (i.e., $A/2.303RT$).

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`.

If `pe` or `Eh`, or `pH` is/are among the variables of interest, `xnames` and/or `yname`s become 'e-' or 'H+' (respective to the property) and the minimum and maximum values in `xlim` and/or `ylim` are adjusted accordingly (using `convert`).

References

Amend, J. P. and Shock, E. L. (1998) Energetics of amino acid synthesis in hydrothermal ecosystems. *Science* **281**, 1659–1662. <http://dx.doi.org/10.1126/science.281.5383.1659>

Amend, J. P. and Shock, E. L. (2001) Energetics of overall metabolic reactions of thermophilic and hyperthermophilic Archaea and Bacteria. *FEMS Microbiol. Rev.* **25**, 175–243. [http://dx.doi.org/10.1016/S0168-6445\(00\)00062-0](http://dx.doi.org/10.1016/S0168-6445(00)00062-0)

Dick, J. M., LaRowe, D. E. and Helgeson, H. C. (2006) Temperature, pressure, and electrochemical constraints on protein speciation: Group additivity calculation of the standard molal thermodynamic properties of ionized unfolded proteins. *Biogeosciences* **3**, 311–336. <http://www.biogeosciences.net/3/311/2006/bg-3-311-2006.html>

Kondepudi, D. K. and Prigogine, I. (1998) *Modern Thermodynamics: From Heat Engines to Dissipative Structures*, John Wiley & Sons, New York, 486 p. <http://www.worldcat.org/oclc/38055900>

Schulte, M. D. and Shock, E. L. (1995) Thermodynamics of Strecker synthesis in hydrothermal systems. *Orig. Life Evol. Biosph.* **25**, 161–173. <http://dx.doi.org/10.1007/BF01581580>

See Also

For prerequisites to `affinity`, see `basis` and `species`. Some calculations may invoke `ionize` and `buffer`. To visualize the results, use `diagram`.

Examples

```
## 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, 10), P=2000)
```

```

# at 25 temperatures and pressures
affinity(T=c(500,1000,5),P=c(1000,5000,5))
# 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")

## Activity of glycine as a function of those of
## HCN and formaldehyde (200 C, 300 bar)
## After Schulte and Shock, 1995, Fig. 5
# we can define the basis as this:
basis(c("formaldehyde","H2O","HCN","O2"))
species("glycine")
a <- affinity(HCHO=c(-10,-2,9),HCN=c(-18,-2,9),T=200,P=300)
# that gave us *affinities* (dimensionless) for logact(glycine)=-3
# (the default). we can now find the *activities* that
# correspond to affinity=0
logact.glycine <- species()$logact + a$values[[1]]
# note transposition of the z-value matrix in the following command
contour(x=-10:-2,y=seq(-18,-2,by=2),z=t(logact.glycine),
  xlab=axis.label("HCHO"),ylab=axis.label("HCN"),
  labcex=1,xaxs="i",yaxs="i")
title(main=paste("log activity glycine, after Schulte and Shock, 1995",
  "200 deg C, 300 bar, logaH2O = 1",sep="\n"))

## 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(c("alanine","argininate","asparagine","aspartate","cysteine",
  "glutamate","glutamine","glycine","histidine","isoleucine",
  "leucine","lysiniun","methionine","phenylalanine","proline",
  "serine","threonine","tryptophan","tyrosine","valine"),
  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))
Tc <- 18
T <- convert(Tc,"K")
# converting A (dimensionless) to G of reaction (cal/mol)
# is like converting log K to standard G of reaction
AS98.18 <-
  convert(convert(as.numeric(affinity(T=Tc)$values),"G",T=T),"J")/1000
# the 100 degree C case
Tc <- 100
T <- convert(Tc,"K")
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-9,5.0e-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)))
AS98.100 <-
  convert(convert(as.numeric(affinity(T=Tc)$values),"G",T=T),"J")/1000
# the nominal carbon oxidation state
Z.C <- ZC(as.character(thermo$sobigt$formula[thermo$species$ispecies]))
# put them together
print(data.frame(T100=AS98.100,T18=AS98.18,Z.C=Z.C))
# values not exactly reproducing AS98 - different amino acid parameters
# forget species to run next example
species(delete=TRUE)

## affinities of metabolic reactions
## after Amend and Shock, 2001, Fig. 7
basis(c("CO2","H2","NH3","O2","H2S","H+"))
basis(c("O2","H2"),"aq") # O2 and H2 were gas
species("H2O")
doplot <- function(T) {
  res <- 20
  a <- affinity(H2=c(-10,0,res),O2=c(-10,0,res),T=T)
  T.K <- convert(T,"K") # temperature in Kelvin
  a <- convert(a$values[[1]],"G",T.K) # affinities (cal/mol)
  a <- convert(a,"J") # affinities (Joule)
  contour(x=seq(-10,0,length.out=res),
          y=seq(-10,0,length.out=res),z=t(a/1000),
          labcex=1,xlab=axis.label("H2"),ylab=axis.label("O2"))
}
layout(matrix(c(1,1,2,3,4,5),ncol=2,byrow=TRUE),heights=c(1,4,4))
T <- c(25,55,100,150)
par(mar=c(0,0,0,0))
plot.new()
text(0.5,0.1,paste(c("H2(aq) + 0.5O2(aq) = H2O(liq)\n\n",
                    "after Amend and Shock, 2001")),cex=2)
par(mar=c(3,3,0.5,0.5),cex=1.3,mgp=c(2,1,0))
for(i in 1:length(T)) doplot(T[i])
# this is so the plots in the next examples show up OK
layout(matrix(1))

## continuation of last example, now in three dimensions
print(affinity(H2=c(-10,0,3),O2=c(-10,0,3),T=c(25,150,4))$values)

## calculations on a transect
# suppose that temperature and oxygen fugacity both change in space
# (say from 1 to 6 meters), and we have six values for each but want to
# interpolate them to make a plot with smooth curves
T <- splinefun(1:6,c(0,25,30,40,55,75))(seq(1,5,length.out=100))
O2 <- splinefun(1:6,c(-90,-83,-78,-73,-68,-63))(seq(1,5,length.out=100))
# what system could this be?
basis("CHNOS+")
species(paste("CSG",c("METBU","METVO","METTL","METJA"),sep="_"))
# now pass T and O2 to affinity, which because their lengths

```

```
# are greater than three, treats them like coordinates for a
# transect through chemical potential space rather than
# the definition of a 2-dimensional grid
a <- affinity(T=T,O2=O2)
diagram(a,ylim=c(-4,-2))
title(main=paste("Computed abundances of surface-layer proteins",
  "as a function of T and logfO2",sep="\n"))
```

anim.TCA

Functions to Make Animations

Description

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

Usage

```
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>), 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 $\log a_{\text{H}_2\text{O}} - \log f_{\text{O}_2}$ 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(-20,0))` to draw a $\log a_{\text{H}_2\text{O}} - \log a_{\text{H}_2}$ diagram. The diagrams are made at 25 degrees C unless `high.T` is TRUE, in which case high-temperature (100 degrees 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 $\log a_{\text{O}_2}$ and $\log a_{\text{H}_2}$, at 25 degrees C. Unlike most other examples in CHNOSZ, the chemical potentials of hydrogen and oxygen in the system are represented by the activities of O_2 and H_2 , and H_2O is not used as a basis species. Therefore, the equilibrium activities of H_2O 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 $\log a_{\text{H}_2}$ 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 $\log a_{\text{H}_2(aq)} = -11 + 3/(40 \times T(^{\circ}\text{C}))$; this equation comes from a model of relative stabilities of proteins in a hot-spring environment (Dick and Shock, 2011).

References

Anderson, N. L. and Anderson, N. G. (2003) The human plasma proteome: History, character and diagnostic prospects (Vol. 1 (2002) 845-867). *Molecular and Cellular Proteomics* **2**, 50. <http://dx.doi.org/10.1074/mcp.A300001-MCP200>

Dick, J. M. and Shock, E. L. (2011) Calculation of the relative chemical stabilities of proteins as a function of temperature and redox chemistry in a hot spring. *PLoS ONE* **6**, e22782. <http://dx.doi.org/10.1371/journal.pone.0022782>

Examples

```
## 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)
```

basis

*Basis Species***Description**

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

Usage

```
basis(basis = NULL, values = NULL, values2=NULL, delete = FALSE,
      quiet = FALSE)
```

Arguments

basis	character, names or formulas of basis species, or numeric, indices of species (rownumbers of <code>thermo\$obigt</code>).
values	character or numeric, physical states or names of buffers, or logarithms of activities or fugacities.
values2	character or numeric, physical states, or logarithms of activities or fugacities.
delete	logical, delete the current basis species definition?
quiet	logical, suppress printing the basis definition?

Details

This is the first function of two (`species` is the second) used to define for the program the system of interest. 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 can, 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 this function with the names or formulas of the basis species in the first argument. If successful, `thermo$basis` is replaced with the new basis definition. If the second and/or third arguments are present: if either one is of type numeric it refers to values for logarithms of activities (or fugacities) of basis species identified in the first argument. If either of these arguments is character, it refers to the name of a state (if present in `thermo$obigt$state`) or to the name of a `buffer` (if present in `thermo$buffers$name`).

To update the logarithms of activities of basis species, provide their formulas or names in the first argument and the values in the second. 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.

If the value of `basis` is one of the keywords in the following table, the corresponding set of basis species is loaded, and their activities set to reference values useful for carrying out the examples. The basis species identified by these keywords are aqueous except for H_2O (liq), O_2 (gas) and Fe_2O_3 (hematite, cr1).

CHNOS	$\text{CO}_2, \text{H}_2\text{O}, \text{NH}_3, \text{H}_2\text{S}, \text{O}_2$
CHNOS+	$\text{CO}_2, \text{H}_2\text{O}, \text{NH}_3, \text{H}_2\text{S}, \text{O}_2, \text{H}^+$
CHNOSe	$\text{CO}_2, \text{H}_2\text{O}, \text{NH}_3, \text{H}_2\text{S}, e^-, \text{H}^+$
CHNOPs+	$\text{CO}_2, \text{H}_2\text{O}, \text{NH}_3, \text{H}_3\text{PO}_4, \text{H}_2\text{S}, e^-, \text{H}^+$
MgCHNOPs+	$\text{Mg}^{+2}, \text{CO}_2, \text{H}_2\text{O}, \text{NH}_3, \text{H}_3\text{PO}_4, \text{H}_2\text{S}, e^-, \text{H}^+$
FeCHNOS	$\text{Fe}_2\text{O}_3, \text{CO}_2, \text{H}_2\text{O}, \text{NH}_3, \text{H}_2\text{S}, \text{O}_2$
FeCHNOS+	$\text{Fe}_2\text{O}_3, \text{CO}_2, \text{H}_2\text{O}, \text{NH}_3, \text{H}_2\text{S}, \text{O}_2, \text{H}^+$

Set `delete` to `TRUE` to clear the current basis definition.

Value

Upon defining or changing the basis definition, the function returns the value of `thermo$basis`. With no arguments present, the function invisibly returns the square dataframe of stoichiometries of elements in the basis species (or `NULL` if `thermo$basis` does not exist). This return value is only the stoichiometric matrix contained in `thermo$basis`. If `basis` is numeric, it refers to indices of species (i.e., rownumbers of `thermo$obigt`). As long as the basis species were previously defined, the function in this case calculates and returns the basis stoichiometry(s) of the corresponding specie(s).

See Also

[info](#) to query the thermodynamic database in order to find what species are available. [species](#) is the usual next step after `basis`. [makeup](#) is used by `basis` to generate the stoichiometric matrix from chemical formulas.

Examples

```
## define basis species
basis("O2") # one element
basis(c("H2O","O2")) # two elements
basis(c("H2O","O2","H+"))

## Not run:
# marked dontrun because they produce errors
# fewer basis species than elements
basis(c("H2O","H+"))
# more basis species than elements
basis(c("H2O","O2","H2","H+"))
# non-independent basis species
basis(c("CO2","H2O","HCl","Cl-","H+"))
## End(Not run)
```

```

## specify activities and states
basis(c("H2O", "O2", "CO2"), c(-2, -78, -3), c("liq", "gas", "aq"))
# change logarithms of activities/fugacities
basis(c("H2O", "O2"), c(0, -80))
# change state of CO2
basis("CO2", "gas")

## get the basis stoichiometry of species
basis("CHNOS")
ispecies <- info(c("glutamic acid", "phenylalanine"))
basis(ispecies)

## changing basis species
b1 <- basis("CHNOS+")
# different basis species, same elements
b2 <- basis(c("HCO3-", "H2O", "NH4+", "HS-", "H2", "H+"))
# back to starting basis species
b3 <- basis(c("CO2", "H2O", "NH3", "H2S", "O2", "H+"))
stopifnot(all.equal(b1$logact, b3$logact))

## changing the basis species
basis("MgCHNOPS+")
# load species whose names are like "ATP"
species(info("ATP ")[[1]])
# different basis species, but same elements
basis(c("MgSO4", "HCHO", "CH4", "NO", "CO", "H3PO4", "CS2", "OH-"))
species() # formation reactions were updated

```

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

```

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

```

Arguments

logK list, equilibrium constants of formation reactions of species, or NULL, indicates to load the species present in the buffer.

ibasis numeric, which of the basis species whose activities are being calculated.

logact.basis list, logarithms of activities of the basis species.

<code>is.buffer</code>	numeric, rownumbers of the buffering species in <code>thermo\$species</code> .
<code>balance</code>	character, balance on this (name/formula of basis species or 'PBB') in buffer reactions.
<code>name</code>	character, name of buffer to add to or find in <code>thermo\$buffers</code> .
<code>species</code>	character, names or formulas of species in a buffer.
<code>state</code>	character, physical states of species in buffer.
<code>logact</code>	numeric, logarithms of activities of species in buffer.

Details

A buffer is treated here as assemblage 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 ($\text{FeS}_2\text{-FeS-Fe}_3\text{O}_4$) 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_2S 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, including `ionized` ones, 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

Garrels, R. M. (1960) *Mineral Equilibria*. Harper & Brothers, New York, 254 p. <http://www.worldcat.org/oclc/552690>

Schulte, M. D. and Shock, E. L. (1995) Thermodynamics of Strecker synthesis in hydrothermal systems. *Orig. Life Evol. Biosph.* **25**, 161–173. <http://dx.doi.org/10.1007/BF01581580>

See Also

The function `change` is a wrapper around `mod.buffer`. See `ionize` for examples using protein buffers.

Examples

```
## 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
affinity(return.buffer=TRUE)
# 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",logact=-10)
affinity(O2=c(-85,-70,4),T=c(25,100,4),return.buffer=TRUE)
# see longex('co2ac') for a different strategy using the
# 'what' argument of diagram

## buffer made of three species
## Pyrite-Pyrrhotite-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
```

```

# attach basis species H2S and O2 to the PPM buffer
basis(c("FeS2", "H2S", "O2", "H2O"), c(0, "PPM", "PPM", 0))
# inspect values of H2S activity and O2 fugacity
affinity(T=c(200,400,11), P=2000, return.buffer=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)$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("CHNOSe")
# 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(0, -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[[1]]), "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)

## log fH2 - temperature diagram for mineral buffers
## and for given activities of aqueous CH2O and HCN
## After Schulte and Shock, 1995, Fig. 6:

```

```

## 300 bars, log fCO2=1, log fN2=0, log aH2O=0
# the mineral buffers FeFeO, QFM, PPM and HM are already
# included in the thermo$buffers table so let's plot them.
basis.logacts <- c(999,1,0,0,999,999,999)
basis(c("Fe", "CO2", "H2O", "nitrogen", "hydrogen",
       "H2S", "SiO2"),basis.logacts)
basis(c("CO2", "N2"), "gas")
# initialize the plot
xlim <- c(0,350)
thermo.plot.new(xlim=xlim,ylim=c(-4,4),
               xlab=axis.label("T"),ylab=axis.label("H2"))
res <- 50
Tseq <- seq(xlim[1],xlim[2],length.out=res)
# a function to plot the log fH2 of buffers and label the lines
logfH2plot <- function(buffer,lty,where) {
  basis("H2",buffer)
  a <- as.numeric(affinity(T=c(xlim, res),P=300,return.buffer=TRUE)$H2)
  lines(Tseq,a,lty=lty)
  # "where" is the percent distance along the x-axis to plot the label
  wherethis <- seq(xlim[1],xlim[2],length.out=100)[where]
  if(length(grep("_",buffer)) > 0) tt <-
    thermo$buffers$logact[thermo$buffers$name==buffer] else tt <- buffer
  text(wherethis,splinefun(Tseq,a)(wherethis),tt)
}
# plot log fH2 of each mineral buffer
logfH2plot("FeFeO",1,16)
logfH2plot("QFM",1,30)
logfH2plot("PPM",1,80)
logfH2plot("HM",1,40)
anotherplotfunction <- function(mybuff,lty,logact,where) {
  for(i in 1:length(logact)) {
    # update the species activity
    mod.buffer(mybuff,logact=logact[i])
    logfH2plot(mybuff,lty,where[i])
  }
}
# add and plot new buffers (formaldehyde and HCN)
mod.buffer("mybuffer_1","formaldehyde","aq")
logact <- c(-6,-10,-15); where <- c(10,10,25)
anotherplotfunction("mybuffer_1",3,logact,where)
mod.buffer("mybuffer_2","HCN","aq")
where <- c(20,73,50)
anotherplotfunction("mybuffer_2",2,logact,where)
# title
title(main=paste("Mineral buffers (solid), HCN (dashed), formaldehyde",
               "(dotted)\n",describe(thermo$basis[c(2,4),],T=NULL,P=300),
       "After Schulte and Shock, 1995"),cex.main=0.9)

```

Description

Plot chemical activity (speciation) or equal-activity (predominance) diagrams as a function of chemical activities of basis species, temperature and/or pressure. Or, plot values of chemical affinity, logK, logQ, Gibbs energies of basis species, species, and formation reactions, as a function of zero or one variables.

Usage

```
diagram(affinity, what = "logact", ispecies = NULL, balance = NULL,
        names = NA, color = NA, add = FALSE, dotted = 0, cex = par("cex"),
        col = par("col"), pe = TRUE, pH = TRUE, ylog = TRUE,
        main = NULL, cex.names = 1, legend.x = "topright",
        lty = NULL, col.names = par("fg"), cex.axis=par("cex"),
        logact = NA, lwd = par("lwd"), alpha = FALSE,
        mar = NULL, residue = FALSE, yline = par("mfp")[1] + 1,
        xrange = NULL, ylab = NULL, xlab = NULL, do.plot = TRUE,
        as.residue = FALSE, mam = TRUE, group = NULL, bg = par("bg"),
        side = 1:4, xlim = NULL, ylim = NULL)
strip(affinity, ispecies = NULL, col = NULL, ns = NULL,
      xticks = NULL, ymin = -0.2, xpad = 1, cex.names = 0.7)
```

Arguments

affinity	list, object returned by affinity .
ispecies	numeric, which species to consider (default of NULL is to consider all species).
balance	character, name of the balanced quantity in formation reactions, or numeric, coefficients to balance formation reactions.
names	character, names of species for activity lines or predominance fields.
color	character, colors of predominance fields, or colors of activity lines.
add	logical, add to current plot?
dotted	numeric, how often to skip plotting points on predominance field boundaries (to gain the effect of dotted or dashed boundary lines).
cex	numeric, character expansion (scaling relative to current).
col	character, color of predominance field boundaries and labels (diagram), or colors of bars in a strip diagram (strip).
pe	logical, convert an 'e-' axis to a 'pe' one? Default is TRUE; set this to FALSE to prevent this conversion.
pH	logical, convert an 'H+' axis to a 'pH' one?
ylog	logical, use a logarithmic y-axis (on 1D degree diagrams).
main	character, a main title for the plot. NULL means to plot no title.
cex.names	numeric, character expansion factor to be used for names of species on plots.
legend.x	character, description of legend placement passed to legend .
lty	numeric, line types to be used in plots.
col.names	character, colors for labels of species.

<code>cex.axis</code>	numeric, character expansion factor for names of axes.
<code>logact</code>	numeric, logarithm of total activity of balanced quantity (for speciation diagrams). If NA, the total activity of the balanced quantity is determined by the activities of the species.
<code>what</code>	character, what property to calculate and plot.
<code>lwd</code>	numeric, line width.
<code>alpha</code>	logical, for speciation diagrams, plot degree of formation instead of activities?
<code>mar</code>	numeric, margins of plot frame.
<code>residue</code>	logical, rewrite reactions to refer to formation of one mole of the balanced quantity (i.e., protein backbone group in proteins)?
<code>yline</code>	numeric, margin line on which to plot the y-axis name.
<code>xrange</code>	numeric, range of x-values between which predominance field boundaries are plotted.
<code>ylab</code>	character, label to use for y-axis.
<code>xlab</code>	character, label to use for x-axis.
<code>do.plot</code>	logical, make a plot?
<code>as.residue</code>	logical, make plot using activities of residues?
<code>mam</code>	logical, should maximum affinity method be used for 2-D diagrams?
<code>group</code>	list of numeric, groups of species to consider as a single effective species.
<code>bg</code>	character, background color for legend.
<code>side</code>	numeric, which sides of plot to draw axes.
<code>xlim</code>	numeric, limits of x-axis.
<code>ylim</code>	numeric, limits of y-axis.
<code>ns</code>	numeric, numbers of species, used to make inset plots for strip diagrams.
<code>xticks</code>	numeric, location of supplemental tick marks on x-axis.
<code>ymin</code>	numeric, lower limit of y-axis.
<code>xpad</code>	numeric, amount to extend x-axis on each side.

Details

The `diagram` function takes as its primary input the results from `affinity` and displays diagrams representing either the thermodynamic properties of species, or the chemical activities of the species. The chemical activities are obtained by switching from an equal-activity reference state (values calculated using `affinity`) to an equal-affinity reference state (calculated using either a reaction matrix or the Maxwell-Boltzmann distribution). The activity diagrams that can be produced include chemical speciation diagrams as a function of one of temperature, pressure, or the chemical activities of the basis species, or equal-activity (predominance) diagrams as a function of two of these variables.

To generate the stability relations from affinities of formation reactions, a reaction conservation rule is either automatically determined or specified by the user. For example, $3\text{Fe}_2\text{O}_3 = 2\text{Fe}_3\text{O}_4 + 1/2\text{O}_2$ is balanced on Fe (or a basis species containing Fe) and $4\text{Fe}_2\text{O}_3 + \text{Fe} = 3\text{Fe}_3\text{O}_4$ is balanced on O (or a basis species containing O). The default action, if `balance` is NULL, is to balance

on a basis species that appears in the formation reactions of all of the species of interest. The first such basis species (in order of their appearance in `thermo$basis`) will be used; this basis species is determined using `which.balance`. If a common basis species is not available, or if `balance` is 1, the balance is set to unity. For proteins, an additional conservation rule is available; if `balance` is `PBB`, or if it is missing and all the species appear to be proteins (their names contain underscores) the metastability calculations will be balanced on protein length (number of protein backbone groups).

Predominance (2-D) diagrams are usually produced using the maximum affinity method, which is based on the notion that the predominant species at any point in the diagram is that one that has the greatest affinity of formation divided by the balanced quantity. This behavior can be altered by specifying `mam` as `FALSE`, in which case the relative abundances of all species are calculated in the manner described below and the most abundant one at each grid point identified as the predominant species. However, this procedure can be very slow unless the reactions are cast in terms of residues (i.e., to use `equil.boltz` instead of `equil.react`).

When the coefficients of the balanced quantity are all equal, the location of the predominance field boundaries does not depend on the actual value of the activities of the species of interest, as long as they are all equal (these are "equal-activity" lines). Generally in the case in reactions among proteins, the coefficients of the balanced quantity are unequal in the different species, so the location of the predominance field boundaries does depend on the actual value chosen to represent equal activities of the species of interest. In this case the predominance field boundaries are "activity equal to X" lines. This is true for `mam` equal to `TRUE`; if `mam` is `FALSE`, the predominance fields really denote the most abundant species in a system, and the boundaries can represent different "equal-activity" values across the diagram.

`residue`, if `TRUE`, instructs the function to rewrite the formation reactions so that each refers to formation of one mole of the balanced quantity (e.g., a residue of a protein for reactions conserving the protein backbone); the balance coefficients are then all unity. This is the recommended option for calculating relative abundances of proteins, because the large numbers that would otherwise be used as balance coefficients often create a situation of utter predominance of a single protein (see Dick, 2008). If `as.residue` is also `TRUE`, the calculated logarithms of activities of the residues are used in the plot and returned by the function, otherwise those of the species are shown. These options, however, are not restricted to systems of proteins.

If `group` 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 2-D diagrams, `group` only makes sense if `mam` is `FALSE`. An example of using the `group` argument is supplied in `get.protein`.

For 1-D diagrams, the logarithmicity and limits of the y-axis can be controlled; the default (`ylog` is `TRUE`) is to plot logarithms of activities of species. If `alpha` is `TRUE`, the degrees of formation (ratios of activities of species to total activity) are plotted instead. The range of the y-axis on these diagrams can be controlled with `yylim` (which defaults to `c(0, 1)` when `alpha` is `TRUE`). Also, a `legend` is placed at the location identified by `legend.x`, or omitted if `legend.x` is `FALSE`.

`what` indicates the type of calculation to perform. Usually, this means the metastable equilibrium logarithms of activities of the species of interest. Alternatively, `what` can be one of the (formulas of the) basis species, meaning to calculate the equilibrium logarithm of activity of that basis species in each of the formation reactions. Another option is to set `what` to 'A', meaning to plot the affinities themselves (don't perform any equilibrium calculation). All of these actions assume that the property supplied in `a` is 'A' (i.e., affinities of formation reactions). If it is anything else, `what` is ignored and the values themselves (as listed in `a`) are plotted.

The `diagram` function by default starts a new plot, but if `add` is `TRUE` it adds to the current plot. If `do.plot` is `FALSE`, no plot will be generated but all the required computations will be performed and the results returned. The names of the species, and the colors used to identify them (on chemical activity lines or predominance fields) can be changed; by default `heat.colors` are used to shade the predominance fields in on-screen diagrams. The `col` option controls the color of the predominance field boundaries, and `col.names` the color of the labels for the predominance fields. The line type (only on 1-D diagrams) and line width can be controlled with (`lty` and `lwd`, respectively). The line style 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. Finally, `cex`, `cex.names`, and `cex.axis` adjust the overall character expansion factors (see `par`) and those of the species names and axis labels.

The x-axis label (1-D and 2-D diagrams) and y-axis label (2-D diagram) are automatically generated unless they are supplied in `xlab` and `ylab`. 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.

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.

References

- Bowers, T. S., Jackson, K. J. and Helgeson, H. C. (1984) *Equilibrium Activity Diagrams for Coexisting Minerals and Aqueous Solutions at Pressures and Temperatures to 5 kb and 600 °C*. Springer-Verlag, Heidelberg, 397 p. <http://www.worldcat.org/oclc/224591948>
- Dick, J. M. (2008) Calculation of the relative metastabilities of proteins using the CHNOSZ software package. *Geochem. Trans.* **9**:10. <http://dx.doi.org/10.1186/1467-4866-9-10>
- Ernst, W. G. (1976) *Petrologic Phase Equilibria*. W. H. Freeman, San Francisco, 333 p. <http://www.worldcat.org/oclc/2072721>
- Helgeson, H. C. (1970) A chemical and thermodynamic model of ore deposition in hydrothermal systems. *Mineral. Soc. Amer. Spec. Pap.* **3**, 155–186. <http://www.worldcat.org/oclc/583263>

- Helgeson, H. C., Delany, J. M., Nesbitt, H. W. and Bird, D. K. (1978) Summary and critique of the thermodynamic properties of rock-forming minerals. *Am. J. Sci.* **278-A**, 1–229. <http://www.worldcat.org/oclc/13594862>
- LaRowe, D. E. and Helgeson, H. C. (2007) Quantifying the energetics of metabolic reactions in diverse biogeochemical systems: electron flow and ATP synthesis. *Geobiology* **5**, 153–168. <http://dx.doi.org/10.1111/j.1472-4669.2007.00099.x>
- Majzlan, J., Navrotsky, A., McClesky, R. B. and Alpers, C. N. (2006) Thermodynamic properties and crystal structure refinement of ferricopiapite, coquimbite, rhomboclase, and $\text{Fe}_2(\text{SO}_4)_3(\text{H}_2\text{O})_5$. *Eur. J. Mineral.* **18**, 175–186. <http://dx.doi.org/10.1127/0935-1221/2006/0018-0175>
- Pourbaix, M. J. N. (1949) *Thermodynamics of dilute aqueous solutions*. Edward Arnold & Co., London, 136 p. <http://www.worldcat.org/oclc/1356445>
- Seewald, J. S. (1997) Mineral redox buffers and the stability of organic compounds under hydrothermal conditions. *Mat. Res. Soc. Symp. Proc.* **432**, 317–331. <http://lucy.mrs.org/meetings/spring96/Program/S.S96.html>
- Tagirov, B. and Schott, J. (2001) Aluminum speciation in crustal fluids revisited. *Geochim. Cosmochim. Acta* **65**, 3965–3992. [http://dx.doi.org/10.1016/S0016-7037\(01\)00705-0](http://dx.doi.org/10.1016/S0016-7037(01)00705-0)
- Tardy, Y., Schaul, R. and Duplay, J. (1997) Thermodynamic stability fields of humus, microflora and plants. *C. R. Acad. Sci. Paris* **324**, 969–976. [http://dx.doi.org/10.1016/S1251-8050\(97\)83981-X](http://dx.doi.org/10.1016/S1251-8050(97)83981-X)

See Also

[affinity](#) for the source of the input for this function; [par](#) for how graphical parameters are set, [protein](#) and [buffer](#) for examples other than those shown below.

Examples

```
## chemical affinities of formation (for equal activities)
## and equilibrium activities (for equal affinities) of amino acids
opar <- par(mfrow=c(2,2))
mycol <- rev(rainbow(5))
basis("CHNOS+")
# comment out next line to find predominance of methionine
basis("H2S",-25)
aa <- c("glutamic acid","methionine","isoleucine",
        "leucine","tyrosine")
species(aa)
# affinities of reactions per CO2 at fixed conditions
a <- affinity()
diagram(a,what="A",col=mycol,yline=3,
        main=paste("affinity of formation reactions per CO2\n",
                  "logfO2 = -80, logaH2O = 0"))
# affinities of reactions as a function of oxygen fugacity
a <- affinity(O2=c(-80,-66))
diagram(a,what="A",ylim=c(-15,5),col=mycol,lwd=2)
title("affinity per CO2; logaH2O = 0")
## logarithms of activities of amino acids
# reactions balanced on CO2
```

```

diagram(a, legend.x="bottomright", col=mycol, lwd=2)
title("equilibrium activities; logaH2O = 0")
# in two dimensions
a <- affinity(O2=c(-80,-66), H2O=c(-8,4))
diagram(a, color=mycol)
title("highest equilibrium activities")
# reset the plot device
par(opar)

### calculate the equilibrium logarithm of activity of a basis species
basis("CHNOS")
species(c("ethanol", "acetic acid", "lactic acid"))
a <- affinity(T=c(0,150))
diagram(a, what="O2")
# can also be done in 2 dimensions
a <- affinity(T=c(0,150), CO2=c(-10,0))
diagram(a, what="O2", ispecies=1, main=NULL)
diagram(a, what="O2", ispecies=2, col="red", add=TRUE)
diagram(a, what="O2", ispecies=3, col="blue", add=TRUE)
title(main="logfO2 ethanol/acetic/lactic (black/red/blue)")

### 1-D logarithm of activity plots
### (speciation diagrams)

## Aqueous sulfur species (after Seewald, 1997)
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-"))
diagram(affinity(O2=c(-50,-15), T=325, P=350),
  ylim=c(-30,0), logact=-2, legend.x="topleft")
title(main=paste("Aqueous sulfur speciation, 325 degC, 350 bar\n",
  "After Seewald, 1997"))
# try it with and without the logact 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))
diagram(a, 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
# Mg+2 here is the immobile component
# cf. LH07, where bulk composition of Mg+2 is specified
basis(c("CO2", "NH3", "H2O", "H3PO4", "O2", "H+", "Mg+2"),
  c(999,999,999,999,999,-5,-4))
species(c("HATP-3", "H2ATP-2", "MgATP-2", "MgHATP-"))
diagram(affinity(T=c(0,120,25)), alpha=TRUE)

```

```

title(main=paste("Degrees of formation of ATP species,\n",
  "pH=5, log(aMg+2)=-3. After LaRowe and Helgeson, 2007"),
  cex.main=0.9)

## using strip(): equilibrium abundances of
## proteins from different mammals
organisms <- c("BOVIN","CANFA","HUMAN","MOUSE","PIG")
proteins <- c("AQP1","MYG","P53")
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,ymin=-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"))

### predominance diagrams (equal activities of
### species as a function of two variables)

## T-P stability diagram for SiO2, after Ernst, 1976, Fig. 4.4
# the system is SiO2 (one component) but the basis species require
# all the elements: note that basis(c("SiO2","O2")) would identify
# SiO2(aq) which is okay but brings in calculations of properties
# of water which are relatively slow.
basis(c("quartz","O2")) # cr, gas
# browse the species of interest
info("SiO2 ")
# we'll take the crystalline ones
si <- info("SiO2","cr")
species(si)
# calculate chemical affinities
# the do.phases argument is necessary for
# dealing with the phase transitions of minerals
a <- affinity(T=c(0,2000),P=c(0,30000),do.phases=TRUE)
diagram(a)
title(main="Phases of SiO2\nafter Ernst, 1976")

## Distribution of copper on Eh-pH diagram
## after Fig. 15 of Pourbaix, 1949
basis(c("Cu+2","Cl-","H2O","H+","e-"))
basis("Cl-",-2)
# two aqueous species, three solid ones
# (our CuCl is aq but it was cr in P's Fig. 15)
species(c("CuCl","Cu+2","copper","cuprite","tenorite"))
# (which is equivalent to ...)
# species(c("CuCl","Cu+2","Cu","Cu2O","CuO"),c("","","","","","cr"))
a <- affinity(pH=c(0,7),Eh=c(-0.1,1))

```

```

# this one corresponds to activity contours of
# aqueous species at 10^-3 (the default aq activity in CHNOSZ)
diagram(a,color=NULL)
# here we set activities to unity; aq-cr boundaries change
species(c("CuCl","Cu+2"),c(0,0))
a <- affinity(pH=c(0,7),Eh=c(-0.1,1))
diagram(a,add=TRUE,names=NULL,col="blue",color=NULL)
water.lines()
title(main=paste("H2O-Cl-(Cu); activities of 10^-3 (black)\n",
  "or 0 (blue); after Pourbaix, 1949"))

## a 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("rhombochase","ferricopiapite","hydronium jarosite",
  "goethite","melanterite","pyrite"))
a <- affinity(pH=c(-1,4),pe=c(-5,23))
diagram(a)
water.lines(yaxis="pe")
title(main=paste("Fe-S-O-H After Majzlan et al., 2006",
  describe(thermo$basis[2:3,],digits=2),sep="\n"))
# reset the database
data(thermo)

## cysteine-cysteinate-cystine Eh-pH at 25 and 100 deg C
basis("CHNOSe")
species(c("cysteine","cysteinate","cystine"))
a <- affinity(pH=c(5,10),Eh=c(-0.5,0))
diagram(a,color=NULL)
water.lines()
a <- affinity(pH=c(5,10),Eh=c(-0.5,0),T=100)
diagram(a,col="red",add=TRUE,names=NULL)
water.lines(T=convert(100,"K"),col="red")
title(main=paste("Cysteine Cysteinate Cystine",
  "25 and 100 deg C",sep="\n"))

## Soil Organic Matter CO2-H2O, O2-H2O (after Tardy et al., 1997)
# NH3 is conserved, and H2O is on an axis of the diagram
# formulas for aqueous species, names for phases ...
add.obigt()
basis(c("NH3","water","CO2","O2"),c(999,999,-2.5,-28))
# switch to gaseous CO2 (aq is the default)
basis("CO2","gas")
# load the species of interest
species(c("microflore","plantes","acide fulvique",
  "acide humique","humine"))
# proceed with the diagrams
diagram(affinity(H2O=c(-0.6,0.1),CO2=c(-3,-1)))
title(main=paste("Relative stabilities of soil organic matter\n",
  "after Tardy et al., 1997"))
# this is the O2-H2O diagram

```

```

# diagram(affinity(H2O=c(-1,0.5),O2=c(-28.5,-27.5)))
data(thermo)

## Aqueous Aluminum Species F-/OH- (after Tagirov and Schott, 2001)
# in order to reproduce this calculation, we have to
# consider the properties of species given by these authors,
# which are not the default ones in the database
add.obigt()
# The coefficients on H+ and O2 in all the formation reactions
# are zero, so the number of basis species here is three. Al+3
# becomes the conservant, and F- and OH- are being plotted ...
# so their initial activities don't have to be set.
basis(c("Al+3", "F-", "OH-", "H+", "O2"), rep(999, 5))
species(c("Al+3", "Al(OH)4-", "AlOH+2", "Al(OH)2+", "Al(OH)3",
  "AlF+2", "AlF2+", "AlF3", "AlF4-", "Al(OH)2F2-", "Al(OH)2F", "AlOHF2"))
# Increase the x- and y- resolution from the default and calculate
# and plot predominance limits. Names of charged basis species,
# such as "H+", "e-" and the ones shown here, should be quoted
# when given as arguments to affinity(). The OH- values shown here
# correspond to pH=c(0,14) (at unit activity of water).
a <- affinity("OH-"=c(-14,0), "F-"=c(-1,-8), T=200)
diagram(a)
title(main=paste("Aqueous aluminium species, T=200 C, P=Psat\n",
  "after Tagirov and Schott, 2001"))
# We could do this to overlay boundaries for a different pressure
#a.P <- affinity("OH-"=c(-14,0), "F-"=c(-1,-8), T=200, P=5000)
#diagram(a.P, names=NULL, color=NULL, col="blue", add=TRUE)
# restore thermodynamic database to default
data(thermo)

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

## Temperature-Pressure: kyanite-sillimanite-andalusite
# this is a system of a single component (Al2SiO5)
# but we still have to use the same number of
# basis species as the number of elements
basis(c("kyanite", "quartz", "oxygen"))
species(c("kyanite", "sillimanite", "andalusite"))
diagram(affinity(T=c(0,1000,200), P=c(500,5000,200)), color=NULL)
title(main="Al2SiO5")
# end donttest

```

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

```
cgl(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)
```

Arguments

<code>property</code>	character, name(s) of properties to calculate.
<code>T</code>	numeric, temperature(s) at which to calculate properties (K).
<code>P</code>	numeric, pressure(s) at which to calculate properties (bar).
<code>ghs</code>	dataframe, values of the standard molal Gibbs energy and enthalpy of formation from the elements and entropy at 25 °C and 1 bar.
<code>eos</code>	dataframe, values of the equations-of-state parameters.
<code>contrib</code>	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 <code>c("n", "s", "o")</code> , for all contributions.
<code>H2O.PT</code>	dataframe, values of the electrostatic properties of water at the temperature(s) and pressure(s) of interest.
<code>H2O.PrTr</code>	dataframe, values of the electrostatic properties of water at the reference temperature and pressure.
<code>domega</code>	logical, calculate the T and P derivatives of omega?

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 either of 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 `a1`, `a2`, `a3`, `a4`, `c1`, `c2`, `omega` and `Z`; the units of these parameters are as indicated for `thermo$obigt`, sans the order of magnitude multipliers. Note that the equation-of-state parameter `Z` (appearing in the *g*-function for the temperature derivatives of the omega parameter; Shock et al., 1992) is taken from `thermo$obigt` and *not* from the `makeup` of the species, although in most cases the two values are coincident. `H2O.PT` and `H2O.PrTr` are optional arguments that contain the

electrostatic properties of H₂O required for the calculations. If either of these is NULL (the default), the required values are retrieved using `water`. Unless `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 `omega` parameter for charged species (where $Z \neq 0$) are calculated using the *g*- and *f*-functions (Shock et al., 1992; Johnson et al., 1992). 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.

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 untenable. These temperature limits are stored in the thermodynamic database, but `cgl` ignores them (`subcrt` warns if they are exceeded).

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.

References

- Helgeson, H. C., Kirkham, D. H. and Flowers, G. C. (1981) Theoretical prediction of the thermodynamic behavior of aqueous electrolytes at high pressures and temperatures. IV. Calculation of activity coefficients, osmotic coefficients, and apparent molal and standard and relative partial molal properties to 600°C and 5 Kb. *Am. J. Sci.* **281**, 1249–1516. <http://www.ajsonline.org/cgi/content/abstract/281/10/1249>
- Helgeson, H. C., Owens, C. E., Knox, A. M. and Richard, L. (1998) Calculation of the standard molal thermodynamic properties of crystalline, liquid, and gas organic molecules at high temperatures and pressures. *Geochim. Cosmochim. Acta* **62**, 985–1081. [http://dx.doi.org/10.1016/S0016-7037\(97\)00219-6](http://dx.doi.org/10.1016/S0016-7037(97)00219-6)
- Maier, C. G. and Kelley, K. K. (1932) An equation for the representation of high-temperature heat content data. *J. Am. Chem. Soc.* **54**, 3243–3246. <http://dx.doi.org/10.1021/ja01347a029>
- Shock, E. L. and Helgeson, H. C. (1988) Calculation of the thermodynamic and transport properties of aqueous species at high pressures and temperatures: Correlation algorithms for ionic species and equation of state predictions to 5 kb and 1000°C. *Geochim. Cosmochim. Acta* **52**, 2009–2036. [http://dx.doi.org/10.1016/0016-7037\(88\)90181-0](http://dx.doi.org/10.1016/0016-7037(88)90181-0)

Shock, E. L., Oelkers, E. H., Johnson, J. W., Sverjensky, D. A. and Helgeson, H. C. (1992) Calculation of the thermodynamic properties of aqueous species at high pressures and temperatures: Effective electrostatic radii, dissociation constants and standard partial molal properties to 1000 °C and 5 kbar. *J. Chem. Soc. Faraday Trans.* **88**, 803–826. <http://dx.doi.org/10.1039/FT9928800803>

Tanger, J. C. IV and Helgeson, H. C. (1988) Calculation of the thermodynamic and transport properties of aqueous species at high pressures and temperatures: Revised equations of state for the standard partial molal properties of ions and electrolytes. *Am. J. Sci.* **288**, 19–98. <http://www.ajsonline.org/cgi/content/abstract/288/1/19>

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

```
## 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=c(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
a <- info(info(rep("n-octane", 3), c("cr", "liq", "gas")))
b <- cgl(property="G", ghs=a, eos=a, T=seq(200, 420, 10), P=1)
which.pmax(b, pmin=TRUE) # 1 = cr, 2 = liq, 3 = gas
# compare that result with the tabulated transition temperatures
print(a)
```

Description

Functions for fitting experimental volume and heat capacities using regression equations. Possible models include the Helgeson-Kirkham-Flowers (HKF) equations of state and other equations defined using any combination of terms derived from the temperature, pressure and thermodynamic and electrostatic properties of water.

Usage

```

EOSregress(exptdata, var = "", T.max = 9999)
EOSvar(var, T, P)
EOScalc(coefficients, T, P)
EOSplot(exptdata, var = NULL, T.max = 9999, T.plot = NULL,
        P = 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`; these names correspond to variables identified below:

T	T (temperature)
P	P (pressure)
TTheta	$(T - \Theta)$ ($\Theta = 228$ K)
invTTheta	$1/(T - \Theta)$
TTheta2	$(T - \Theta)^2$
invTTheta2	$1/(T - \Theta)^2$
V	V (volume of water)
E	E (isobaric expansivity of water)
kT	κ_T (isothermal compressibility of water)
alpha	α (coefficient of isobaric expansivity of water)
beta	β (coefficients of isothermal compressibility of water)
X	X (Born function X)
Q	Q (Born function Q)
TX	TX (temperature times X)
drho.dT	$d\rho/dT$ (temperature derivative of density of water)
V.kT	$V\kappa_T$ (volume times isothermal compressibility of water)

EOSvar takes as input `var` (one of the names of the variables listed above), and `T` (temperature in degrees Kelvin), `P` (pressure in bars). It returns the value of the variable at the specified temperature-pressure condition(s). 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 `EOSpred` 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 `as.expressions` in plots. The value of `coeff` is prefixed (using `substitute`) to the name of the variable.

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 dataframe has column names of '(Intercept)', 'invTTheta2' and 'TX', respectively holding the coefficients c_1 , c_2 and ω in equation $Cp^\circ = c_1 + c_2/(T - \Theta)^2 + \omega TX$. If the property is 'V', the data frame has column names of '(Intercept)', 'invTTheta' and 'Q', respectively holding the coefficients σ , ξ and $-\omega$ in $V^\circ = \sigma + \xi/(T - \Theta) - \omega Q$.

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 degrees C).

Value

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

References

Schulte, M. D., Shock, E. L. and Wood, R. H. (1995) The temperature dependence of the standard-state thermodynamic properties of aqueous nonelectrolytes. *Geochim. Cosmochim. Acta* **65**, 3919–3930. [http://dx.doi.org/10.1016/S0016-7037\(01\)00717-7](http://dx.doi.org/10.1016/S0016-7037(01)00717-7)

See Also

See `lm` for the details of the regression calculations.

Examples

```
## regress 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="CHNOSZ")
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", "TX")
# 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")

## model experimental volumes of CH4
## using HKF equation and an exploratory one
f <- system.file("extdata/cpetc/V.CH4.HWM96.csv", package="CHNOSZ")
d <- read.csv(f)
d$P <- convert(d$P, "bar")
# the HKF equation
varHKF <- c("invTTheta", "Q")
# 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

```

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

<code>file</code>	character, path to EQ6 output file
<code>species</code>	character, name(s) of species or minerals
<code>property</code>	character, property to get
<code>outfile</code>	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 degrees 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

Wolery, T. J. (1992) EQ3/6, A Software Package for Geochemical Modeling of Aqueous Systems: Package Overview and Installation Guide (Version 7.0). Lawrence Livermore National Laboratory, UCRL-MA-110662 PT I. http://www.wipp.energy.gov/library/cra/2009_cra/

[references/Others/Wolery_1992_EQ36_A_Software_Package_for_Geochemical_Modeling_of_Aqueous_Systems_ERMS241375.pdf](http://www.wipp.energy.gov/library/cra/2009_cra/references/Others/Wolery_Daveler_1992_EQ36_A_Computer_Program_for_Reaction_Path_Modeling_of_Aqueous_Geochemical_Systems_ERMS241375.pdf)

Wolery, T. J. and Daveler, S. A. (1992) EQ6, A Computer Program for Reaction Path Modeling of Aqueous Geochemical Systems: Theoretical Manual, User's Guide, and Related Documentation (Version 7.0). Lawrence Livermore National Laboratory, UCRL-MA-110662 PT IV. http://www.wipp.energy.gov/library/cra/2009_cra/references/Others/Wolery_Daveler_1992_EQ36_A_Computer_Program_for_Reaction_Path_Modeling_of_Aqueous_Geochemical_Systems_ERMS241379.pdf

Examples

```
## 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+", "sio2,aq", "h2,aq"), "log act")
## End(Not run)
```

equil

Calculate Logarithms of Activity of Species

Description

Calculate equilibrium chemical activities of species taking as input the affinities of formation of the species at unit activity.

Usage

```
equil.boltz(Astar, nbalance, thisloga)
equil.react(Astar, av, nbalance, thisloga)
```

Arguments

Astar	numeric, affinities of formation reactions excluding species contribution.
nbalance	numeric, vector of balance coefficients.
thisloga	numeric, logarithm of total activity of balanced thing.
av	numeric, values of affinities of formation reactions.

Details

The chemical activities of species shown by [diagram](#) are calculated using either the `equil.boltz` or `equil.react` functions. The former is used if `residue` in `diagram` is set to `TRUE`, and the latter if `residue` is set to `FALSE`.

The input values are in a list, `Astar`, all elements of the list having the same dimensions. Each element contains the chemical affinities of the formation reactions of one of the species of interest at

unit activity. The metastable equilibrium activities calculated using either `equil.*` function satisfy the constraints that 1) the resulting chemical affinities of the formation reactions of the species are all equal and 2) the total activity of the conserved component (`thisloga`) is unchanged.

In `equil.react` (the algorithm described in Dick, 2008 and the only one available prior to CHNOSZ-0.8), the calculations of relative abundances of species use the activities in the `affinity` output as initial guesses, and attempt to solve a system of equations that represent the two constraints stated above. So, if you supply a value for `thisloga` that is much different from that of the initial guess you may end up with errors from `uniroot` such as "f() values at end points not of opposite sign".

In `equil.boltz` (algorithm available beginning with CHNOSZ-0.8), the chemical activities of species are calculated using the Boltzmann distribution. This calculation is faster than the equation-solving approach used above, but is limited to systems where the balance coefficients are all unity. Therefore, this function is only called by `diagram` for systems of proteins when `residue` set to `TRUE`.

Value

A numeric value (possibly an array) with dimensions equal to those of one of the list elements of `Astar`.

examples

Run examples from the documentation

Description

Run the examples contained in each of the documentation topics.

Usage

```
examples(do.png = FALSE)
longex(which)
```

Arguments

<code>do.png</code>	logical, generate PNG files for the plots?
<code>which</code>	character, which example to run.

Details

`examples` runs all the examples in the documentation for the package. The `example` function 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.

`longex` is a function that contains code to run various other examples that are not contained in the documentation. The example to run is specified by `which`. The available examples are listed below. See the comments in the source code for more information about each one.

sources	cross-check the reference list with the thermodynamic database
copper	an Eh-pH diagram for the copper-water-glycine system
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
pie	pie charts comparing relative abundances of organisms (Spear et al., 2005) and model proteins in metastable
orp	oxidation-reduction potential of redox standards as a function of temperature
findit	detailed example of usage of <code>findit</code> using log-normal distribution as an objective
co2ac	activity of CO ₂ buffered by acetic acid; comparing <code>affinity</code> (<code>return.buffer=TRUE</code>) with <code>diagra</code>

References

Spear, J. R., Walker, J. J., McCollom, T. M. and Pace, N. R. (2005) Hydrogen and bioenergetics in the Yellowstone geothermal ecosystem. *Proc. Natl. Acad. Sci. U. S. A.* **102**, 2555–2560. <http://dx.doi.org/10.1073/pnas.0409574102>

Examples

```
longex(c("copper", "orp"))
```

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:

- `AA03.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 the 'protactiv' vignette for an example that uses 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), copy-number (emPAI-derived copy number/cell), GRAVY (Kyte-Doolittle), FunCat (FunCat class description), PSORT (PSORT localisation), ribosomal (yes/no). See `get.expr` and the 'protactiv' vignette 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_refseq47.blast.xz`, `bisonS_vs_refseq47.blast.xz`, `bisonR_vs_refseq47.blast.xz`, `bisonQ_vs_refseq47.blast.xz`, `bisonP_vs_refseq47.blast.xz` are partial tabular BLAST results for proteins in the Bison Pool Environmental Genome. Predicted protein sequences 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 47, representing 3266 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 similarity greater than 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 was extracted from the complete 'gi_taxid_prot.dmp.gz' downloaded from <ftp://ftp.ncbi.nih.gov/pub/taxonomy/> on 2011-06-16. A small number (about 0.2 percent) of the gi numbers appearing in the BLAST results were not found in 'gi_taxid_prot.dmp.gz' and therefore are also excluded from `gi.taxid.txt`. See `id.blast` for an example that uses this file and the BLAST files described above.

Files in `cpet0` 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 $\text{J mol}^{-1} \text{K}^{-1}$ in the third. See `ionize` 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 $\text{J K}^{-1} \text{mol}^{-1}$ 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` for an example that uses this file.
- `SOJSH.csv` Experimental equilibrium constants for the reaction $\text{NaCl(aq)} = \text{Na}^+ + \text{Cl}^-$ 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.

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:

- `SGD.csv.xz` Dataframe of amino acid composition of proteins from the *Saccharomyces* Genome Database. Contains twenty-two columns. Values in the first column are the rownumbers, the second column (OLN) has the ordered locus names of proteins, and the remaining twenty columns (Ala.Val) contain the numbers of the respective amino acids in each protein; the columns are arranged in alphabetical order based on the three-letter abbreviations for the amino acids. The source of data for 'SGD.csv' is the file 'protein_properties.tab' found on the FTP site of the SGD project on 2008-08-04. Blank entries were replaced with "NA" and column headings were added. See `get.protein` for examples that use this file.
- `ECO.csv.xz` Contains 24 columns. Values in the first column correspond to rownumbers, the second column AC holds the accession numbers of the proteins, the third column (Name) has the names of the corresponding genes, and the fourth column OLN lists the ordered locus names of the proteins. The remaining twenty columns (A..Y) give the numbers of the respective amino acids in each protein and are ordered alphabetically by the one-letter abbreviations of the amino acids. The sources of data for 'ECO.csv' are the files 'ECOLI.dat' <ftp://>

`ftp.expasy.org/databases/hamap/complete_proteomes/entries/bacteria` and `'ECOLI.fas'` `ftp://ftp.expasy.org/databases/hamap/complete_proteomes/fasta/bacteria` downloaded from the HAMAP (High-quality Automated and Manual Annotation of microbial Proteomes system) FTP site (Gattiker et al., 2003) on 2007-12-20. The proteins can be included in calculations using `get.protein` as well as `get.expr`; see the 'protactiv' vignette for an example that uses the latter function.

Files in `refseq` contain code and results of processing NCBI Reference Sequences (RefSeq) for microbial proteins:

- `README.txt` Instructions for producing the data files.
- `gencat.sh` Bash script to microbial protein records from the RefSeq catalog.
- `mkfaa.sh` Combine the contents of `.faa.gz` files into a single FASTA file.
- `protein.refseq.R` Calculate average amino acid composition of all proteins for each organism identified by a taxonomic ID.
- `protein_refseq.csv.xz` Output from above. See example for [ZC](#).
- `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.
- `taxid_names.csv.xz` Output from above. See example for [id.blast](#).

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`, 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`.

References

- Anderson, N. L. and Anderson, N. G. (2002) The human plasma proteome: History, character and diagnostic prospects. *Molecular and Cellular Proteomics* **1**, 845–867. <http://dx.doi.org/10.1074/mcp.R200007-MCP200>
- Anderson, N. L. and Anderson, N. G. (2003) The human plasma proteome: History, character and diagnostic prospects (Vol. 1 (2002) 845-867). *Molecular and Cellular Proteomics* **2**, 50. <http://dx.doi.org/10.1074/mcp.A300001-MCP200>
- Benson, S. W. and Buss, J. H. (1958) Additivity rules for the estimation of molecular properties. Thermodynamic properties. *J. Chem. Phys.* **29**, 546–572. <http://dx.doi.org/10.1063/1.1744539>
- Domalski, E. S. and Hearing, E. D. (1993) Estimation of the thermodynamic properties of C-H-N-O-S-Halogen compounds at 298.15 K *J. Phys. Chem. Ref. Data* **22**, 805–1159. <http://dx.doi.org/10.1063/1.555927>
- Gattiker, A., Michoud, K., Rivoire, C., Auchincloss, A. H., Coudert, E., Lima, T., Kersey, P., Pagni, M., Sigrist, C. J. A., Lachaize, C., Veuthey, A.-L., Gasteiger, E. and Bairoch, A. (2003) Automatic annotation of microbial proteomes in Swiss-Prot. *Comput. Biol. Chem.* **27**, 49–58. [http://dx.doi.org/10.1016/S1476-9271\(02\)00094-4](http://dx.doi.org/10.1016/S1476-9271(02)00094-4)
- Gaucher, E. A., Thomson, J. M., Burgan, M. F. and Benner, S. A (2003) Inferring the palaeoenvironment of ancient bacteria on the basis of resurrected proteins. *Nature* **425**(6955), 285–288. <http://dx.doi.org/10.1038/nature01977>
- Ghaemmaghami, S., Huh, W., Bower, K., Howson, R. W., Belle, A., Dephoure, N., O’Shea, E. K. and Weissman, J. S. (2003) Global analysis of protein expression in yeast. *Nature* **425**(6959), 737–741. <http://dx.doi.org/10.1038/nature02046>
- Huh, W. K., Falvo, J. V., Gerke, L. C., Carroll, A. S., Howson, R. W., Weissman, J. S. and O’Shea, E. K. (2003) Global analysis of protein localization in budding yeast. *Nature* **425**(6959), 686–691. <http://dx.doi.org/10.1038/nature02026>
- Ishihama, Y., Schmidt, T., Rappsilber, J., Mann, M., Hartl, F. U., Kerner, M. J. and Frishman, D. (2008) Protein abundance profiling of the *Escherichia coli* cytosol. *BMC Genomics* **9**:102. <http://dx.doi.org/10.1186/1471-2164-9-102>
- HAMAP system. HAMAP FTP directory, <ftp://ftp.expasy.org/databases/hamap/>
- Hnedkovsky, L., Wood, R. H. and Majer, V. (1996) Volumes of aqueous solutions of CH₄, CO₂, H₂S, and NH₃ at temperatures from 298.15 K to 705 K and pressures to 35 MPa. *J. Chem. Thermodyn.* **28**, 125–142. <http://dx.doi.org/10.1006/jcht.1996.0011>
- Hnedkovsky, L. and Wood, R. H. (1997) Apparent molar heat capacities of aqueous solutions of CH₄, CO₂, H₂S, and NH₃ at temperatures from 304 K to 704 K at a pressure of 28 MPa. *J. Chem. Thermodyn.* **29**, 731–747. <http://dx.doi.org/10.1006/jcht.1997.0192>
- Joint Genome Institute (2007) Bison Pool Environmental Genome. Protein sequence files downloaded from IMG/M (<http://img.jgi.doe.gov/cgi-bin/m/main.cgi?section=FindGenomes&page=findGenomes>)
- Privalov, P. L. and Makhatadze, G. I. (1990) Heat capacity of proteins. II. Partial molar heat capacity of the unfolded polypeptide chain of proteins: Protein unfolding effects. *J. Mol. Biol.* **213**, 385–391. [http://dx.doi.org/10.1016/S0022-2836\(05\)80198-6](http://dx.doi.org/10.1016/S0022-2836(05)80198-6)

Robie, R. A. and Hemingway, B. S. (1995) *Thermodynamic Properties of Minerals and Related Substances at 298.15 K and 1 Bar (10⁵ Pascals) Pressure and at Higher Temperatures*. U. S. Geol. Surv., Bull. 2131, 461 p. <http://www.worldcat.org/oclc/32590140>

Roxby, R. and Tanford, C. (1971) Hydrogen ion titration curve of lysozyme in 6 M guanidine hydrochloride. *Biochemistry* **10**, 3348–3352. <http://dx.doi.org/10.1021/bi00794a005>

SGD project. *Saccharomyces* Genome Database, <http://www.yeastgenome.org>

Shock, E. L., Oelkers, E. H., Johnson, J. W., Sverjensky, D. A. and Helgeson, H. C. (1992) Calculation of the thermodynamic properties of aqueous species at high pressures and temperatures: Effective electrostatic radii, dissociation constants and standard partial molal properties to 1000 °C and 5 kbar. *J. Chem. Soc. Faraday Trans.* **88**, 803–826. <http://dx.doi.org/10.1039/FT9928800803>

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

findit

Multidimensionally Optimize Chemical Activities

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 target function of the metastable equilibrium chemical activities of the species of interest.

Usage

```
findit (lims = list(), target = "cv", n = NULL, iprotein = NULL,
       do.plot = TRUE, T = 25, P = "Psat", res = NULL, labcex = 0.6,
       loga.ref = NULL, as.residue = FALSE, loga.tot = 0)
## 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.
target	character, target statistic to optimize.
n	numeric, number of iterations.
res	numeric, grid resolution (number of points on one edge).
iprotein	numeric, indices of proteins.
do.plot	logical, make a plot?
T	numeric, temperature.
P	numeric, pressure; or character, "Psat".
labcex	numeric, character expansion for plot labels.
loga.ref	numeric, reference logarithms of activity of species.

<code>as.residue</code>	logical, use the activities of residues instead of proteins?
<code>loga.tot</code>	numeric, logarithm of total activity of residues (or other immobile component).
<code>x</code>	list, object of class <code>findit</code> .
<code>which</code>	numeric, which of the parameters to plot.
<code>mar</code>	numeric, plot margin specification.
<code>xlab</code>	character, x-axis label.
<code>...</code>	additional arguments passed to <code>plot</code> .

Details

`findit` implements a gridded optimization to find the minimum or maximum value of `target` as a function of one or more of the chemical activities, temperature and/or pressure whose ranges are listed in `lims`. Any target available in `revisit` can be optimized. Generally, the system (`basis` species and `species` of interest) must be set up before calling this function. If `iprotein` is supplied, indicating a set of proteins to use in the calculation, the definition of the `species` is not required.

`lims` is a list, each element of which contains two values indicating the range of the parameter indicated by the `names` of `lims`. The `names` should be formula(s) of one or more of the basis species, 'T' and/or 'P'. If the latter two are missing, the calculations are performed at isothermal and/or isobaric conditions indicated by T and P.

It can be used for one, two, or more parameters. If `nd` is the number of parameters (dimensions), default values of `n` and `res` come from the following table. These settings are selected mostly for quickness of testing the function in high-dimensional space. Detailed studies of a system might have to use more iterations and/or higher resolutions.

<code>nd</code>	<code>n</code>	<code>res</code>	<code>res^nd</code>	<code>rat</code>
1	4	128	128	2/(1+sqrt(5))
2	6	64	4096	2/(1+sqrt(5))
3	6	16	4096	0.75
4	8	8	4096	0.8
5	12	6	7776	0.8
6	12	4	4096	0.85
7	12	4	16384	0.85

The function performs `n` iterations. At first, the limits of the parameters given in `lims` define the extent of a `nd`-dimensional box around the space of interest. The value of `target` is calculated at each of the res^{nd} grid points and an optimum value located (see `revisit` and `where.extreme`). 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 `rat` * the length in the previous step. If the limits of any of the parameters extend beyond those in `lims`, they are pushed in to fit (preserving the difference between them).

`plot.findit` plots the values of the parameters and the target statistic as a function of the number of iterations.

Value

findit returns a list having class findit with elements value (values of the parameters, and value of the target statistic, at each iteration), lolim (lower limits of the parameters) and hilim (upper limits of the parameters).

Examples

```
# an inorganic example: sulfur species
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-"))
# to minimize the standard deviations of the loga of the species
target <- "sd"
# this one gives logfO2=-27.8
f1 <- findit(list(O2=c(-50, -15)), target, T=325, P=350, n=3)
title("SD of equilibrium log activities of sulfur species")
# this one gives logfO2=-30.0 pH=9.38
f2 <- findit(list(O2=c(-50, -15), pH=c(0, 14)), target, T=325, P=350, res=16, n=4)
title("SD of equilibrium log activities of sulfur species")

# test the findit function in three dimensions
# first calculate equilibrium activities of some proteins
# using specified activities of basis species
ip <- 1:20
basis("CHNOS+")
basis("CO2", -pi)      # -3.141593
basis("H2O", -exp(1))  # -2.718282
basis("NH3", -sqrt(2)) # -1.414214
a <- affinity(iprotein=ip)
# scale relative abundances such that total activity of residues
# is unity (since loga.tot=0 is default for findit)
d <- diagram(a, do.plot=FALSE, logact=0)
loga.ref <- as.numeric(d$logact)
# return to default values for activities of basis species,
basis("CHNOS+")
a <- affinity(iprotein=ip)
d <- diagram(a, do.plot=FALSE, logact=0)
# we have diverged from the reference activities of proteins
r0 <- revisit(d, "rmsd", loga.ref, main="")
title(main=paste("log activities of 20 proteins for basis activities",
  "from numerical constants (ref) and CHNOSZ default (calc)", sep="\n"))
# now find the activities of the basis species, within search
# intervals, that get us close to reference activities of proteins
f <- findit(lims=list(CO2=c(-5, 0), H2O=c(-5, 0), NH3=c(-5, 0)),
  "rmsd", 10, iprotein=ip, loga.ref=loga.ref, res=16)
title(main="RMSD of log activities from reference values")
# -pi, -e and -sqrt(2) were approximately retrieved!
-sapply(f$value, tail, 1)[1:3]
# we can plot the trajectories
```

```
plot(f, mar=c(4, 5, 2, 2))
```

get.expr

Protein Expression Data

Description

Get abundance data from a protein expression experiment and add the proteins to the working instance of CHNOSZ.

Usage

```
get.expr(file, idcol, abundcol, seqfile, filter=NULL,
         is.log=FALSE, loga.total = 0)
```

Arguments

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.
seqfile	character, name of the FASTA file with protein sequences.
filter	list, optional filters to apply.
is.log	logical, are the abundances in the file in logarithmic (base 10) units?
loga.total	numeric, logarithm of total activity of residues.

Details

This function reads a CSV *file* that contains protein sequence IDs and protein abundance data. The header (first line) of this file contains the column names; the names of the columns holding the sequence IDs and protein abundances are indicated by *idcol* and *abundcol*, respectively. The sequence IDs are searched for in the accession lines in the FASTA file indicated by *seqfile* (using `grep`); a match can occur in any part of an accession line, and the first such match is used. Any IDs that are NA or can not be found in *seqfile* are excluded from further consideration. The amino acid compositions of the matched proteins are computed (using `read.fasta`) and are added to the inventory of proteins in CHNOSZ (`thermo$protein`).

The function returns values of the logarithms of activities of the proteins. We associate molality with activity (i.e., activity coefficients are implicitly unity). If *loga.total* is not NULL, the abundances of the proteins from the data file are scaled to give a logarithm of total activity of amino acid residues equal to the value in *loga.total*, usually set to zero (see `unitize`). This operation preserves the relative abundances of the proteins. If the abundances of the proteins in the file are already in logarithmic units, set *is.log* to TRUE.

If *seqfile* is one of 'SGD', 'ECO' or 'HUM' it refers to the database of amino acid compositions of proteins packaged with CHNOSZ for either *Saccharomyces cerevisiae*, *Escherichia coli* or *Homo sapiens*. In this case, the search for matching IDs is performed using `get.protein`.

The data file can be filtered by using *filter*. This argument should be a list with one element, the name of which indicates the column to apply the filter to, and the value of which is a search term.

Value

Returns a list with objects `iprotein` (the indices of the proteins in `thermo$protein`) and `loga.ref` (the logarithms of activities of the proteins).

See Also

`findit` for finding combinations of chemical activities that optimize the fit of metastable protein assemblages to experimental protein abundances.

Examples

```
# let's use a sample data file
file <- system.file("extdata/abundance/ISR+08.csv", package="CHNOSZ")
# read the abundances and get the proteins from ECO.csv
expr <- get.expr(file, "ID", "emPAI", "ECO")
# what if we just wanted kinases?
expr <- get.expr(file, "ID", "emPAI", "ECO", list(description="kinase"))
# the abundances were scaled so that the total activity of residues is unity
pl <- protein.length(-expr$iprotein)
stopifnot(all.equal(sum(pl*10^expr$loga), 1))
# see the 'protactiv' vignette for comparison with equilibrium calculations

# if you want to read the protein sequences from a FASTA file...
# e <- get.expr(file, "ID", "emPAI", "ECOLI.fasta")
```

get.protein

Proteins from Model Organisms

Description

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

Usage

```
get.protein(protein, organism, abundance = NULL, pname = NULL,
  average = TRUE, digits = 1)
yeastgfp(location, exclusive = TRUE)
```

Arguments

<code>protein</code>	character, name of protein or stress response experiment.
<code>organism</code>	character, organism ('ECO', 'SGD') or 'YeastGFP'.
<code>abundance</code>	numeric, stoichiometry of proteins applied to sums of compositions.
<code>pname</code>	character, names of proteins.
<code>average</code>	logical, return an average composition of the proteins?

digits	numeric, number of decimal places to round the amino acid counts.
location	character, name of subcellular location (compartment).
exclusive	logical, report only proteins exclusively localized to a compartment?

Details

When `protein` contains one or more Ordered Locus Names (OLN) or Open Reading Frame names (ORF), `get.protein` retrieves the amino acid composition of the respective proteins in *Escherichia coli* or *Saccharomyces cerevisiae* (for `organism` equal to 'ECO' or 'SGD', respectively). The calculation depends on presence of the objects `thermo$ECO` and `thermo$SGD`, which contain the amino acid compositions of proteins in these organisms. If `protein` is instead a name of one of the stress response experiments contained in `thermo$stress`, e.g. 'low.C' or 'heat.up', the function returns the amino acid compositions of the corresponding proteins.

If the abundances of the proteins are given in `abundance`, the individual protein compositions are multiplied by these values then summed into an overall composition; the average is taken if `average` is TRUE; then the amino acid frequencies are rounded to the number of decimal places specified in `digits`. Unless names for the new proteins are given in `pname`, they are generated using the values in `protein`.

The `yeastgfp` function returns the identities and abundances of proteins with the requested subcellular localization (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). The values returned by `yeastgfp` can be fed to `get.protein` in order to get the amino acid compositions of the proteins.

Value

For `get.protein`, returns the amino acid composition(s) of the specified protein(s), or a single overall composition if `abundance` is not NULL. `yeastgfp` returns a list with elements `yORF` and `abundance`, unless `location` is NULL, when the function returns (`invisible-y`) the names of all locations.

References

- Boer, V. M., de Winde, J. H., Pronk, J. T. and Piper, M. D. W. (2003) The genome-wide transcriptional responses of *Saccharomyces cerevisiae* grown on glucose in aerobic chemostat cultures limited for carbon, nitrogen, phosphorus, or sulfur. *J. Biol. Chem.* **278**, 3265–3274. <http://dx.doi.org/10.1074/jbc.M209759200>
- Dick, J. M. (2009) Calculation of the relative metastabilities of proteins in subcellular compartments of *Saccharomyces cerevisiae*. *BMC Syst. Biol.* **3**:75. <http://dx.doi.org/10.1186/1752-0509-3-75>
- Richmond, C. S., Glasner, J. D., Mau, R., Jin, H. F. and Blattner, F. R. (1999) Genome-wide expression profiling in *Escherichia coli* K-12. *Nucleic Acids Res.* **27**, 3821–3835. <http://nar.oxfordjournals.org/cgi/content/abstract/27/19/3821>

Tai, S. L., Boer, V. M., Daran-Lapujade, P., Walsh, M. C., de Winde, J. H., Daran, J.-M. and Pronk, J. T. (2005) Two-dimensional transcriptome analysis in chemostat cultures: Combinatorial effects of oxygen availability and macronutrient limitation in *Saccharomyces cerevisiae*. *J. Biol. Chem.* **280**, 437–447. <http://dx.doi.org/10.1074/jbc.M410573200>

See Also

The output of `get.protein` can be used as input to `add.protein` to add the proteins to the `thermo$protein` data frame in preparation for further calculations (see examples below).

Examples

```
## basic examples of get.protein
# amino acid composition of two proteins
get.protein(c("YML020W", "YBR051W"), "SGD")
# average composition of proteins
get.protein(c("YML020W", "YBR051W"), "SGD",
  abundance=1, pname="PROT1_NEW")
# 1 of one and 1/2 of the other
get.protein(c("YML020W", "YBR051W"), "SGD",
  abundance=c(1, 0.5), average=FALSE, pname="PROT2_NEW")
# compositions of proteins induced in carbon limitation
get.protein("low.C", "SGD")

## overall composition of proteins exclusively localized
## to cytoplasm of S. cerevisiae with reported expression levels
y <- yeastgfp("cytoplasm")
p <- get.protein(y$yORF, "SGD", y$abundance, "cytoplasm")
# add the protein and calculate its properties
i <- add.protein(p)
protein(i)

## speciation diagram for ER.to.Golgi proteins (COPII coat
## proteins) as a function of logfO2, after Dick, 2009
y <- yeastgfp("ER.to.Golgi")
# take out proteins with NA experimental abundance
ina <- which(is.na(y$abundance))
y$yORF <- y$yORF[-ina]
y$abundance <- y$abundance[-ina]
# get the amino acid compositions of the proteins
p <- get.protein(y$yORF, "SGD")
ip <- add.protein(p)
# 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(O2=c(-80, -73), iprotein=ip, loga.protein=logact)
# make a speciation diagram
diagram(a, ylim=c(-4.9, -2.9))
```

```

# where we are closest to experimental log activity
logfO2 <- rep(-78,length(ip))
abline(v=logfO2[1],lty=3)
# scale experimental abundances such that
# total activity of residues is unity
logact.expt <- unitize(log10(y$abundance),pl)
# plot experimental log activity
points(logfO2,logact.expt,pch=16)
text(logfO2+0.5,logact.expt,y$yORF)
# add title
title(main=paste("ER.to.Golgi; points - relative abundances",
  "from YeastGFP. Figure after Dick, 2009",sep="\n"))

## Chemical activities of model subcellular proteins
# speciation diagram as a function of logfO2, after Dick, 2009
basis("CHNOS+")
names <- yeastgfp()
# calculate amino acid compositions using "get.protein" function
for(i in 1:length(names)) {
  y <- yeastgfp(names[i])
  p <- get.protein(y$yORF,"SGD",y$abundance,names[i])
  add.protein(p)
}
species(names,"SGD")
# set unit activity of residues
pl <- protein.length(thermo$species$name)
species(NULL,unitize(thermo$species$logact,pl))
res <- 200
a <- affinity(O2=c(-82,-65,res))
mycolor <- topo.colors(6)[1:4]
mycolor <- rep(mycolor,times=rep(6,4))
logact <- diagram(a,balance="PBB",names=names,ylim=c(-5,-3),legend.x=NULL,
  col=mycolor,lwd=2)$logact
# so far good, but how about labels on the plot?
for(i in 1:length(logact)) {
  myloga <- as.numeric(logact[[i]])
  # don't take values that lie above the plot (vacuole in this example)
  myloga[myloga > -3.1] <- -999
  imax <- which.max(myloga)
  adj <- 0.5
  if(imax > 180) adj <- 1
  if(imax < 20) adj <- 0
  text(seq(-82,-65,length.out=res)[imax],logact[[i]][imax],
    labels=names[i],adj=adj)
}
title(main=paste("Subcellular proteins of S. cerevisiae, after Dick, 2009",
  describe(thermo$basis[-5,]),sep="\n"),col.main=par("fg"),cex.main=0.9)

## Oxygen fugacity - activity of H2O predominance
## diagrams for proteologs for 23 YeastGFP localizations
# 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")
nloc <- c(4, 5, 3, 4, 4, 3)
inames <- 1:length(names)
# define the system
basis("CHNOS+")
# calculate amino acid compositions using "get.protein" function
for(i in 1:length(names)) {
  y <- yeastgfp(names[i])
  p <- get.protein(y$yORF, "SGD", y$abundance, names[i])
  add.protein(p)
}
species(names, "SGD")
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\n", describe(thermo$basis[-c(2, 5), ])), cex=1.5)
opar <- par(mar=c(3, 4, 1, 1), xpd=TRUE)
for(i in 1:length(nloc)) {
  cex.axis <- 0.75
  # uncomment the following and dev.off() below to generate png files
  #png(paste(i, "png", sep="."), width=300, height=250); cex.axis <- 1
  diagram(a, balance="PBB", names=names[inames],
    ispecies=inames, cex.axis=cex.axis)
  label.plot(letters[i])
  title(main=paste(length(inames), "locations"))
  #dev.off()
  # take out the stable species
  inames <- inames[-(1:nloc[i])]
}
# make an animated gif from png files (with ImageMagick convert tool)
#system(paste("convert -delay 100 1.png 1.png 1.png 2.png",
#  "3.png 4.png 5.png 6.png 6.png 6.png yeast.gif"))
# return to plot defaults
layout(matrix(1))
par(opar)

## Compare calculated and experimental relative abundances
## of proteins in a subcellular location, after Dick, 2009
# get the amino acid composition of the proteins
loc <- "vacuolar.membrane"
y <- yeastgfp(loc)
ina <- which(is.na(y$abundance))
p <- get.protein(y$yORF[-ina], "SGD")
add.protein(p)
# set up the system

```

```

basis("CHNOS+")
# this is the logfO2 value that gives the best fit (see paper)
basis("O2",-74)
is <- species(p$protein,p$organism)
np <- length(is)
pl <- protein.length(species()$name)
# we use unitize so total activity of residues is unity
loga <- rep(0,np)
species(1:np,unitize(loga,pl))
a <- affinity()
d <- diagram(a,do.plot=FALSE)
calc.loga <- as.numeric(d$logact)
expt.loga <- unitize(log10(y$abundance[-ina]),pl)
# which ones are outliers
rmsd <- sqrt(sum((expt.loga-calc.loga)^2)/np)
residuals <- abs(expt.loga - calc.loga)
iout <- which(residuals > rmsd)
pch <- rep(16,length(is))
pch[iout] <- 1
# the colors reflect average oxidation number of carbon
# corrects misassigned colors in Figs. 5 and 6 of Dick 2009
ZC <- ZC(thermo$obigt$formula[species()$ispecies])
col <- rgb(0.15-ZC,0,0.35+ZC,max=0.5)
# there is a color-plotting error on line 567 of the plot.R file
# of Dick, 2009 that can be reproduced with
#col <- rep(col,length.out=9)
xlim <- ylim <- extendrange(c(calc.loga,expt.loga))
thermo.plot.new(xlim=xlim,ylim=ylim,xlab=expression(list("log"*italic(a),
  "calc")),ylab=expression(list("log"*italic(a),"expt")))
points(calc.loga,expt.loga,pch=pch,col=col)
lines(xlim,ylim+rmsd,lty=2)
lines(xlim,ylim-rmsd,lty=2)
title(main=paste("Calculated and experimental relative abundances of\n",
  "proteins in ",loc,",", after Dick, 2009",sep=""),cex.main=0.95)

### examples for stress response experiments

## predominance fields for overall protein compositions induced by
## carbon, sulfur and nitrogen limitation
## (experimental data from Boer et al., 2003)
expt <- c("low.C","low.N","low.S")
for(i in 1:length(expt)) {
  p <- get.protein(expt[i],"SGD",abundance=1)
  add.protein(p)
}
basis("CHNOS+")
basis("O2",-75.29)
species(expt,"SGD")
a <- affinity(CO2=c(-5,0),H2S=c(-10,0))
diagram(a,balance="PBB",names=expt,color=NULL)
title(main=paste("Proteins induced by",
  "carbon, sulfur and nitrogen limitation",sep="\n"))

```

```

## predominance fields for overall protein compositions
## induced and repressed in an/aerobic carbon limitation
## (experiments of Tai et al., 2005)
# the activities of glucose, 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", "H+"),
      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 <- get.protein(expt[i], "SGD", abundance=1)
  add.protein(p)
}
species(expt, "SGD")
a <- affinity(C6H12O6=c(-35, -20), H2=c(-20, 0))
diagram(a, color=NULL, as.residue=TRUE)
title(main=paste("Average protein residue composition in",
                "an/aerobic carbon limitation in yeast", sep="\n"))

```

 info

Search the Thermodynamic Database

Description

Search for species by name or formula, and retrieve their thermodynamic properties and parameters.

Usage

```
info(species = NULL, states = NULL, quiet = FALSE, return.approx = TRUE)
```

Arguments

species	character, names or formulas of species, or numeric, indices of species in the thermodynamic database.
states	numeric, physical states of the species.
quiet	logical, produce fewer messages and make fewer tests?
return.approx	logical, return the indices of approximately matching species?

Details

This function searches the names or chemical formulas of `species` in the thermodynamic database (`thermo$obigt`). Searches can optionally be limited to certain physical states (among 'aq', 'cr', 'gas', 'liq' as well as 'cr1', 'cr2' etc.) For each of the `species` that is matched, the index of that species is appended to the return value. If an exact match of a species is not located, `info` searches the database for similar names or formulas; if any of these are found, the results are summarized on the screen, and the indices of the approximately matching species are included in the return value (if `return.approx` is TRUE). Species that have no exact or approximate matches are indicated by NA in the return value.

If `states` is NULL (the default) and there are multiple matches for the name of a species, the species whose state is that in `thermooptstate` ('aq' by default) is selected, or failing that, the first matching species is taken. The exceptions to this rule is 'O2', for which the state defaults to 'gas'.

If `species` is numeric instead of character, the corresponding rows of the thermodynamic database are returned, 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.

The names of proteins are distinguished from those of other species insofar as they contain an underscore character, as in 'LYSC_CHICK'. If the name of a protein is provided to `info` and the composition of the protein can be found using `protein`, the thermodynamic properties and parameters of the nonionized protein (calculated using `protein`) 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.

If `quiet` is FALSE, several checks of self consistency among the thermodynamic properties and parameters are performed. 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. In addition, the value of G calculated from those of H and S is compared with the tabulated value and a message printed if the difference exceeds 1000 cal mol⁻¹. Likewise, if the values of standard molal heat capacity or volume calculated from the equations-of-state parameters differ (by greater than 1 cal mol K⁻¹ or 1 cm³ mol⁻¹) from the tabulated values, messages to this effect are printed.

Value

If `species` is character of length n and no approximate matches are found, the `invisible` return is a numeric vector of length n containing for each species its index in the thermodynamic database or NA. If two or more approximate matches are located for any of the species, the return value is instead a list of length n , each element of which contains the matching index or indices of the species or NA. If `species` is numeric, the corresponding rows of the thermodynamic database are visibly returned, after removing order-of-magnitude multipliers.

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

```

## basic operation
## Not run:
# run a consistency check on each species in the database
# (marked dontrun because it takes a while)
info(check=TRUE)
## End(Not run)

## species information
# search for something named (or whose formula is) "Fe"
si <- info("Fe")
# use the number to get the full record
info(si)
# it is possible to get a range of records
info(si:(si+3))

## dealing with states
# default order of preference for names: aq > gas > cr, liq
info(c("methane", "ethanol", "glycinate")) # aq, aq, aq
info(c("adenosine", "alanine", "hydroxyapatite")) # aq, aq, cr
# state argument overrides the default
info(c("ethanol", "adenosine"), state=c("gas", "cr"))
# formulas default to aqueous species, if available
info(c("CH4", "CO2", "CS2", "MgO")) # aq, aq, gas, cr
# state argument overrides the default
info(c("CH4", "CO2", "MgO"), "gas") # gas, gas, NA
# exceptions to the aqueous default is O2
info("O2") # gas

## partial name or formula searches
info("ATP")
info("thiol")
info("MgC")
# add an extra character to refine a search
# or to search using terms that have exact matches
info("MgC ")
info("acetate ")
info(" H2O")

```

ionize

Ionization Properties of Proteins

Description

Calculate the charges of proteins and contributions of ionization to their chemical affinities of formation reactions or other thermodynamic properties.

Usage

```
ionize(affinity = NULL, other = NULL)
```

Arguments

<code>affinity</code>	affinities of formation of ionizable groups and nonionized proteins.
<code>other</code>	other properties of ionizable groups and nonionized proteins.

Details

Most of the time, the user doesn't need to call this function directly. It is called by `affinity` if proteins are among the species of interest, 'H+' is in the basis, and `thermooptionize` is TRUE. This function can also be used from the top level to calculate the ionization state and ionization contributions to thermodynamic properties other than the chemical affinity (see examples below). The ionization model used by this function takes account of the ionization constants of amino acid sidechain groups and the terminal groups in proteins as a function of temperature and pressure calculated using the properties and parameters taken from Dick et al., 2006.

`ionize` responds to different argument configurations. `affinity` and `other` both refer to the list contained in the `values` element of the output of `affinity`. If both these arguments are missing, `species` is called to load the ionizable sidechain and backbone groups of proteins into the global species list. If `other` is missing, the function calculates and outputs the affinities of formation of the ionized proteins. If `other` is present, it contains the values of other properties, for example 'Cp', for which ionization values are calculated. Finally, if `other` is identical to `affinity`, the function outputs the calculated net charges (overall ionization states) of the proteins. Note that `ionize` only computes values for proteins; those for other species are returned unaltered in the output.

Value

List, each element of which corresponds to the affinity of the formation reaction or other property of an ionized protein.

References

Dick, J. M., LaRowe, D. E. and Helgeson, H. C. (2006) Temperature, pressure, and electrochemical constraints on protein speciation: Group additivity calculation of the standard molal thermodynamic properties of ionized unfolded proteins. *Biogeosciences* **3**, 311–336. <http://www.biogeosciences.net/3/311/2006/bg-3-311-2006.html>

Makhatadze, G. I. and Privalov, P. L. (1990) Heat capacity of proteins. 1. Partial molar heat capacity of individual amino acid residues in aqueous solution: Hydration effect. *J. Mol. Biol.* **213**, 375–384. [http://dx.doi.org/10.1016/S0022-2836\(05\)80197-4](http://dx.doi.org/10.1016/S0022-2836(05)80197-4)

Privalov, P. L. and Makhatadze, G. I. (1990) Heat capacity of proteins. II. Partial molar heat capacity of the unfolded polypeptide chain of proteins: Protein unfolding effects. *J. Mol. Biol.* **213**, 385–391. [http://dx.doi.org/10.1016/S0022-2836\(05\)80198-6](http://dx.doi.org/10.1016/S0022-2836(05)80198-6)

See Also

[protein.info](#) and [residue.info](#) use this function to display ionization states of proteins.

Examples

```

### Direct Interaction with 'ionize'

## Charge of LYSC_CHICK as a function of pH and T
# After Fig. 10 of Dick et al., 2006
basis(c("CO2", "H2O", "NH3", "H2S", "O2", "H+"), rep(999, 6))
species("LYSC_CHICK")
# add the ionizable groups
ionize()
# get the affinities and charges (along equal temperature increments)
T <- c(25, 150, 6); pH <- c(0, 14, 50)
x <- affinity(pH=pH, T=T)
z <- ionize(x$values, x$values)
# plot charges at the temperatures we're interested in
plot(z[[1]][, 1], x=seq(0, 14, length.out=50), type="l", xlab="pH",
     ylab="net charge (Z)") # 25 deg C
lines(z[[1]][, 4], x=seq(0, 14, length.out=50), col="red") # 100 deg C
lines(z[[1]][, 6], x=seq(0, 14, length.out=50), col="orange") # 150 deg C
text(x=c(12, 10, 9), y=c(-15, -16, -18), labels=paste("T=", c(25, 100, 150), sep=""))
# if cysteine is oxidized (to cystine disulfide bonds) it may not ionize.
# suppress its ionization by upping energy of the ionized group
mod.obigt("[Cys-]", G=999999)
x <- affinity(pH=pH, T=25)
z <- ionize(x$values, x$values)
lines(z[[1]], x=seq(0, 14, length.out=50), lty=2)
text(x=12, y=-7, "T=25, oxidized")
# add experimental points
RT71 <- read.csv(system.file("extdata/cpetc/RT71.csv", package="CHNOSZ"))
points(RT71$pH, RT71$Z)
title(main=paste("Ionization of unfolded LYSC_CHICK\n",
                "Experimental points at 25 degC from Roxby and Tanford, 1971"),
      cex.main=0.9)
# forget our changes to thermo$obigt for next examples
data(thermo)

## Heat capacity of LYSC_CHICK as a function of T
basis("CHNOS+"); species("LYSC_CHICK")
pH <- c(5, 9, 3); T <- c(0, 100, 25)
T.values <- seq(T[1], T[2], length.out=T[3])
# add the ionizable groups
ionize()
a <- affinity(pH=pH, T=T)
# values for non-ionized protein
c <- affinity(pH=pH, T=T, property="Cp.species")
plot(T.values, c$values[[1]][1, ],
     xlab=axis.label("T"), ylab=axis.label("Cp"),
     ylim=c(5000, 8000), type="l", mgp=c(2.2, 0.2, 0))
# values for ionized protein
cp <- ionize(a$values, c$values)
for(i in 1:3) {
  lines(seq(T[1], T[2], length.out=T[3]), cp[[1]][i, ], lty=2)
}

```

```

    text(80,cp[[1]][i,][21],paste("pH=",pH[i],sep=""))
  }
# Makhatadze and Privalov's group contributions
T.MP <- c(5,25,50,75,100,125)
points(T.MP,convert(MP90.cp(T.MP,"LYSC_CHICK"),"cal"))
# Privalov and Makhatadze's experimental values
e <- read.csv(system.file("extdata/cpetc/PM90.csv",package="CHNOSZ"))
e <- e[e$protein=="LYSC_CHICK",]
points(e$T,convert(e$Cp,"cal"),pch=16)
title(main=paste("Calc'd heat capacity of LYSC_CHICK:",
  "non/ionized(solid/dashed);\n",
  "Makhatadze+Privalov 1990 (open, calc; filled, expt)",
  cex.main=0.9)

### Metastability calculations using 'ionize'

## Eh-pH diagrams for intra/extracellular proteins
organism <- c("PYRFU","ECOLI","YEAST")
intracellular <- c("AMPM","AMPM","AMPM1")
extracellular <- c("O08452","AMY1","PST1")
basis("CHNOSe") # for Eh we need electrons
mycol <- c("red","green","blue")
for(i in 1:3) {
  species(delete=TRUE)
  species(c(intracellular[i],extracellular[i]),organism[i])
  if(i == 1) add <- FALSE else add <- TRUE
  a <- affinity(pH=c(0,14),Eh=c(-1,0))
  diagram(a,add=add,color=NULL,names=species()$name,
    col=mycol[i],col.names=mycol[i])
}
title(main=paste("Intracellular (AMPM) and extracellular proteins\n",
  describe(thermo$basis[1:4,])))

## Buffer + ionization: Metastabilities of
## thiol peroxidases from model bacteria
## (ECOLI, BACSU mesophile; AQUAE thermophile,
## THIDA acidophile, BACHD alkaliphile)
basis("CHNOS+")
organisms <- 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(25,80,200))
diagram(a,color=NULL)
title(main=paste("Thiol peroxidases from bacteria\n",
  describe(thermo$basis[-6,],T=NULL),cex.main=0.9)

## Buffer + ionization: Metastable assemblage
## for E. coli sigma factors on a T-pH diagram

```

```

# (sigma factors 24, 32, 38, 54, 70, i.e.
# RpoE, RpoH, RpoS, RpoN, RpoD)
proteins <- c("RPOE", "RP32", "RPOS", "RP54", "RPOD")
basis("CHNOS+")
basis("pH", 7.4)
# define and set the buffer
change("_sigma", paste(proteins, "ECOLI", sep="_"))
basis(c("CO2", "NH3", "H2S", "O2"), "sigma")
logact <- affinity(return.buffer=TRUE, T=25)
# Set the activities of the basis species to constants
# corresponding to the buffer, and diagram the relative
# stabilities as a function of T and pH
basis(c("CO2", "NH3", "H2S", "O2"), as.numeric(logact))
species(paste(proteins, "ECOLI", sep="_"))
a <- affinity(pH=c(5, 10), T=c(10, 40))
diagram(a, balance="PBB", residue=FALSE)
title(main=paste("Sigma factors in E. coli\n",
  describe(thermo$basis[-6, ], T=NULL), cex.main=0.95)

```

makeup

Chemical Formulas and Compositions

Description

Convert between different representations of chemical compositions of species.

Usage

```
makeup(compound = "", component = NULL)
```

Arguments

compound	character, the formula of a compound; other types are possible (see below).
component	numeric, coefficient used to sum makeups; other types are possible (see below).

Details

This function performs operations on objects representing chemical compositions, or formulas, of compounds. In the simple case, when `compound` is a character object containing a chemical formula as a string, the return value is a dataframe object with a single column named `count` that lists the coefficients of each of the elements in the formula. With other combinations of arguments, this function can be used to sum compositions, count only specified elements, generate string representations of formulas from dataframes, or perform other operations listed in the table below.

The elements in a formula are symbolized by an uppercase letter followed by zero or more lowercase letters. Multiple pairs of parentheses are allowed, but may not be nested. Coefficients after the elemental symbols or parentheses can be non-integer values and/or negative. Charge is denoted (at the end of a formula, or preceding an asterisk or colon) by a value which can be non-integer,

optionally preceded by '+' or '-' (Hence, 'Fe+3' stands for ferric iron; 'Fe3+' is interpreted as three iron atoms with a net charge of plus one). An asterisk or a colon indicates to add the composition that follows, which may have a leading numerical coefficient. Some examples of valid formulas are 'Rh (SO4) 3-4', 'KAl3 (OH) 6 (SO4) 2' and 'Ca (Al2Si7) O18*6H2O'.

Charge is treated here like a chemical element. The elemental symbol used for it, 'Z', can appear in chemical formulas, and has atomic number zero and oxidation state +1. For example, the formula of the ammonium ion is written commonly as 'NH4+'; in **CHNOSZ** this same composition can also be identified with 'NH4Z'. Likewise, the electron is represented as 'Z-1', and the common symbol for the electron, 'e-', does not appear in chemical formulas in this package. Using this framework, any computations that are based on the complete stoichiometries of charged species, if they impose conservation of elements, account simultaneously for conservation of mass and charge.

Value

When this function is called with a single argument that is a string representation of a formula, the elemental composition, or *makeup* is returned as a dataframe of one column with *n* rows, where *n* is the number of elements present. If the single argument is other than character, or there is more than one argument, the action of the function is determined by the characteristics of the arguments. In the summary below, *makeup* indicates a dataframe, and *formula* refers to a string (character) representation of a formula. *elements* represents a dataframe with a single row.

(*formula*) Return *makeup*, the elemental composition of the formula.

(*makeup,makeup*) Return *makeup*, the sum of the two arguments.

(numeric,numeric) Return *makeup*, the sum of the makeups of the species identified in the first argument (corresponding to rownumber of `thermo$obigt`), each multiplied by the coefficient given in the second argument (taken as 1 if missing).

(*formula,numeric*) Return a list which contains the word (name of or coefficient on an element) beginning at the position in the formula string specified by the second argument, and the length of the word.

(*formula,formula*) Return *makeup*, sum of the two arguments.

(*makeup,character*) Return *elements*: transpose *makeup*, using all the elements (if the second argument is ""), or only those specified in the second argument.

(*makeup,TRUE*) Return *elements*, **not** the elements, but the stoichiometry of the basis species in the formation reaction for this compound. Requires prior definition of basis species; see `basis`.

(*elements,FALSE*) Return *elements*, change entries in the dataframe that are less than the value in `thermooptcutoff` to zero.

(*formula,character*) Return character, the unique elements contained in the *formulas* (character argument is necessary but ignored).

(list of *makeup,character*) Return character, the unique elements contained in the *makeups* (character argument is necessary but ignored).

(*elements,character*) Return character, a formula representation (i.e., "CO2") of the *elements*, (character argument must be "" or the names of elements to consider).

See Also

Functions used by `makeup` to parse character objects include `substr`, and `c2s`. Functions that use `makeup` include `subcrt` (to check reaction balance), `species` (to generate coefficients in

formation reactions), and `GHS` (to calculate entropies of elements in formulas). The list of elements is contained in `thermo$element`.

Examples

```
## makeups (nrow = number of elements)
# the composition of a single compound (ncol=1)
makeup("CO2")
# negative coefficients
makeup("C-4O2") # minus four C's
makeup("C-4O-2") # interpreted as having charge -2
makeup("C-4O-2+0") # no charge
# sum the compositions
makeup(c("CO2", "CH4"))
# sum with coefficients
makeup(info(c("CO2", "CH4")), c(-1, 1))
# this one adds up to zero
makeup(c("C6H12O6", "C2H5OH", "CO2"), c(-1, 2, 2))

## formulas (ncol = number of elements)
# as a dataframe
makeup("Zn(CH3CH2CH2CH2CO2)2", "")
print(m <- makeup(c("HCl", "H2O"), ""))
# as a character formula
makeup(makeup(info("glycinium"), "", ""))
makeup(m, "")
# C6H12O6
makeup(makeup(makeup("CHO6H11C5"), "", ""))

## charged species, electron
# these return the same:
makeup("NH4+")
makeup("NH4Z")
# constitution of the electron
makeup(info("e-"))
# this produce an incorrect makeup for the electron,
# and gives a warning because "e" is not in the
# table of elements.
makeup("e-")
# compositions of made-up compounds
makeup("CHNOS")
makeup("CHNOSZ")
makeup("CHNOSe") # Se is an element
```

Description

Retrieve the amino acid compositions or thermodynamic properties and equations of state parameters of proteins.

Usage

```
protein(protein, organism=NULL, online=thermo$opt$online, chains=1)
protein.residue(proteins)
protein.info(T=25)
residue.info(T=25)
```

Arguments

protein	character, names of proteins, protein identifiers, or amino acid sequences, or numeric, indices of proteins (rownumbers of <code>thermo\$protein</code>), or dataframe, protein compositions to sum into new protein.
organism	character, organism identifiers, or physical state.
proteins	character, names of proteins.
online	logical, try an online search if the specified protein(s) are not found locally?
chains	numeric, number of polypeptide chains in added proteins.
T	numeric, temperature in units specified by <code>nuts</code> .

Details

`protein` is a function to query the protein database and to perform group additivity calculations of the standard molal thermodynamic properties and equations of state parameters of proteins. In CHNOSZ, the database of amino acid compositions of proteins is located at `thermo$protein` and is populated when the package loaded. See the help for `thermo` for more information.

To distinguish names of proteins from those of other species, protein names in CHNOSZ have an underscore ("_") somewhere in their name, as in 'LYSC_CHICK'. If a protein name is submitted as a single argument to `protein` it is searched for in `thermo$protein`; if matches are found, the selected rows are returned. If protein and organism identifiers (e.g. 'LYSC' and 'CHICK', respectively) are provided, the rownumbers of matches in `thermo$protein` are returned.

If no match is found in `thermo$protein`, an online search is invoked, unless `online` is FALSE. (If `online` is NA, the default value of `thermooptonline`, the user is prompted whether the online search should be performed, and this response is stored in `thermooptonline`.) The function attempts a search of the SWISS-Prot database (Boeckmann et al., 2003). If the amino acid composition of the protein is successfully retrieved by the online search, that composition is stored in `thermo$protein`.

If `protein` is numeric, the compositional information found in that row(s) of `thermo$protein` is combined with sidechain and backbone group contributions to generate the standard molal thermodynamic properties and equations of state parameters of the proteins at 25 °C and 1 bar (Dick et al., 2006), and a dataframe of these values returned. The physical state of the proteins in this calculation is controlled by the value of `organism` ('aq' or 'cr'; NULL defaults to 'aq'). Note that the properties of aqueous (and crystalline) proteins calculated in this step are hypothetically completely nonionized proteins; the contributions by ionization to the chemical affinities of formation reactions

of aqueous proteins can be calculated during execution of `affinity` if the basis species contain 'H+'.

If `protein` is a `data.frame`, it is taken to contain the compositions of one or more proteins that are summed to define a new protein whose amino acid composition is returned. In this case, the argument `organism` should contain the name of the new protein, e.g. 'PROTEIN_NEW'.

The `protein` function modifies the database if `organism` is a valid name for a protein (it contains an underscore). The function assumes that `protein` contains the amino acid sequence of the new protein to be added to `thermo$protein`. The `chains` argument specifies the number of polypeptide chains that are in the molecule (0 for amino acid residues, 1 for amino acids and most proteins).

`protein.residue` generates average amino acid residue compositions of proteins. It takes the name(s) of one or more proteins (e.g. 'LYSC_CHICK'), retrieves their amino acid compositions from `thermo$protein`, and divides by the total number of amino acid residues in each of the proteins.

`protein.info` is a utility to tabulate some properties of proteins. A dataframe is returned containing for each protein that is among the `species` of interest, the name of the protein, its length, formula, and values of the standard molal Gibbs energy of the neutral protein, net charge, standard molal Gibbs energy of the ionized protein, and average oxidation number of carbon. The value of `T` indicates the temperature at which to calculate the Gibbs energies and net charge. Net charge and standard molal Gibbs energy of the ionized protein become NA if 'H+' is not among the basis species. The values are rounded at a set number of digits for display, and the values of Gibbs energy are in kcal/mol.

`residue.info` calculates the per-residue makeup of the proteins that have been loaded using `species`. This amounts to dividing the reaction coefficients in `thermo$species` by the length of the protein, but also takes into account the ionization state of the protein if 'H+' is one of the basis species. As with `protein.info`, the ionization state of the protein is calculated at the pH defined in `thermo$basis` and at the temperature specified by the `T` argument.

Value

If `protein` is one or more protein names, the matching row(s) of `thermo$protein`. If `protein` and `organism` are protein and organisms identifiers, rownumbers of `thermo$protein`. If `protein` is numeric, a dataframe with calculated thermodynamic properties and parameters of the neutral protein.

References

- Anderson, N. L. and Anderson, N. G. (2003) The human plasma proteome: History, character and diagnostic prospects (Vol. 1 (2002) 845-867). *Molecular and Cellular Proteomics* **2**, 50. <http://dx.doi.org/10.1074/mcp.A300001-MCP200>
- Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M.-C., Estreicher, A., Gasteiger, E., Martin, M. J., Michoud, K., Donovan, C., Phan, I., Pilbout, S. and Schneider, M. (2003) The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucleic Acids Res.* **31**, 365–370. <http://www.expasy.org>, accessed on 2007-12-19.
- Dick, J. M., LaRowe, D. E. and Helgeson, H. C. (2006) Temperature, pressure, and electrochemical constraints on protein speciation: Group additivity calculation of the standard molal thermo-

dynamic properties of ionized unfolded proteins. *Biogeosciences* **3**, 311–336. <http://www.biogeosciences.net/3/311/2006/bg-3-311-2006.html>

Dick, J. M. (2008) Calculation of the relative metastabilities of proteins using the CHNOSZ software package. *Geochem. Trans.* **9**:10. <http://dx.doi.org/10.1186/1467-4866-9-10>

See Also

Functions `info`, `subcrt`, `species` and others accept protein names as species arguments. Properties of ionized proteins can be calculated using `ionize` (usually implicitly called as part of a calculation of `affinity`). Compositions of proteins beyond those in `thermo$protein` (including yeast and *E. coli*) are also provided in CHNOSZ; see `get.protein` for examples.

Examples

```
### Interaction with the 'protein' function

## Thermodynamic properties of proteins
## get the composition of a protein
protein("BPT1_BOVIN")
## retrieve the rownumber of a protein in thermo$protein
iprotein <- protein("LYSC", "CHICK")
## calculate properties and parameters of aqueous protein
protein(iprotein)
## of crystalline protein
protein(iprotein, "cr")
## a call to info() causes the protein properties to
## be appended to thermo$obigt
info("LYSC_CHICK")
## thermodynamic properties can be calculated with subcrt()
subcrt("LYSC_CHICK")

### Table of properties of some proteins
basis("CHNOS+")
species(c("LYSC_CHICK", "CYC_BOVIN", "MYG_HORSE", "RNAS1_BOVIN"))
protein.info()
## the following gives the per-residue composition (i.e. formation
## reaction coefficients) for the ionized proteins
residue.info()

## Protein Data from Online Sources
## Not run:
## marked dontrun because it requires internet
## this asks to search SWISS-Prot
info("PRND_HUMAN")
## an online search can also be started from the
## "subcrt" function
subcrt("SPRN_HUMAN")
## End(Not run)

## Inputting protein compositions
## make a new protein
```

```

protein("GGSGG", "PROTEIN_TEST")
# a sequence can be pasted into the command line:
# type this
protein("
# then paste the sequence (this is it)
# and end the command by typing
", "PROTEIN_NEW")
# or use whatever name you want (with an underscore).

## Standard molal entropy of a protein reaction
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)
d <- describe(s$reaction,
  use.name=c(TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE))
text(170, -3, c2s(s2c(d, sep="="), sep="\n"), cex=0.8)

### Metastability calculations

## subcellular homologs of yeast glutaredoxin
## as a function of logfO2 - logaH2O, after Dick, 2009
basis("CHNOS+")
protein <- c("GLRX1", "GLRX2", "GLRX3", "GLRX4", "GLRX5")
loc <- c(" (C) ", " (M) ", " (N) ", " (N) ", " (M) ")
species(protein, "YEAST")
a <- affinity(H2O=c(-10, 0), O2=c(-85, -60))
diagram(a, names=paste(protein, loc))
title(main=paste("Yeast glutaredoxins (black) and residues (blue)\n",
  describe(thermo$basis[-c(2, 5), ])))
# note the difference when we set as.residue=TRUE to
# plot stability fields for the residue equivalents of the
# proteins instead of the proteins themselves ...
# the residue equivalent for one of the larger proteins appears
diagram(a, names=paste(protein, loc), as.residue=TRUE,
  add=TRUE, col="blue")

## surface-layer proteins from Methanococcus and others:
## a speciation diagram for surface layer proteins
## as a function of oxygen fugacity after Dick, 2008

```

```

# make our protein list
organisms <- c("METSC", "METJA", "METFE", "HALJP", "METVO",
  "METBU", "ACEKI", "BACST", "BACLI", "AERSA")
proteins <- c(rep("CSG", 6), rep("SLAP", 4))
proteins <- paste(proteins, organisms, sep="_")
# set some graphical parameters
lwd <- c(rep(3, 6), rep(1, 4))
lty <- c(1:6, 1:4)
# load the basis species and proteins
basis("CHNOS+")
species(proteins)
# calculate affinities
a <- affinity(O2=c(-100, -65))
# make diagram
d <- diagram(a, ylim=c(-5, -1), legend.x=NULL, lwd=lwd,
  ylab=as.expression(quote(log~italic(a[j]))), yline=1.7)
# label diagram
text(-80, -1.9, "METJA")
text(-74.5, -1.9, "METVO")
text(-69, -1.9, "HALJP")
text(-78, -2.85, "METBU", cex=0.8, srt=-22)
text(-79, -3.15, "ACEKI", cex=0.8, srt=-25)
text(-81, -3.3, "METSC", cex=0.8, srt=-25)
text(-87, -3.1, "METFE", cex=0.8, srt=-17)
text(-79, -4.3, "BACST", cex=0.8)
text(-85.5, -4.7, "AERSA", cex=0.8, srt=38)
text(-87, -4.25, "BACLI", cex=0.8, srt=30)
# add water line
abline(v=-83.1, lty=2)
title(main=paste("Surface-layer proteins",
  "After Dick, 2008", sep="\n"))

## relative metastabilities 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, "GSH-GSSG", -7))
basis("CO2", "gas")
prot <- c("CYC", "RNAS1", "BPT1", "ALBU", "INS", "PRIO")
species(prot, "BOVIN")
a <- affinity(T=c(0, 200))
d <- diagram(a, as.residue=TRUE, ylim=c(-2, 0.5))
# add some text labels
text(40, data.frame(d$logact)[25, ], prot)
title(main="Relative stabilities of bovine proteins on glutathione buffer")

## relative metastabilities of plasma proteins,
## using chemical activities of H2 and O2
# clean up basis species, species from previous example
data(thermo)
basis(c("CO2", "NH3", "H2S", "H2", "O2", "H+"))
basis("O2", "aq")
basis(c("CO2", "NH3", "H2S", "H+"), c(-3, -3, -10, -7))

```

```

f <- system.file("extdata/abundance/AA03.csv", package="CHNOSZ")
pdata <- read.csv(f, as.is=TRUE)
notna <- !is.na(pdata$name)
pname <- pdata$name[notna]
# take out insulin C peptide to show more proteins
pname <- pname[!pname %in% "INS.C"]
species(pname, "HUMAN")
a <- affinity(H2=c(-20,0), O2=c(-80,-60))
diagram(a)
title(main="Human Plasma Proteins")
# note that the darker colors go with higher abundances
# as reported by Anderson and Anderson, 2003
# add lines showing equilibrium activity of H2O
species(delete=TRUE)
species("H2O")
logaH2 <- seq(-20,0,length.out=128)
for(logaH2O in c(-5,0,5)) {
  species("H2O",logaH2O)
  a <- affinity(H2=logaH2)
  logaO2 <- diagram(a,what="O2",do.plot=FALSE)$logact[[1]]
  lines(logaH2,logaO2,lty=2)
  itext <- 72 + 5 * logaH2O
  lab <- paste("logaH2O =",logaH2O)
  text(logaH2[itext]+0.4,logaO2[itext],lab,srt=-64)
}

```

revisit

Diversity Calculations for Chemical Species

Description

Calculate species richness, or standard deviation, coefficient of variation or Shannon diversity index of activities or logarithms of activities of chemical species, and plot the results.

Usage

```

revisit(d, target = "cv", loga.ref = NULL,
  do.plot = NULL, col = par("fg"), yline = 2, ylim = NULL,
  ispecies = NULL, add = FALSE, cex = par("cex"), lwd = par("lwd"),
  mar = NULL, side = 1:4, xlim = NULL, labcex = 0.6, pch = 1,
  legend = "", legend.x = NULL, lpch = NULL, main = NULL,
  lograt.ref = NULL, plot.ext = TRUE)
extremes(z, target)
where.extreme(z, target, do.sat = FALSE)

```

Arguments

d list, output from [diagram](#), or list of logarithms of activities of species.

<code>target</code>	character, what statistic to calculate.
<code>loga.ref</code>	numeric, logarithm of activities for comparison statistics
<code>do.plot</code>	logical, make a plot?
<code>col</code>	character, color to use for points or lines.
<code>ylim</code>	numeric, margin line for y-axis label.
<code>ylim</code>	numeric, limits of y axis.
<code>ispecies</code>	numeric, which species to consider.
<code>add</code>	logical, add to an existing plot?
<code>cex</code>	numeric, character expansion factor.
<code>lwd</code>	numeric, line width.
<code>mar</code>	numeric, plot margin specifications.
<code>side</code>	numeric, which sides of plot to draw axes.
<code>xlim</code>	numeric, limits of x axis.
<code>labcex</code>	numeric, character expansion factor for species labels.
<code>pch</code>	numeric, plotting symbol(s) to use for points.
<code>legend</code>	character, text to use for legend.
<code>legend.x</code>	character, placement of legend.
<code>lpch</code>	numeric, plotting symbol(s) to use in legend.
<code>main</code>	character, main title for plot.
<code>lograt.ref</code>	numeric, log10 of reference abundance ratios.
<code>plot.ext</code>	logical, show the location of the extreme value(s)?
<code>z</code>	numeric, matrix of values.
<code>do.sat</code>	logical, identify multiple extreme values.

Details

The purpose of `richness` is to calculate and visualize summary statistics for logarithms of activities of chemical species. For most uses, supply the output of `diagram` as the value for `d`. Alternatively, `d` can be a list of logarithms of activities; the list elements each correspond to a different species and can be vectors, matrices, or higher-dimensional arrays, but they must all have the same dimensions. (This is always the case for `d$logact` if `d` is the output from `diagram`; the dimensionality is determined by the number of variables used in the calculations of `affinity`.) The type of statistic to be calculated is indicated by `target`, as summarized in the following table.

<code>target</code>	description	extremum	additional arguments
<code>sd</code>	standard deviation	min	none
<code>cv</code>	coefficient of variation	min	none
<code>shannon</code>	Shannon diversity index	max	none
<code>qqr</code>	correlation coefficient on q-q plot (normal distribution)	max	none
<code>richness</code>	species richness	max	<code>loga.ref</code>
<code>cvrmsd</code>	coefficient of variation of RMSD	min	<code>loga.ref</code>
<code>spearman</code>	Spearman correlation coefficient	max	<code>loga.ref</code>
<code>pearson</code>	Pearson correlation coefficient	max	<code>loga.ref</code>

`sd`, `cv`, `shannon` and `qqr` all operate on just the sample values. `richness` counts the numbers of species whose logarithms of activities are above `log.min`. `cvrmsd`, `spearman` and `pearson` are comparison statistics where `loga.target` represents the observed values. `ratio` determines the correlation coefficient of a predicted change in loga ratios (`d$logact` vs. `loga.ref`) plotted against observed changes in loga ratios (e.g., from changes in protein expression deduced from microarray experiments; given in `loga.target`)

If `do.plot` is TRUE, `d` is the output from `diagram`, 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.

The value of `'extremum'` in the table shows whether the extreme value that optimizes the system is the minimum (`sd`, `cv`, `cvrmsd`) or the maximum (all the others). On plots the location of the extreme value is indicated (by a dashed vertical line on a 1-D plot or a point marked by an asterisk on a 2-D plot). On 2-D plots the valleys (or ridges) leading to the location of the extremum are plotted. The ridges or valleys are plotted as dashed lines and are colored green for the x values returned by `extremes` and blue for the y values returned by `extremes`.

The location of the extreme value in a matrix or vector `z` is calculated using `where.extreme`. Whether the extreme is the minimum or the maximum value depends on the value of `target`. For matrices, if `do.sat` is TRUE, if the extreme value is repeated, the row and column numbers for all instances are returned. Given a matrix of numeric values in `z`, `extremes` locates the maximum or minimum values in both dimensions. That is, the x values that are returned are the column numbers where the extreme is found for each row, and the y values that are returned are the row numbers where the extreme is found for each column.

If `lograt.ref` is provided, these values are the reference values for logarithm of abundance ratio.

The function name was changed from `diversity` to `revisit` in CHNOSZ-0.9 because there is a function named `diversity` in the **vegan** package. Note that while `diversity` takes a matrix with species on the columns, `revisit` takes a list with species as the elements of the list.

Value

`revisit` returns a list containing at least an element named `'H'` giving the calculated values for the target statistic. This has the same dimensions as a single element of `d` (or `d$logact`, if `d` was the output from `diagram`). For calculations as a function of one or two variables, the output also contains the elements `ix` (location of the extremum in the first direction), `x` (x -value at the extremum), and `extval` (extreme value). For calculations as a function of two variables, the output also contains the elements `iy` (location of the extremum in the second direction) and `y` (y -value at the extremum).

Examples

```
### using grep.file, read.fasta, add.protein
# calculations for Pelagibacter ubique
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 protein
```

```

p <- read.fasta(f, j)
# add these proteins to CHNOSZ's inventory
i <- add.protein(p)
# set up a the chemical system
basis("CHNOS+")
# calculate affinities of formation in logfO2 space
a <- affinity(O2=c(-90,-60), iprotein=i)
# show the equilibrium activities
d <- diagram(a, cex=1.5, logact=0)
# make a title
expr <- as.expression(substitute(x~y~"proteins in"~
  italic("P. ubique"), list(x=length(j), y=w)))
mtitle(c("Equilibrium activities of", expr), cex=1.5)
# show the coefficient of variation
revisit(d, "CV", cex=1.5)
mtitle(c("CV of equilibrium activities of", expr), cex=1.5)
# calculate affinities in logfO2-logaH2O space
a <- affinity(O2=c(-90,-60), H2O=c(-20,0), iprotein=i)
# calculate the equilibrium activities
d <- diagram(a, do.plot=FALSE, mam=FALSE, logact=0)
# show the coefficient of variation
revisit(d, "CV", cex=1.5)
mtitle(c("CV of equilibrium activities of", expr), cex=1.5)

```

species

Species of Interest

Description

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

Usage

```
species(species = NULL, state = NULL, delete = FALSE, quiet = FALSE)
```

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 <code>species</code> (all species if that argument is missing)?
quiet	logical, make less output to the screen?

Details

After defining the `basis` species of your system you can use this function 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 dataframe in `thermo$species` holds the identifying characteristics of the species (including the logarithms of their activities or fugacities) as well as the stoichiometric reaction coefficients for the formation of each of the species from the basis species. The default values for logarithms of activities are -3 for aqueous (`'aq'`) species and 0 for others. If the basis species do not contain all the elements in any of `species`, a message to this effect is generated but the species is nevertheless added with its incomplete stoichiometric definition, even if all zeros.

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 `thermooptstate` 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'`, `'cr3'`, ... (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.

Value

With no arguments, the value of `thermo$species`. Successful addition of species returns the rownumbers of the species added to `thermo$species`; for every species that is not located in the thermodynamic database a warning is generated, and if no species are found the value of `thermo$species` (which is `NULL` at program start-up) is returned. A successful deletion returns the number of species remaining after deletion, or `NULL` if no species remain.

See Also

Use `info` to search the thermodynamic database without adding a species to the system. `basis` is a prerequisite for `species`. `affinity` depends on the definitions made using `species`, but `subcrt`, used to calculate standard molal properties, does not.

Examples

```
## add, modify, delete species
basis("CHNOS")
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
species(delete=TRUE)

## add and delete species
basis(c("CaO","CO2","H2O","SiO2","MgO","O2"))
# a species with Fe (faylite) can be loaded,
# but its initial logarithm of activity is NA to
# indicate that it falls outside the basis definition
species(c("dolomite","quartz","calcite","forsterite","fayalite"))
# changing the elements in the basis definition
# causes species to be deleted
basis(c("CO2","H2O","O2"))
species() # NULL

```

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, do.phases = TRUE, logact = NULL,
       action.unbalanced = "warn", IS = 0)
read.supcrt(file)
write.supcrt(file = "supcrt.dat", obigt = thermo$obigt)

```

Arguments

species	character, name or formula of species, or numeric, rownumber of species in thermo\$obigt.
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).
do.phases	logical, replace Gibbs energies of phases of minerals and of other species beyond their upper temperature limits of stability or extrapolation with 999999?
logact	numeric, logarithms of activities of species in reaction.
file	character, name of SUPCRT92 data file.
obigt	dataframe, thermodynamic data.
convert	logical, are input and output units of T and P those of the user (TRUE) (see nuts), 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 .

Details

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 H₂O 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 H₂O ([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 or P to perform the computation of a T×P or P×T grid. The `property`s to be calculated can be one or more of those shown below:

rho	Density of water	g cm ⁻³
logK	Logarithm of equilibrium constant	dimensionless
G	Gibbs energy	(cal J) mol ⁻¹
H	Enthalpy	(cal J) mol ⁻¹
S	Entropy	(cal J) K ⁻¹ mol ⁻¹
V	Volume	cm ³ mol ⁻¹
Cp	Heat capacity	(cal J) K ⁻¹ mol ⁻¹
E	Exapansibility	cm ³ K ⁻¹
kT	Isothermal compressibility	cm ³ bar ⁻¹

(Note that E and kT can only be calculated for aqueous species and only if the option (`thermooptwater`) 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 global units definition (see [nuts](#)) the values of G, H, S and Cp are returned in calories or Joule, but only if `convert` is TRUE. Likewise, the input values of T and P should be

in units specified through `nuts`, 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 `dPdTr`), 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`), the Gibbs energies of these minerals are assigned values of 999999 at conditions where another of the phases is stable (only if `do.phases` is `TRUE`). (NOTE: if you set `do.phases` to `FALSE` 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 generally not advised, since the computed standard molal properties of a phase species lose their physical meaning if computed beyond the transition temperatures of the corresponding phase.)

The chemical affinities of reactions are calculated if `logact` is provided. This argument 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 global basis definition. Columns appended to the output are `logQ` for the activity product of the reaction, and `A` for the chemical affinity (the latter in units corresponding to those of Gibbs energy). 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 C_p) 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 H_2O 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.

`read.subcrt` and `write.subcrt` are used to read and write thermodynamic data files in the format of SUPCRT92. `read.subcrt` parses the thermodynamic data contained in a SUPCRT92-format file into a dataframe that is compatible with the format used in CHNOSZ (see `thermo$obigt`). `write.subcrt` creates a SUPCRT92 file using the dataframe given in the `obigt` argument (if missing the entire species database is processed). An edited version of the 'slop98.dat' data file (Shock et al., 1998) that is compatible with `read.subcrt` is available at <http://chnosz.net/download/> (small formatting quirks in the original version cause glitches with this function).

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.

`read.subcrt` returns a dataframe of properties and parameters of species with the same structure as `thermo$obigt`.

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 and for charged aqueous species over the temperature range 0 to 300 °C. At higher temperatures, there are significant divergences for charged species. For example, there is less than a 2 percent difference in the value of ΔG° for $K^+(aq)$ at °C and 5000 bar, but this deviation increases with decreasing pressure at the same temperature. Further testing is necessary in CHNOSZ in connection with the g - and f -functions (Shock et al., 1992) for the partial derivatives of the omega parameter which are incorporated into the `hkf` function. (Note in particular the NaCl dissociation example under `water`, the results of which are noticeably different from the calculations of Shock et al., 1992.)

References

- Alberty, R. A. (2003) *Thermodynamics of Biochemical Reactions*, John Wiley & Sons, Hoboken, New Jersey, 397 p. <http://www.worldcat.org/oclc/51242181>
- Johnson, J. W., Oelkers, E. H. and Helgeson, H. C. (1992) SUPCRT92: A software package for calculating the standard molal thermodynamic properties of minerals, gases, aqueous species, and reactions from 1 to 5000 bar and 0 to 1000°C. *Comp. Geosci.* **18**, 899–947. [http://dx.doi.org/10.1016/0098-3004\(92\)90029-Q](http://dx.doi.org/10.1016/0098-3004(92)90029-Q)

LaRowe, D. E. and Helgeson, H. C. (2007) Quantifying the energetics of metabolic reactions in diverse biogeochemical systems: electron flow and ATP synthesis. *Geobiology* **5**, 153–168. <http://dx.doi.org/10.1111/j.1472-4669.2007.00099.x>

Schulte, M. D. and Shock, E. L. (1995) Thermodynamics of Strecker synthesis in hydrothermal systems. *Orig. Life Evol. Biosph.* **25**, 161–173. <http://dx.doi.org/10.1007/BF01581580>

Shock, E. L., Oelkers, E. H., Johnson, J. W., Sverjensky, D. A. and Helgeson, H. C. (1992) Calculation of the thermodynamic properties of aqueous species at high pressures and temperatures: Effective electrostatic radii, dissociation constants and standard partial molal properties to 1000 °C and 5 kbar. *J. Chem. Soc. Faraday Trans.* **88**, 803–826. <http://dx.doi.org/10.1039/FT9928800803>

Shock, E. L. et al. (1998) *SLOP98.dat* (computer data file). http://geopig.asu.edu/supcrt92_data/slop98.dat, accessed on 2005-11-05.

See Also

[info](#) for querying the thermodynamic database; [makeup](#) for parsing formulas to check reaction balance; [water](#), [hkf](#) and [cgl](#) for equations of state calculations.

Examples

```
## 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"))

## input/output in different units
nuts(c("K","J"))
subcrt("water")
subcrt("water",T=298.15)
nuts(c("cal","C")) # back to defaults

## 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"))
# this warns about an unbalanced reaction
subcrt(c("glucose","ethanol"),c(-1,3))

## 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 NAs and warnings
# 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)
nuts(c("J", "K")) # set the units
# we set pressure here otherwise subcrt goes for Psat (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)
plot(s$out[[1]]$T, s$out[[1]]$Cp, type="l",
     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
nuts(c("C", "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
# T <- seq(0,1000,100)
# we use this one to avoid warnings at 0 deg C, 5000 bar
T <- c(2,seq(100,1000,100))
subcrt(c("andradite","carbon dioxide","H2S","Cu+","quartz","calcite",
"chalcopyrite","H+","H2O"),coeff=c(-1,-3,-4,-2,3,3,2,2,3),
state=c("cr","g","aq","aq","cr","cr","cr","aq","liq"),
P=P,T=T,grid="P")
# 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

## 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$out$T,rxn1$out$G/1000,type="l",ylim=c(-40,-10),
xlab=axis.label("T"),ylab=axis.label("DG0r","k"))
lines(rxn1$out$T,rxn2$out$G/1000)
# write the reactions on the plot
text(150,-14,describe(rxn1$reaction,
use.name=c(TRUE,FALSE,FALSE,TRUE,FALSE)))
text(200,-35,describe(rxn2$reaction,
use.name=c(TRUE,FALSE,FALSE,TRUE)))
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 ")
T <- seq(0,120,10)
# 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,xlim=range(T),ylim=c(1.4,5.4),
xlab=axis.label("T"),ylab=axis.label("A",opt="k"),type="l")
text("NAD(ox)-/NAD(red)-2 = 1",x=median(T),y=median(A))
# 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=median(T),y=median(A))
# 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=s$out$A/24/1000)
text("NAD(ox)-/NAD(red)-2 = 0.1",x=median(T),y=median(A))
# this command prints the reaction on the plot
text(40,4.5,c2s(s2c(describe(s$reaction,
  use.name=c(TRUE,TRUE,FALSE,FALSE,FALSE,FALSE)),
  sep="="),sep="\n"))
# label the plot
title(main=paste("Chemical affinity of NAD reduction",
  "after LaRowe and Helgeson, 2007",sep="\n"),
  sub=describe(thermo$basis,T=NULL))

### non-ideality calculations -- activity coefficients of
### aqueous species as a function of charge, temperature,
### and ionic strength -- after Alberty, 2003
## p. 16 Table 1.3 apparent pKa of acetic acid with
## changing ionic strength
subcrt(c("acetic acid","acetate","H+"),c(-1,1,1),
  IS=c(0,0.1,0.25),T=25,property="logK")
# note that these *apparent* values of G and logK approach
# their *standard* counterparts as IS goes to zero.
## p. 95: basis and elemental stoichiometries of species
## (a digression here from the nonideality calculations)
# note coefficient of O2 and NH3 will be zero for these species
basis(c("ATP-4","H+","H2O","HPO4-2","O2","NH3"))
# cf Eq. 5.1-33: (basis composition)
species(c("ATP-4","H+","H2O","HPO4-2","ADP-3","HATP-3","HADP-2","H2PO4-"))
lb <- length(basis())
# cf Eq. 5.1-32: (elemental composition)
as.matrix(species()[,1:lb])
## p. 273-275: activity coefficient (gamma)
## as a function of ionic strength and temperature
## (as of 20080304, these do look quantitatively different
## from the plots in Alberty's book.)
iplofun <- function(T,col,add=TRUE) {
  IS <- seq(0,0.25,0.0025)
  s <- subcrt(c("H2PO4-","HADP-2","HATP-3","ATP-4"),IS=IS,grid="IS",T=T)
  if(!add) thermo.plot.new(xlim=range(IS),ylim=c(0,1),
    xlab=axis.label("IS"),ylab="gamma")
  for(i in 1:4) lines(IS,10^s$out[[i]]$loggam,col=col)
}
iplofun(0,"blue",add=FALSE)
iplofun(25,"black")
iplofun(40,"red")
title(main=paste("activity coefficient (gamma) of -1,-2,-3,-4",
  "charged species at 0, 25, 40 deg C, after Alberty, 2003",
  sep="\n"),cex.main=0.95)

```

Description

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

Usage

```
getnodes(taxdir)
getrank(id, taxdir, nodes=NULL)
parent(id, taxdir, rank=NULL, nodes=NULL)
allparents(id, taxdir, nodes=NULL)
getnames(taxdir)
sciname(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 <code>getnodes</code> (optional).
rank	character, name of the taxonomic rank of interest.
names	dataframe, output from <code>getnames</code> (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. `sciname` 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 `sciname` 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

```
## get information about Homo sapiens from the
## packaged taxonomy files
taxdir <- system.file("extdata/taxonomy", package="CHNOSZ")
# H. sapiens' taxonomic id
id1 <- 9606
# that is a species
getrank(id1, taxdir)
# the next step up the taxonomy
```

```

id2 <- parent(id1,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 `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 `water` so that subsequent calculations at the same temperature-pressure conditions can be accelerated.

The `thermo` object is a `list` composed of `data.frames` or lists each representing a class of data. The object is created in the global environment upon loading the package (by `.onAttach`). At any time, the user can restore the data object to its initial state by calling `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 `add.obigt` is available to update the thermodynamic database in use in a running session. Using this function is the recommended way for users to incorporate custom additions or changes to thermodynamic data. For example, one can run `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 `OBIGT.csv` file provided with CHNOSZ. Although changes made using `add.obigt` are lost when the current R session is closed, the data can always be restored the next time as long as the user has the `mydata.csv` (or other) file available.

The first example below shows how to find the installation locations of `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

```
data(thermo)
```

Format

The items in the `thermo` object are documented below.

- `thermo$opt` List of operational parameters

<code>Tr</code>	numeric	Reference temperature (K)
<code>Pr</code>	numeric	Reference pressure (bar)
<code>Theta</code>	numeric	Θ in the revised HKF equations of state (K)
<code>Psi</code>	numeric	Ψ in the revised HKF equations of state (bar)
<code>cutoff</code>	numeric	Cutoff below which values are taken to be zero (see makeup)
<code>E.units</code>	character	The user's units of energy ('cal' (default) or 'J')
<code>T.units</code>	character	The user's units of temperature ('C' (default) or 'K')
<code>P.units</code>	character	The user's units of pressure ('bar' (default) or 'MPa')
<code>state</code>	character	The default physical state for searching species ('aq' by default)
<code>ionize</code>	logical	Should affinity perform ionization calculations for proteins?
<code>water</code>	character	Computational option for properties of water ('SUPCRT' (default) or 'IAPWS')
<code>online</code>	logical	Allow online searches of protein composition? Default (NA) is to ask the user.

- `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 $S(\text{H}_2, \text{g}) + 2S(Z) = 2S(\text{H}^+)$, 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 dataframe using `element` 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.

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

- `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`. Note the following additional modifications:

- Use corrected values of a_2 and a_4 for $[-\text{CH}_2\text{NH}_2]$ (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 entropy of the ‘cr2’ phase species at T_{tr} , and taking account of the enthalpy of transition (ΔH° , 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 ΔH° 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`:

name	character	Species name
abbrv	character	Species abbreviation
formula	character	Species formula
state	character	Physical state
ref1	character	Primary source
ref2	character	Secondary source
date	character	Date of data entry
G	numeric	Standard molal Gibbs energy of formation from the elements (cal mol ⁻¹)
H	numeric	Standard molal enthalpy of formation from the elements (cal mol ⁻¹)
S	numeric	Standard molal entropy (cal mol ⁻¹ K ⁻¹)
Cp	numeric	Standard molal isobaric heat capacity (cal mol ⁻¹ K ⁻¹)
V	numeric	Standard molal volume (cm ³ mol ⁻¹)

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 [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 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):

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

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

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 ^{-λ-1} mol ⁻¹)
lambda	numeric	λ (exponent on the f term)
T	numeric	Temperature of phase transition or upper temperature limit of validity of extrapolation (K)

- `thermo$source` Dataframe of references to sources of thermodynamic data.

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

- `thermo$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 [buffer](#) do not take into account phase transitions of minerals, so individual phase species of such minerals must be specified in the buffers.

name	character	Name of buffer
species	character	Name of species
state	character	Physical state of species
logact	numeric	Logarithm of activity (fugacity for gases)

- `thermo$protein` Dataframe 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, ‘MOD5.M’ and MOD5.N proteins of ‘YEAST’ denote the mitochondrial and nuclear isoforms of this protein. Note that `get.protein` and `get.expr` can both be used to add proteins to this table using some of the `extdata` files provided with the package. `add.protein` is available to add proteins from local files provided by the user.

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

- `thermo$stress` Dataframe 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 ‘SGD’) or gene names (for ‘ECO’). The column names and first two rows give the following information:

<code>colname</code>	character	Name of the experiment
<code>organism</code>	character	Name of the organism (‘SGD’ or ‘ECO’)
<code>ref</code>	character	Reference key for source of the data

- `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).

<code>...</code>	numeric	One or more columns of stoichiometric coefficients of elements in the basis species
<code>ispecies</code>	numeric	Rownumber of basis species in <code>thermo\$obigt</code>
<code>logact</code>	numeric	Logarithm of activity or fugacity of basis species
<code>state</code>	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`).

<code>...</code>	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

- Amend, J. P. and Helgeson, H. C. (1997b) Calculation of the standard molal thermodynamic properties of aqueous biomolecules at elevated temperatures and pressures. Part 1. L- α -amino acids. *J. Chem. Soc., Faraday Trans.* **93**, 1927–1941. <http://dx.doi.org/10.1039/a608126f>
- Cox, J. D., Wagman, D. D. and Medvedev, V. A., eds. (1989) *CODATA Key Values for Thermodynamics*. Hemisphere Publishing Corporation, New York, 271 p. <http://www.worldcat.org/oclc/18559968>
- Dick, J. M., LaRowe, D. E. and Helgeson, H. C. (2006) Temperature, pressure, and electrochemical constraints on protein speciation: Group additivity calculation of the standard molal thermodynamic properties of ionized unfolded proteins. *Biogeosciences* **3**, 3110–336. <http://www.biogeosciences.net/3/311/2006/bg-3-311-2006.html>
- Johnson, J. W., Oelkers, E. H. and Helgeson, H. C. (1992) SUPCRT92: A software package for calculating the standard molal thermodynamic properties of minerals, gases, aqueous species, and reactions from 1 to 5000 bar and 0 to 1000°C. *Comp. Geosci.* **18**, 899–947. [http://dx.doi.org/10.1016/0098-3004\(92\)90029-Q](http://dx.doi.org/10.1016/0098-3004(92)90029-Q)
- Shock, E. L. and Koretsky, C. M. 1995 Metal-organic complexes in geochemical processes: Estimation of standard partial molal thermodynamic properties of aqueous complexes between metal cations and monovalent organic acid ligands at high pressures and temperatures. *Geochim. Cosmochim. Acta* **59**, 1497–1532. [http://dx.doi.org/10.1016/0016-7037\(95\)00058-8](http://dx.doi.org/10.1016/0016-7037(95)00058-8)
- Shock, E. L. et al. 1998 *SLOP98.dat* (computer data file). http://geopig.asu.edu/supcrt92_data/slop98.dat, accessed on 2005-11-05. Current location: <http://geopig.asu.edu/sites/default/files/slop98.dat>.
- Wagman, D. D., Evans, W. H., Parker, V. B., Schumm, R. H., Halow, I., Bailey, S. M., Churney, K. L. and Nuttall, R. L. (1982) The NBS tables of chemical thermodynamic properties. Selected values for inorganic and C₁ and C₂ organic substances in SI units. *J. Phys. Chem. Ref. Data* **11** (supp. 2), 1–392. <http://www.nist.gov/srd/PDFfiles/jpcrdS2Vol11.pdf>

See Also

`add.protein` and `add.obigt` for adding amino acid compositions of proteins and thermodynamic data from local .csv files.

Examples

```
## where are OBIQT.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$state)
# formulas of ten random species
n <- nrow(thermo$obigt)
thermo$obigt$formula[runif(10)*n]

## make a table of duplicated species
name <- thermo$obigt$name
state <- thermo$obigt$state
ref <- thermo$obigt$ref1
species <- paste(name, state)
dups <- species[which(duplicated(species))]
id <- numeric()
for(i in 1:length(dups)) id <- c(id, which(species %in% dups[i]))
data.frame(name=name[id], state=state[id], ref=ref[id])
# give an error if there are any duplicates
stopifnot(length(id) > 0)
```

transfer

Mass Transfer Calculations

Description

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

Usage

```
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", do.plot = FALSE)
apc(which = "open", basis = "CO2", do.plot = 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.

<code>plot</code>	numeric, which basis species to use as plotting and coupling variables.
<code>ibalance</code>	numeric, which basis species is the primary conservant.
<code>fmode</code>	character, formation mode.
<code>buffers</code>	list, basis species to be buffered during the simulation.
<code>alphamax</code>	numeric, maximum value of the destruction exponent.
<code>alphastart</code>	numeric, initial value of the destruction exponent.
<code>T</code>	numeric, temperature.
<code>P</code>	numeric, pressure.
<code>do.title</code>	logical, plot a title?
<code>beta</code>	numeric, alpha + beta = buffer transfer exponent.
<code>t</code>	list, the output of <code>ttransfer</code> .
<code>ylim</code>	numeric, y-axis limits.
<code>ylimbasis</code>	numeric, y-axis limits for the logarithms of activities of basis species.
<code>logprogress</code>	logical, put reaction progress on a logarithmic scale?
<code>which</code>	character, type of system to simulate.
<code>do.plot</code>	logical, summarize the results using <code>draw.transfer</code> ?
<code>basis</code>	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 `species`.

At each step, a small amount (10^α) 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 `ibalance`) is not changed. The changes in the activities of the other basis species are calculated, and the process is iterated until `nsteps` is reached or the value of α is driven to a very low value (`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 `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 `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 `iplot`.

The initial value of alpha is given by `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 `devmax`. Therefore, throughout the simulation the value of α dynamically adjusts based on the `devmax` set by the user.

`buffers` is a list with elements `basis` indicating the basis species to be buffered and `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 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.

`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 `logprogress` is set to `TRUE`. The y-limits of the plots can be set using `ylim` and `ylimbasis`.

`feldspar` and `apc` encode examples for feldspar weathering and reactions among proteins in the anaphase-promoting complex of yeast. Equivalent code for two selected examples is shown below.

Value

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

References

Helgeson, H. C., Garrels, R. M. and Mackenzie, F. T. (1969) Evaluation of irreversible reactions in geochemical processes involving minerals and aqueous solutions. II. Applications. *Geochim. Cosmochim. Acta* **33**, 455–481. [http://dx.doi.org/10.1016/0016-7037\(69\)90127-6](http://dx.doi.org/10.1016/0016-7037(69)90127-6)

Steinmann, P., Lichtner, P. C. and Shotyk, W. (1994) Reaction path approach to mineral weathering reactions. *Clay Clay Min.* **42**, 197–206. <http://ccm.geoscienceworld.org/cgi/content/abstract/42/2/197>

Examples

```
## 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", "pyrophyllite", "kaolinite", "gibbsite"))
a <- affinity(H4SiO4=c(-6, -2), "K+"=c(-3, 8))
diagram(a)
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(1))
# note, the same calculation can be performed using
# feldspar("closed")

## react APC2 from Saccharomyces cerevisiae
## to other proteins in the anaphase-promoting complex
```

```

basis(c("CO2", "H2O", "NH3", "H2", "H2S"), c(-10, 0, -4, -10, -7))
species(c("APC1", "APC2", "APC5", "CDC16", "APC10", "SWM1"), "YEAST")
a <- affinity(CO2=c(-10, 0), H2=c(-10, 0))
diagram(a, as.residue=TRUE)
species(1:nrow(species()), -999)
species("APC2_YEAST", 0)
tr <- transfer(510, ibalance="PBB", plot=c(1, 4), dmode="coupled", devmax=0.15)
# this calculation is also available with
# apc("closed")

```

util.args

Functions for Processing Argument Lists

Description

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

Usage

```

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 `propertys` 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

```
## argument processing
eos.args("hkf",c("g","H","S","cP","V","kT","e"))
## produces an error because "Q" is not allowed in water.SUPCRT92
## Not run:
eos.args("hkf",c("G","H","S","Cp","V","kT","E","Q"))
## End(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"))
```

util.array

*Functions to Work with Multidimensional Arrays***Description**

These functions can be used to turn a list into an array and extract or replace values or take the sum along a certain dimension of an array.

Usage

```
list2array(l)
slice(arr, d = NULL, i = 1, value = NULL)
dimSums(arr, d = 1, i = NULL)
slice.affinity(affinity, d = 1, i = 1)
```

Arguments

<code>l</code>	a list.
<code>arr</code>	an array.
<code>d</code>	numeric, what dimension to use.
<code>i</code>	numeric, what slice to use.
<code>value</code>	values to assign to the portion of an array specified by <code>d</code> and <code>i</code> .
<code>affinity</code>	list, output from <code>affinity</code> 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.

`slice` 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

```
# start with a matrix
x <- matrix(1:12,ncol=3)
# pay attention to the following when
# writing examples that test for identity!
identical(1*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( slice(c,3), a ))
# the second slice of the third dimension == b
stopifnot(identical( slice(c,3,2), b ))
# 'slice' works just like the bracket operator
c11 <- slice(c,1)
c12 <- slice(c,1,2)
c21 <- slice(c,2,1)
c212 <- slice(c,2,1:2)
stopifnot(identical( c11, c[1,,] ))
stopifnot(identical( c12, c[2,,] ))
stopifnot(identical( c21, c[,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$values,a3$values))
```

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

```
read.blast(file, similarity = 30, evalue = 1e-5, max.hits = 1,
  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

file	character, name of BLAST tabular output file.
similarity	numeric, hits above this similarity score are kept.
evalue	character, hits below this E value are kept.
max.hits	numeric, up to this many hits are kept for each query sequence.
quiet	logical, produce fewer messages?
blast	dataframe, BLAST table.
gi.taxid	list, first component is sequence identifiers (gi numbers), second is taxon ids (taxids).
taxid.names	dataframe, with at least columns 'taxid' (taxon id), 'phylum' (name of phylum), 'species' (name of species).
min.taxon	numeric, this taxon is kept if it makes up at least this fraction of total.
min.query	numeric, query sequence is counted if a single phylum makes up this fraction of its hits.
min.phylum	numeric, this phylum is kept if it makes up at least this fraction of total.
take.first	logical, keep only first hit after all other filtering steps?
outfile	character, name of output file.
def	character, FASTA define(s)

Details

`read.blast` reads a BLAST tabular output file, keeping only those hits with greater than or equal to the `similarity` and less than or equal to the `evalue` (expectation value) specified in the arguments. Furthermore, for each query sequence, only the top number of hits specified by `max.hits` are kept. Note that BLAST (Altschul et al., 1997) tabular output files can be generated using the `-m 8` switch to the `'blastall'` command.

`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 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 define.

Value

`read.blast` returns a dataframe with as many columns (12) as the BLAST file. `id.blast` returns a dataframe with columns `query`, `subject` (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

Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J. H., Zhang, Z., Miller, W. and Lipman, D. J. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* **25**, 3389–3402. <http://dx.doi.org/doi:10.1093/nar/25.17.3389>

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

```

## 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)
# read the file that connects taxids with the sequence identifier
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_refseq47.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")
    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(row.names(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("Most Abundant Phyla in a Portion",
  "of the Bison Pool Metagenome", sep="\n"))

```

Description

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

Usage

```
c2s(x, sep = " ")
s2c(x, sep = NULL, keep.sep = TRUE)
can.be.numeric(x)
```

Arguments

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

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 `s2c` is determined by `n`; the default value of NULL indicates an unrestricted length.

`can.be.numeric` returns a value of TRUE or FALSE for each element of `x`.

Value

`s2c`, `c2s` and `axis.label` return character values. `can.be.numeric` returns logical.

Examples

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

```
add.obigt(file = system.file("extdata/thermo/OBIGT-2.csv",
  package = "CHNOSZ"), force = FALSE, E.units = "cal")
mod.obigt(species, ..., missingvalues = NA)
add.protein(file="protein.csv")
change(name, ...)
browse.refs(key=NULL)
checkEOS(eos, state, prop, ret.diff = FALSE)
checkGHS(ghs, ret.diff = FALSE)
check.obigt()
obigt2eos(obigt, state, fixGHS = FALSE)
```

Arguments

file	character, path to a file.
force	logical, force replacement of already existing species?
E.units	character, units of energy, 'cal' or 'J'.
species	character, formulas or names of species to modify or add to the thermodynamic database.
...	character or numeric, properties of species to modify in the thermodynamic database.
missingvalues	numeric, values to assign to undefined properties.
name	character or numeric, name (or numeric index) of species or name of buffer to be modified.
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 thermo\$obigt.
state	character, physical state of species ('aq' or other).
prop	character, property of interest ('Cp' or 'V').
ret.diff	logical, return the difference between calculated and tabulated values?
ghs	dataframe, containing G, H and S, in the format of thermo\$obigt.
obigt	dataframe, in the format of thermo\$obigt.
fixGHS	logical, calculate one of missing G, H, or S?

Details

`add.obigt` and `add.protein` read data from the specified file and add it to either `thermo$obigt` or `thermo$protein`, as appropriate. Set `force` to `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). `force`, if not specified, reverts to `TRUE` if the file is left at its default. Given the default setting of `E.units`, the function does not perform any unit conversions. If `E.units` is set to 'J', then the thermodynamic parameters are converted from units of Joules to calories, as used in the CHNOSZ database.

`mod.obigt` changes one or more of the properties of one or more species or adds species to the thermodynamic database. These changes are lost if you reload the database by calling `data(thermo)` or if you quit the R session without saving it. To modify the properties of species, give the names in the `species` argument and supply other arguments: if one of these arguments is `state`, species in those states will be updated. Additional arguments refer to the name of the property(s) to be updated and correspond to the column names of `thermo$obigt` (the names of the properties are matched to any part of compound column names, such as 'z.T'). The values provided should be in the units specified in the documentation for the `thermo` data object. To add species, supply the new names in `species` and provide an argument named `formula` with the corresponding chemical formulas. Additional arguments refer to any of the properties you wish to specify. Properties that are not specified are assigned the value of `missingvalues` which is `NA` by default (however if `state` is missing it is set to the value of `thermooptstate`). The values returned (`invisible-y`) by `mod.obigt` are the rownumbers of the affected species.

`change` is a wrapper function to `mod.obigt` and `mod.buffer`. The name provided in the argument refers to the name or numeric index of the species to update or add using `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 `mod.buffer`. The arguments in `...` are sent without `change` to the subordinate function.

`browse.refs` with default arguments uses `browseURL` to display the sources of thermodynamic data in `thermo$refs`, with the URLs in that table showing as hyperlinks in the browser. Otherwise, if `key` is character, the URLs of those reference keys are opened in the browser. If `key` is numeric, the values refer to the species in those rows of `thermo$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 degrees 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 tolerances 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 degrees C and 1 bar. A message is printed and the calculated difference is returned if it exceeds a tolerance of 500 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).

Side Effects

These functions can modify the `obigt`, `protein` and `buffers` dataframes in the global `thermo` list.

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

```
## modify/add species
info(ia <- info("alanine", "cr"))
mod.obigt("alanine", state="cr", G=0, H=0, S=0)
# now the values of G, H, and S are inconsistent
# with the elemental composition of alanine
info(ia)
# add a species
mod.obigt("myspecies", formula="CHNOSZ", G=0, H=0)
info(im <- info("myspecies"))

## 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 <- subcrt(c("O2", "O2"), c("gas", "aq"), c(-1, 1))
```

```

    browse.refs(s)
## End(Not run)

```

util.fasta

Functions for Accessing FASTA Files

Description

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

Usage

```

is.fasta(file)
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, pnff = FALSE)
splitline(line, length)
trimfas(file, start, stop)

```

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 <code>grep</code> 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.
pnff	logical, get the protein name from the filename?
line	character, a line to be split into multiple lines.
length	numeric, the maximum length of any line.
start	numeric, starting position to extract from sequences.
stop	numeric, last position to extract from sequences.

Details

`is.fasta` checks if a file is in FASTA format. A very simple test is performed: if either of the first two lines of the file starts with '>', then the function returns TRUE, otherwise it returns FALSE.

`grep.file` is used to search for entries in a FASTA file. It returns the line numbers of the matching FASTA headers. It takes a search term in `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 line for each sequence 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. The line numbers for the headers of the desired sequences are passed to the function in `i` (they can be generated using `grep.file`). The function returns various formats depending on the value of `ret`; the default 'count' returns a dataframe of amino acid counts (the dataframe 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. `lines`, if it is given, bypasses the reading of the file and also overrides the use of the OS's tools. If the line numbers of the header lines were previously determined, they can be supplied in `ihead`.

`splitline` takes a single character object (the `line`) and splits it into multiple lines of the given length (the last line can be shorter than this). It returns a character object that contains the lines. This function is utilized by `trimfas`, which extracts the specified positions from a (usually) aligned FASTA file. The length of the lines used by `trimfas` is equal to the length of the first sequence line in the given `file`.

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`.

Side Effects

None

See Also

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

```
# basic use of splitline
splitline("abcdefghijklmnopqrstuvwxy",10)

# get the first ten positions of each of the sequences
# in a FASTA file
f <- system.file("extdata/fasta/HTCC1062.faa.xz",package="CHNOSZ")
fnew <- trimfas(f,1,10)
```

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 number of carbon. Also, create a matrix having the chemical formulas of amino acid residues in proteins and calculate the chemical formulas of proteins from their amino acid composition.

Usage

```
GHS(species = NULL, DG = NA, DH = NA, S = NA, T = thermo$opt$Tr)
element(compound, property = c("mass", "entropy"))
expand.formula(elements, makeup)
ZC(x)
residue.formula()
protein.formula(proteins, as.residue = FALSE)
```

Arguments

species	character, formula of a compound from which to calculate entropies of the elements.
DG	numeric, standard molal Gibbs energy of formation.
DH	numeric, standard molal enthalpy of formation.
S	numeric, standard molal molal entropy.
T	numeric, temperature in Kelvin.
compound	character, name of element(s) or compound(s).
property	character, name(s) of thermodynamic properties.
elements	character, name(s) of elements.
makeup	dataframe, elemental composition of a compound returned by makeup .
x	character, object representing chemical formula.
proteins	dataframe, amino acid composition of one or more proteins in the same format as thermo\$protein .
as.residue	logical, return the per-residue formula of the protein(s)?

Details

GHS computes one of the standard molal Gibbs energy or enthalpy of formation from the elements (DG, DH) or entropy (S) at 298.15 K and 1 bar from values of the other two. If the `species` argument is present, it is used to calculate the entropies of the elements (S_e) using `element`, otherwise S_e is set to zero. The equation in effect can be written as $\Delta G^\circ = \Delta H^\circ - T\Delta S^\circ$, where $\Delta S^\circ = S - S_e$ and T denotes the reference temperature of 298.15 K. If two of DG, DH, and S are provided, the value of the third is returned. If three are provided, the value of DG in the arguments is ignored and the calculated value of DG is returned. If none of DG, DH or S are provided, the value of S_e is returned. If only one of the values is provided, an error results. Units of cal mol^{-1} (DG, DH) and $\text{cal K}^{-1} \text{mol}^{-1}$ (S) are assumed. If T is provided, it is used instead of the reference temperature.

`element` returns a dataframe of the mass and entropy of one or more elements or formulas given in `compound`. The property can be 'mass' and/or 'entropy'.

`expand.formula` converts a 1-column dataframe representing the elemental composition of a compound (see `makeup`) to a numeric vector, each value of which is the coefficient of the elements given in the argument. If any of these is not present in the `makeup` dataframe, its coefficient is set to zero. A non-zero coefficient of an element in the `makeup` dataframe does not appear in the output if that element is not one of `elements`.

`ZC` returns the nominal carbon oxidation state for the chemical formula represented by `x`. (For discussion of nominal carbon oxidation state, see Hendrickson et al., 1970; Buvet, 1983.) If carbon is not present in the formula the result is NaN.

`protein.formula` exists to quickly compute the chemical formulas of many proteins. The `proteins` argument contains the amino acid compositions of the proteins in the same format as the `thermo$protein` dataframe. `residue.formula` is called to calculate the chemical formulas of each of the 20 common amino acid residues (and the terminal H- and -OH). The amino acid compositions of the proteins and the output of `residue.formula` are multiplied using matrix multiplication to generate the result.

Value

GHS and ZC return numeric values. `expand.formula` returns a numeric vector.

References

Buvet, R. (1983) General criteria for the fulfillment of redox reactions, in *Bioelectrochemistry I: Biological Redox Reactions*, Milazzo, G. and Blank, M., eds., Plenum Press, New York, p. 15–50. <http://www.worldcat.org/oclc/9282370>

Hendrickson, J. B., Cram, D. J., and Hammond, G. S. (1970) *Organic Chemistry*, 3rd ed., McGraw-Hill, New York, 1279 p. <http://www.worldcat.org/oclc/78308>

See Also

`makeup` can be used to count the elements in formulas and display formulas in various formats.

Examples

```

## converting among Gibbs, enthalpy, entropy
GHS("H") # entropy of H (element)
# calculate enthalpy of formation of arsenopyrite
GHS("FeAsS",DG=-33843,S=68.5)
# return the value of DG calculated from DH and S
# cf. -56687.71 from subcrt("water")
GHS("H2O",DH=-68316.76,S=16.7123)

## mass and entropy of compounds of elements
element("CH4")
element(c("CH4","H2O"),"mass")
element("Z") # charge
# same mass, opposite entropy as charge
element("Z-1") # i.e., electron

## count selected elements in a formula
m <- makeup("H2O")
expand.formula(c("H","O"),m)
expand.formula(c("C","H","S"),m)

## calculate the average chemical formula of all of
## the proteins in CHNOSZ' database
## this is much faster than a for-loop
pf <- protein.formula(thermo$protein)
colSums(pf)/nrow(pf)

## nominal carbon oxidation states
ZC("CO2") # 4
ZC("CH4") # -4
ZC("CHNOSZ") # 7
si <- info(info("LYSC_CHICK"))
ZC(si$formula) # 0.01631

## plot ZC of reference protein sequence
## for different organisms
file <- system.file("extdata/refseq/protein_refseq.csv.xz",package="CHNOSZ")
ip <- add.protein(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$protein[ip,])
zc <- ZC(pf)
# the organism names we search for
# "" matches all organisms
terms <- c("Streptomyces","Pseudomonas","Salmonella",
  "Escherichia","Vibrio","Bacteroides","Lactobacillus",
  "Staphylococcus","Streptococcus","Methano","Bacillus","Thermo","")
tps <- thermo$protein$ref[ip]
plot(0,0,xlim=c(1,13),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[13] <- paste("all organisms")
  axis(1, 1:13, terms, las=2)
  title(main=paste("Average Oxidation State of Carbon:",
    "Total Protein per taxID in NCBI RefSeq", sep="\n"))

```

util.list

*Functions to Work with Lists***Description**

Combine lists or perform arithmetic operations on elements of lists.

Usage

```

lsub(x, y)
lsum(x, y)
pprod(x, y)
psum(x)
which.pmax(elts, na.rm = FALSE, pmin = FALSE)
mylapply(X, FUN, ...)

```

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?
X	vector, argument for <code>lapply</code> or <code>mclapply</code>
FUN	function, argument for <code>lapply</code> or <code>mclapply</code>
...	additional arguments to be passed to <code>lapply</code> or <code>mclapply</code>

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`. `psum` sums all elements of the list `x`.

`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.

`mylapply` passes the given arguments to `lapply`, or to `mclapply` if the **multicore** package is loaded and the length of `X` is greater than 20. `mylapply` is used in `affinity` (in calculations for proteins activated by the `iprotein` argument), `equil.boltz` (in parallel operations on list elements), and `aminoacids` and `protein.length` (in counting amino acids in sequences and determining lengths of proteins).

Value

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

util.misc

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, execute multicore calculations, scale logarithms of activity to a desired total activity, get name of calling function.

Usage

```
protein.length(protein)
MP90.cp(T, protein)
dPdTtr(x)
Ttr(x, P = 1, dPdT = NULL)
describe(x = NULL, T = NULL, P = NULL, use.name = FALSE,
  as.reaction = NULL, digits = 1)
basis.comp(basis)
nonideal(species, proptable, IS, T)
which.balance(species)
unitize(logact = NULL, length = NULL, logact.tot = 0)
caller.name(n)
```

Arguments

<code>protein</code>	character, name of protein species; numeric, species index of protein.
<code>T</code>	numeric, temperature (K) (<code>lines.water</code> , <code>describe</code> , <code>MP90.cp</code> , <code>nonideal</code>).
<code>x</code>	numeric index of a mineral phase (<code>dPdTtr</code> , <code>Ttr</code>), or dataframe, definition of basis species or reaction (<code>describe</code>).
<code>P</code>	numeric, pressure (bar).
<code>dPdT</code>	numeric, values of (dP/dT) of phase transitions (<code>Ttr</code>).
<code>use.name</code>	logical, write names instead of formulas? (<code>describe</code>).

<code>as.reaction</code>	logical, interpret input as a reaction?
<code>digits</code>	numeric, how many digits to round logarithms of activities.
<code>basis</code>	numeric or character, species number or formula.
<code>species</code>	Names or indices of species for which to calculate nonideal properties (<code>nonideal</code>), or dataframe, species definition such as that in <code>thermo\$species</code> (<code>which.balance</code>).
<code>proptable</code>	list of dataframes of species properties.
<code>IS</code>	numeric, ionic strength(s) used in nonideal calculations, mol kg ⁻¹ .
<code>logact</code>	numeric, logarithms of activity.
<code>length</code>	numeric, numbers of residues.
<code>logact.tot</code>	numeric, logarithm of total activity.
<code>n</code>	numeric, number of frame to go up.

Details

The argument of `protein.length`, if it is character, refers to the name of protein(s) (e.g., 'LYSC_CHICK') for which to calculate the length (number of amino acid residues). If the argument is numeric, it refers to the species index of a protein (value in `thermo$species$ispecies`). For a positive numeric argument to work, the protein information must have been previously loaded into the species list (using `info`). If the numeric value is negative, it refers to the rownumber of the protein in `thermo$protein`.

`MP90.cp` takes `T` (one or more temperatures in °C) and `protein` (name of protein) and returns the heat capacity of the unfolded protein using values of heat capacities of the residues taken from Makhatazde 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.

`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 ΔS and ΔV 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.

`describe` generates a textual representation of the temperature, pressure, and logarithms of activities of the basis species, given in the arguments by `x` (i.e. the dataframe in `thermo$basis`) and `T` and `P` (given in Kelvin and bar and converted by the function to those specified by `nuts`). The `digits` argument tells to what decimal place the logarithms of activities should be rounded. If any of the supplied arguments is NULL its specification is not printed in the output; `T` and `P`, if present, are prepended to the basis summary. If `x` instead is a dataframe representing a chemical reaction (as output by `subcrt` and identified by having a column named `coeff`), the function returns a textual summary of that reaction (i.e., showing reactants on the left, an equal sign, and products on the right; reactants and products are preceded by their reaction coefficient unless it is 1). However, if only reactants or products can be found, or `as.reaction` is set to FALSE, the names or formulas of the species are printed with their coefficients and interceding plus or minus signs, as appropriate. Whether the names or formulas are printed is controlled by `use.name` (FALSE by default), a logical vector the length of which corresponds to the number of rows in `x` (but is expanded to the right length if needed).

`basis.comp` calculates the composition of the given

`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`. `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`.

`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().

Value

Numeric returns are made by, `protein.length`, `dPdTtr`, `Ttr`, `MP90.cp`, `spearman`, `mod.obigt`. Functions with no (or unspecified) returns are `thermo.plot.new`, `thermo.postscript`, `label.plot` and `water.lines`.

References

Alberty, R. A. (2003) *Thermodynamics of Biochemical Reactions*, John Wiley & Sons, Hoboken, New Jersey, 397 p. <http://www.worldcat.org/oclc/51242181>

Makhatadze, G. I. and Privalov, P. L. (1990) Heat capacity of proteins. 1. Partial molar heat capacity of individual amino acid residues in aqueous solution: Hydration effect *J. Mol. Biol.* **213**, 375–384. [http://dx.doi.org/10.1016/S0022-2836\(05\)80197-4](http://dx.doi.org/10.1016/S0022-2836(05)80197-4)

See Also

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

Examples

```
## calculate protein length
protein.length("LYSC_CHICK")
# another way to do it
```

```

basis("CHNOS")
species("LYSC_CHICK")
protein.length(species())$ispecies)
# another way to do it
ip <- protein("LYSC","CHICK")
protein.length(-ip)

## heat capacity as a function of temperature
## (Makhatadze & Privalov, 1990) units: J mol-1
MP90.cp(c(5,25,50,75,100,125),"LYSC_CHICK")

## 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))

## the basis stoichiometry of a made-up species
# warns because P isn't in our basis
basis("CHNOS")
basis.comp("SPONCH")

## describing the basis species
basis("CHNOSe")
describe(thermo$basis)
describe(thermo$basis,T=NULL,P=NULL)

## 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(1.2, 0.3, 0), cex.axis = par("cex"), col = par("col"),
  yline = NULL, axs = "i", do.box = TRUE, ticks = NULL, las = 1,
  xline = NULL)
thermo.postscript(file, family = "Helvetica", width = 8,
  height = 6, horizontal = FALSE)
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)
axis.label(lab, opt = NULL, do.state = TRUE, oldstyle = FALSE,
  do.upper = FALSE, mol = "mol", state = NULL, as.expression = TRUE)
species.label(formula, do.state = FALSE, state = "", do.log = FALSE,
  as.expression = TRUE)
water.lines(xaxis = "pH", yaxis = "Eh", T = 298.15, P = "Psat",
  which = c("oxidation", "reduction"), logaH2O = 0, lty = 2,
  col = par("fg"), xpoints = NULL)
mtitle(main, ...)
residualsplot(residuals, property = "Cp", model = "big")
```

Arguments

xlim	numeric, limits of the <i>x</i> -axis.
ylim	numeric, limits of the <i>y</i> -axis.
xlab	character, <i>x</i> -axis label.
ylab	character, <i>y</i> -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 <code>side</code> (retained for backwards compatibility).
las	numeric, style for axis labels
xline	numeric, margin line on which to plot x -axis name.
file	character, path to a file.
family	character, font family.
width	numeric, width of plot.
height	numeric, height of plot.
horizontal	logical, create plot in landscape mode?
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.
opt	character or numeric, options for axis labels.
do.state	logical, append state abbreviation to label?
oldstyle	logical, use old style of axis labels?
do.upper	logical, use uppercase letters in axis label?
mol	character, string to use as the denominator of units in axis label.
formula	character, representation of chemical formula.
state	character, designation of physical state.
as.expression	logical, make the return value an expression?
do.log	logical, prepend a text indicating logarithm of activity or fugacity?
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 H ₂ O.
lty	numeric, line type.
xpoints	numeric, points to plot on x axis.
main	character, text for plot title.
...	further arguments passed to <code>mtext</code> .
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.postscript` calls `postscript` with some custom parameters used by the package author (and might be handy for other users of the package as well).

`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.

`axis.label` returns an `expression` to be used for plotting an axis label, which may be the symbol for a thermodynamic properties, chemical activity or fugacity, or one of 'T', 'P', 'Eh', 'pH', 'pe' or 'logK'. If `as.expression` is FALSE, the quoted text of the label is returned instead. An expression for chemical activity or fugacity is returned if the first argument is the name of one of the basis species (e.g., 'O2'). The expression in this case includes italic and subscripted symbols, unless `oldstyle` is TRUE, when labels with a simpler format (e.g. 'O2 (log f)') are returned. The default value of NULL of `opt` means to use the state this basis species is in, or if this basis species is not present to use the value in `thermooptstate`. Likewise, if `x` is 'T' or 'P' the units of temperature or pressure are determined using `nuts` (which also refers to `thermo$opt`). `do.upper`, if TRUE, tells the function to print the label using uppercase letters. Labels for properties can be generated by using e.g. 'DGf' or 'DG0r' as arguments. `mol` (default: 'mol') refers to the denominator of the units (default: molality); this can be changed to represent e.g. specific units, by setting `mol` to 'g'. `opt` when generating labels for properties indicates the prefix to place on the units.

`species.label` is like `axis.label` but is specifically intended for generating expressions for the chemical formulas of species. The state (in parentheses) is included in the expression only if `state` is not NULL. If `do.log` is TRUE, the expression will contain a prefix in front of the formula that indicates chemical activity (like 'log_a' or 'log_f').

`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.

 util.seq

Functions to Work with Sequence Data

Description

Count amino acids in protein sequences, 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(seq, nchar=1)
nucleicacids(seq, type = "DNA", comp = NULL, comp2 = NULL)
```

Arguments

seq	character, amino acid sequence of a protein (aminoacids) or base sequence of a nucleic acid (nucleicacids).
nchar	numeric, 1 to return one-letter, 3 to return three-letter abbreviations for amino acids (aminoacids).
type	character, type of nucleic acid sequence (DNA or RNA) (nucleicads).
comp	character, type of complement sequence.
comp2	character, type of second complement sequence.

Details

aminoacids takes a character argument containing a protein sequence and counts the number of occurrences of each type of amino acid. The output is a dataframe with 20 columns, each corresponding to an amino acid, ordered in the same way as thermo\$protein. If the first argument is NULL, the function returns the one-letter abbreviations (for nchar equal to 1) or the three-letter ones (if nchar is equal to 3) or the names of the amino acids (if nchar is NA) of twenty amino acids in the order used in thermo\$protein.

nucleicacids takes a DNA or RNA sequence and counts the numbers of bases of each type. Whether the sequence is DNA or RNA is specified by type. Setting comp to 'DNA' or 'RNA' tells the function to compute the base composition of that type of complement of the sequence. If comp2 is specified, another complement is taken. The two rounds of complementing can be used in a single function call e.g. to go from a sequence on DNA minus strand (given in seq) to the plus strand (with comp="DNA") and then from the DNA plus strand to RNA (with comp2="RNA"). The value returned by the function is a dataframe of base composition, which can be passed back to the function to obtain the overall chemical formula for the bases.

Value

An object of type character or dataframe.

Examples

```
## count amino acids in a sequence
aminoacids("GGSGG")
aminoacids("WhatAmIMadeOf?")

## count nucleobases in a sequence
nucleicacids("ACCGGGTTT")
# the DNA complement of that sequence
nucleicacids("ACCGGGTTT", comp="DNA")
# the RNA complement of the DNA complement
n <- nucleicacids("ACCGGGTTT", comp="DNA", comp2="RNA")
# the formula of the RNA complement
nucleicacids(n, type="RNA")
```

util.stat

Functions for Various Statistical Operations

Description

Calculate root mean square deviation (RMSD), coefficient of variation of the RMSD, Spearman's rank correlation coefficient, or correlation coefficient of a q-q plot.

Usage

```
qqr(a)
spearman(a, b)
rmsd(a, b)
cvrmsd(a, b)
lograt(a, b)
```

Arguments

a numeric values.
b numeric values.

Details

qqr returns the correlation coefficient of a quantile-quantile plot using the values given in *y*.
spearman calculates Spearman's rank correlation coefficient between *a* and *b*. rmsd and cvrmsd return the root mean square deviation and coefficient of variation of the RMSD, respectively. The coefficient of variation is computed as the RMSD divided by the mean of the values in *a*.
lograt calculates the difference between *a* and *b* (i.e. \log_{10} of the ratio of activities *b/a*) where *a* is a list of single values and *b* is a list of values, any dimension.

See Also

These functions are among the target statistics available in [revisit](#).

Examples

```
a <- 1:9
b <- a + 1
spearman(a, b)      # 1
spearman(a, rev(b)) # -1
qqr(a)              # <1
rmsd(a, a)          # 0
rmsd(a, b)          # 1
cvrmsd(a, b)        # 1/5
```

util.units

Functions to Convert Units

Description

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

Usage

```
nuts(units = NULL)
convert(value, units, T = thermo$opt$Tr,
        P = thermo$opt$Pr, pH = 7, logaH2O = 0)
invert(value, units)
outvert(value, units)
```

Arguments

units	character, name of units to set in global preferences (<code>nuts</code>), or name of units (see table) to convert to or from.
value	numeric, value(s) to be converted.
T	numeric, temperature (Kelvin), used in 'G'-'logK', 'pe'-'Eh' and 'logfO2'-'E0' conversions.
P	numeric, pressure (bar), used in 'logfO2'-'E0' conversions.
pH	numeric, pH, used in 'logfO2'-'E0' conversions.
logaH2O	numeric, logarithm of activity of water, used in 'logfO2'-'E0' conversions.

Details

The global 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 `nuts`, 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). If the `units` argument to `nuts` is NULL (the default) the current units settings are printed (with a NULL return); if it is one of 'T', 'P' or 'E', the units for that property are printed and also returned.

The actual units conversions are handled by `convert`, through which values are transformed into destination units (names not case sensitive). The possible unit conversions are shown in the following table. Note that 'Eh' and 'E0' both stand for the value of Eh (do not confuse 'E0' with the standard reduction potential, which is written similarly); they have different names so that one can choose to convert to 'pe' or 'logfO2'.

C	K	temperature
cal	J	energy
bar	MPa	pressure
E0	logfO2	oxidation state
G	logK	energy
cm3bar	calories	energy
pe	Eh	oxidation state

`invert` and `outvert` are wrappers for `convert` that handle the conditional conversion of values from or to the user's units. The name of the function `nuts` is such to avoid clashing with the name of `units`, which is present in the R base package.

Value

Numeric, except for `nuts`, which returns character or NULL.

Examples

```
## 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)
# logfO2 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 user unit preference
nuts("K") # set temperature units
nuts("J") # set energy units
# return the name of the (E)nergy units
nuts("E")
# print names of all units (NULL return)
nuts()
# defaults
nuts("K"); nuts("bar"); nuts("cal")

```

water

Properties of Water

Description

Calculate thermodynamic and electrostatic properties of water.

Usage

```

water(property = NULL, T = thermo$opt$Tr, P = "Psat")
water.SUPCRT92(property, T = 298.15, P = 1, isat = 0)
water.IAPWS95(property, T = 298.15, rho = 1000)
water.WP02(property, T = 298.15)
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	character or numeric, 'Psat'; or pressure (bar for water, water.SUPCRT92; MPa for water.AW90, water.IAPWS95, water.WP02).
isat	numeric, if 1, calculate values of 'Psat'
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 H₂O using equations of state taken from the literature. The wrapper function `water` responds to two computational options. The default option (for `thermooptwater` equal to 'SUPCRT') indicates to 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 `thermooptwater`

is set to 'IAPWS', the thermodynamic properties are calculated using an implementation in R code (hence relatively slow) of the IAPWS-95 formulation (Wagner and Pruss, 2002) and electrostatic properties are calculated using the equations of Archer and Wang, 1990.

The allowed `property`s for `water` are one or more of those given below, depending on the computational option (availability is shown by an asterisk). The names of properties in the arguments are not case sensitive, and some of the properties that can actually be calculated using the equations of state are not implemented here.

Property	Description	Units	IAPWS	SUPCRT
A	Helmholtz energy	cal mol ⁻¹	*	*
G	Gibbs energy	cal mol ⁻¹	*	*
S	Entropy	cal K ⁻¹ mol ⁻¹	*	*
U	Internal energy	cal mol ⁻¹	*	*
H	Enthalpy	cal mol ⁻¹	*	*
Cv	Isochoric heat capacity	cal K ⁻¹ mol ⁻¹	*	*
Cp	Isobaric heat capacity	cal K ⁻¹ mol ⁻¹	*	*
w (Speed)	Speed of sound	cm s ⁻¹	NA	*
E	Isobaric expansivity	cm ³ K ⁻¹	NA	*
kT	Isothermal compressibility	cm ³ bar ⁻¹	NA	*
alpha	Coefficient of isobaric expansivity	K ⁻¹	NA	*
beta	Coefficient of isothermal compressibility	bar ⁻¹	NA	*
epsilon (diel)	Dielectric constant	dimensionless	NA	*
visc	Dynamic viscosity	g cm ⁻¹ s ⁻¹	NA	*
tcond	Thermal conductivity	cal cm ⁻¹ s ⁻¹ K ⁻¹	NA	*
tdiff	Thermal diffusivity	cm ² s ⁻¹	NA	*
Prndtl	Prandtl number	dimensionless	NA	*
visck	Kinematic viscosity	cm ² s ⁻¹	NA	*
albe	Isochoric expansivity -compressibility	bar K ⁻¹	NA	*
Z (ZBorn)	Z Born function	dimensionless	NA	*
Y (YBorn)	Y Born function	K ⁻¹	*	*
Q (QBorn)	Q Born function	bar ⁻¹	*	*
daldT	Isobaric temperature derivative of expansibility	K ⁻²	NA	*
X (XBorn)	X Born function	K ⁻²	*	*
N	N Born function	bar ⁻²	*	NA
UBorn	U Born function	bar ⁻¹ K ⁻¹	*	NA
V	Volume	cm ³ mol ⁻¹	*	*
rho	Density	kg cm ³	*	*
Psat	Saturation vapor pressure	bar	*	*
P	Pressure	bar	*	NA
de.dT	Temperature derivative of dielectric constant	K ⁻¹	*	NA
de.dP	Pressure derivative of dielectric constant	bar ⁻¹	*	NA

UBorn refers to the *U* Born function, 'U' to internal energy. The coefficients of isobaric expan-

sivity and of isothermal compressibility are defined as $(1/V)(dV/dT)_P$ and $-(1/V)(dV/dP)_T$ respectively. All of the properties are calculated as a function of temperature and pressure except 'P_{sat}' f(T) (values supplied in the argument P are ignored) and 'P' f(T,rho). Except for those of rho, the units used are as in Johnson and Norton, 1991. Names of properties that are used in `water.SUPCRT92` (but not in `water`) are shown in parentheses.

`water.SUPCRT92` 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). If `isat` is equal to 1 (or `TRUE`), the values of P are ignored and values of 'P_{sat}' are returned. 'P_{sat}' refers to one bar below 100 °C, otherwise to the vapor-liquid saturation pressure at temperatures below the critical point ('P_{sat}' is not available at temperatures above the critical point). `water.SUPCRT92` function 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 ([cgl](#)). The IAPWS-95 formulation also has provisions for computing the properties of steam, but these are currently not used by `CHNOSZ`.

`water.IAPWS95` provides an implementation of the IAPWS-95 formulation for properties (including pressure) calculated as a function of temperature and density. To compute the thermodynamic and electrostatic properties of water as a function of temperature and pressure using `water.IAPWS95`, `water` applies a root-finding function ([uniroot](#)) to determine the corresponding values of density. Electrostatic properties in this case are derived from values of the static dielectric constant (`epsilon`) calculated using equations given by Archer and Wang, 1990 and coded in `water.AW90`. Note that the `water.AW90` computes the static dielectric constant at given temperatures and pressures, so `water` contains routines for calculating its derivatives with respect to temperature and pressure. A keyword, 'test', may be given as `property` to `water.IAPWS95`, which causes the printing of two tables, one representing the ideal-gas and residual contributions to the Helmholtz free energy (Table 6.6 of Wagner and Pruss, 2002), and a second with a selection of calculated properties for the liquid and vapor at the triple and boiling points.

The `water.IAPWS95` function returns values of thermodynamic properties in specific units (per gram) which are converted to molar properties by `water`. 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). For compatibility with geochemical modeling conventions, the values of Gibbs energy, enthalpy and entropy output by `water.IAPWS95` are converted by `water` to the triple point reference state adopted in `SUPCRT92` (Johnson and Norton, 1991; Helgeson and Kirkham, 1974). Auxiliary equations to the IAPWS-95 formulation (Wagner and Pruss, 2002) are provided in `water.WP02`; 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).

The stated temperature limits of validity of calculations in `water.SUPCRT92` are from the greater of 0 °C or the melting temperature at pressure to 2250 °C (Johnson et al., 1992); for `water.IAPWS` 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 30000 bar (`water.SUPCRT92`) or 10000 bar (`water.IAPWS95`; again, 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 re-

turn 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

For `water`, `water.SUPCRT92` and `water.IAPWS`, a dataframe 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

- Archer, D. G. and Wang, P. M. (1990) The dielectric constant of water and Debye-Huckel limiting law slopes. *J. Phys. Chem. Ref. Data* **19**, 371–411. <http://www.nist.gov/srd/PDFfiles/jpcrd383.pdf>
- Haar, L., Gallagher, J. S. and Kell, G. S. (1984) *NBS/NRC Steam Tables*. Hemisphere, Washington, D. C., 320 p. <http://www.worldcat.org/oclc/301304139>
- Helgeson, H. C. and Kirkham, D. H. (1974) Theoretical prediction of the thermodynamic behavior of aqueous electrolytes at high pressures and temperatures. I. Summary of the thermodynamic/electrostatic properties of the solvent. *Am. J. Sci.* **274**, 1089–1098. <http://www.ajsonline.org/cgi/content/abstract/274/10/1089>
- Johnson, J. W. and Norton, D. (1991) Critical phenomena in hydrothermal systems: state, thermodynamic, electrostatic, and transport properties of H₂O in the critical region. *Am. J. Sci.* **291**, 541–648. <http://www.ajsonline.org/cgi/content/abstract/291/6/541>
- Johnson, J. W., Oelkers, E. H. and Helgeson, H. C. (1992) SUPCRT92: A software package for calculating the standard molal thermodynamic properties of minerals, gases, aqueous species, and reactions from 1 to 5000 bar and 0 to 1000°C. *Comp. Geosci.* **18**, 899–947. [http://dx.doi.org/10.1016/0098-3004\(92\)90029-Q](http://dx.doi.org/10.1016/0098-3004(92)90029-Q)
- Levelt-Sengers, J. M. H., Kamgarpari, B., Balfour, F. W. and Sengers, J. V. (1983) Thermodynamic properties of steam in the critical region. *J. Phys. Chem. Ref. Data* **12**, 1–28. <http://www.nist.gov/srd/PDFfiles/jpcrd214.pdf>
- Wagner, W. and Pruss, A. (2002) The IAPWS formulation 1995 for the thermodynamic properties of ordinary water substance for general and scientific use. *J. Phys. Chem. Ref. Data* **31**, 387–535. <http://dx.doi.org/10.1063/1.1461829>

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 `thermooptwater`. Equations of state for species other than water are coded in `hkf` and `cgl`.

Examples

```
## set temperature, density
T <- 500; rho <- 838.0235
# calculate pressure
P <- as.numeric(water.IAPWS95('P', T=T, rho=rho))
# output table of test values
```

```

water.IAPWS95('test')
# calculate dielectric constant
water.AW90(T=T,rho=rho,P=P)
# find water density for this T, P
water('rho',T=T,P=convert(P,'bar'))

## density along saturation curve
T <- seq(273.15,623.15,25)
water.WP02(T=T) # liquid from WP02
water.WP02('rho.vapor',T) # steam from WP02
water('rho',T=T,P='Psat') # liquid from SUPCRT92
# values of the density, Psat, Gibbs energy
water(c('rho','psat','G'),T=T,P='Psat')
# derivatives of the dielectric constant (Born functions)
water(c('Q','Y','X','U'),T=T)
# now at constant pressure
water(c('Q','Y','X','U'),T=T,P=2000)

## NaCl dissociation logK f(T,P)
# after Shock et al., 1992, Fig. 1
# make note of the warning in the subcrt help page
species <- c('NaCl','Na+','Cl-')
coeffs <- c(-1,1,1)
# start a new plot with the experimental data
thermo.plot.new(xlim=c(0,1000),ylim=c(-5.5,1),
  xlab=axis.label("T"),ylab=axis.label("logK"))
expt <- read.csv(system.file("extdata/cpetc/SOJSH.csv",package="CHNOSZ"))
points(expt$T,expt$logK,pch=expt$pch)
T <- list(seq(0,370,25),seq(265,465,25),
  seq(285,760,25),seq(395,920,25))
for(i in 5:9) T[[i]] <- seq(400,1000,25)
P <- list("Psat",500,1000,1500,2000,2500,3000,3500,4000)
for(i in 1:length(T)) {
  s <- subcrt(species,coeffs,T=T[[i]],P=P[[i]])
  lines(s$out$T,s$out$logK)
}
legend("bottomleft",pch=unique(expt$pch),
  legend=unique(expt$source))
title(main=paste('NaCl(aq) = Na+ + Cl-\n',
  'Psat and 500-4000 bar, after Shock et al., 1992'))

## comparing the computational options
prop <- c('A','G','S','U','H','Cv','Cp','w','epsilon',
  'Y','Q','X','rho','Psat')
thermo$opt$water <- 'SUPCRT'
print(water(prop,T=convert(c(25,100,200,300),'K'))))
thermo$opt$water <- 'IAPWS'
print(water(c(prop,'N','UBorn'),T=convert(c(25,100,200,300),'K'))))
# fixme: things seem to be working except speed of
# sound in our IAPWS calculations

# calculating Q Born function
# after Table 22 of Johnson and Norton, 1991

```

```
thermo$opt$water <- 'SUPCRT'  
T <- rep(c(375,400,425,450,475),each=5)  
P <- rep(c(250,300,350,400,450),5)  
w <- water('Q',T=convert(T,'K'),P=P)  
# the rest is to make a readable table  
w <- as.data.frame(matrix(w[[1]],nrow=5))  
colnames(w) <- T[1:5*5]  
rownames(w) <- P[1:5]  
print(w)
```

Index

- *Topic **datasets**
 - extdata, 39
 - thermo, 82
- *Topic **extra**
 - anim.TCA, 12
 - EOSregress, 32
 - eqdata, 35
 - examples, 38
- *Topic **package**
 - CHNOSZ-package, 3
- *Topic **primary**
 - affinity, 6
 - basis, 14
 - diagram, 20
 - findit, 44
 - get.expr, 47
 - get.protein, 48
 - info, 54
 - protein, 62
 - revisit, 68
 - species, 71
 - subcrt, 73
 - taxonomy, 80
 - transfer, 89
- *Topic **secondary**
 - buffer, 16
 - eos, 29
 - equil, 37
 - ionize, 56
 - makeup, 60
 - water, 119
- *Topic **util**
 - util.args, 92
 - util.array, 93
 - util.blast, 95
 - util.character, 97
 - util.data, 99
 - util.fasta, 102
 - util.formula, 104
 - util.list, 107
 - util.misc, 108
 - util.plot, 112
 - util.seq, 115
 - util.stat, 116
 - util.units, 117
 - .onAttach, 83
 - [, 93
 - add.obigt, 42, 83, 88
 - add.obigt (*util.data*), 99
 - add.protein, 50, 87, 88, 103
 - add.protein (*util.data*), 99
 - affinity, 3, 6, 17, 21, 22, 25, 38, 39, 57, 64, 65, 69, 72, 75, 83, 87, 93, 94, 103, 108, 118
 - allparents (*taxonomy*), 80
 - aminoacids, 108
 - aminoacids (*util.seq*), 115
 - anim (*anim.TCA*), 12
 - anim.TCA, 12, 42, 94, 101
 - apc (*transfer*), 89
 - array, 93
 - as.expression, 34
 - axis.label (*util.plot*), 112
 - barchart, 114
 - basis, 3, 7–9, 14, 17, 45, 61, 72, 75, 87
 - basis.comp (*util.misc*), 108
 - browse.refs (*util.data*), 99
 - browseURL, 100
 - buffer, 3, 6, 8, 9, 14, 16, 17, 25, 86
 - buffers (*thermo*), 82
 - c2s, 61
 - c2s (*util.character*), 97
 - caller.name (*util.misc*), 108
 - can.be.numeric (*util.character*), 97
 - cgl, 3, 74, 77, 86, 121, 122

- cgl (*eos*), 29
- change, 18
- change (*util.data*), 99
- check.obigt, 42, 55
- check.obigt (*util.data*), 99
- checkEOS (*util.data*), 99
- checkGHS (*util.data*), 99
- CHNOSZ-package, 3
- colSums, 94
- convert, 7, 9
- convert (*util.units*), 117
- cvrmsd (*util.stat*), 116

- data, 100
- data.frame, 4, 64, 83
- def2gi, 96
- def2gi (*util.blast*), 95
- describe (*util.misc*), 108
- diagram, 3, 4, 7–9, 20, 37, 39, 42, 68, 69, 101, 110, 118
- dimSums (*util.array*), 93
- diversity, 70
- dPdTtr, 75
- dPdTtr (*util.misc*), 108
- draw.transfer (*transfer*), 89

- element, 84, 105
- element (*util.formula*), 104
- energy (*affinity*), 6
- envert (*util.units*), 117
- eos, 29
- eos.args (*util.args*), 92
- EOScalc (*EOSregress*), 32
- EOScoeffs (*EOSregress*), 32
- EOSlab (*EOSregress*), 32
- EOSplot (*EOSregress*), 32
- EOSregress, 32, 41
- EOSvar (*EOSregress*), 32
- eqdata, 35
- equil, 37
- equil.boltz, 3, 23, 108
- equil.react, 3, 23
- example, 38
- examples, 4, 38
- expand.formula (*util.formula*), 104
- expression, 114
- extdata, 13, 39, 49, 82, 87, 96, 101
- extremes (*revisit*), 68

- feldspar (*transfer*), 89
- findit, 4, 39, 44, 48

- get.expr, 40, 42, 47, 87
- get.protein, 23, 41, 42, 47, 48, 65, 87
- getnames (*taxonomy*), 80
- getnodes (*taxonomy*), 80
- getrank (*taxonomy*), 80
- GHS, 62
- GHS (*util.formula*), 104
- grep, 47, 103
- grep.file (*util.fasta*), 102
- grid, 74
- groups (*thermo*), 82

- heat.colors, 13, 24
- help.start, 4
- hkf, 3, 74, 76, 77, 85, 122
- hkf (*eos*), 29

- id.blast, 40, 42
- id.blast (*util.blast*), 95
- info, 15, 32, 54, 65, 72, 74, 77, 100, 101, 109
- install.packages, 4
- invisible, 24, 49, 55, 96, 100
- ionize, 8, 9, 17, 18, 41, 56, 57, 65
- is.fasta (*util.fasta*), 102

- label.plot (*util.plot*), 112
- lapply, 107, 108
- legend, 21, 23
- library, 4
- list, 83
- list2array (*util.array*), 93
- lm, 33, 34
- lograt (*util.stat*), 116
- longex (*examples*), 38
- lsub (*util.list*), 107
- lsum (*util.list*), 107

- makeup, 3, 15, 30, 60, 77, 83, 104, 105, 110
- mod.buffer, 100, 101
- mod.buffer (*buffer*), 16
- mod.obigt (*util.data*), 99
- MP90.cp (*util.misc*), 108
- mtext, 114
- mtime (*util.plot*), 112
- mylapply (*util.list*), 107

- names, 24, 45

- nonideal, 75
- nonideal (*util.misc*), 108
- nucleicacids (*util.seq*), 115
- nuts, 8, 63, 74, 109, 114, 118
- nuts (*util.units*), 117

- OBIGT (*thermo*), 82
- obigt2eos (*util.data*), 99
- opt (*thermo*), 82
- outvert (*util.units*), 117

- par, 24, 25, 110, 114
- parent (*taxonomy*), 80
- paste, 110
- plot.findit (*findit*), 44
- plot.new, 114
- png, 38
- postscript, 114
- pprod (*util.list*), 107
- protein, 3, 25, 55, 62
- protein.formula (*util.formula*), 104
- protein.info, 57
- protein.length, 108
- protein.length (*util.misc*), 108
- psum (*util.list*), 107

- qqr (*util.stat*), 116

- rainbow, 24
- read.blast, 40
- read.blast (*util.blast*), 95
- read.fasta, 3, 47
- read.fasta (*util.fasta*), 102
- read.supcrt (*subcrt*), 73
- readLines, 103
- refs (*thermo*), 82
- residualsplot (*util.plot*), 112
- residue.formula (*util.formula*), 104
- residue.info, 57
- residue.info (*protein*), 62
- revisit, 4, 41, 45, 68, 103, 114, 117
- richness (*revisit*), 68
- rmsd (*util.stat*), 116
- rowSums, 94

- s2c (*util.character*), 97
- sciname (*taxonomy*), 80

- slice (*util.array*), 93
- spearman (*util.stat*), 116
- species, 3, 7–9, 14, 15, 45, 57, 61, 64, 65, 71, 72, 87, 90
- species.label (*util.plot*), 112
- splinefun, 109
- splitline (*util.fasta*), 102
- state.args (*util.args*), 92
- stress (*thermo*), 82
- strip (*diagram*), 20
- subcrt, 3, 7, 9, 30–32, 41, 61, 65, 72, 73, 92, 101, 109, 118
- substitute, 34
- substr, 61, 103, 110
- sum.refseq (*util.fasta*), 102
- system, 12

- taxonomy, 42, 80
- text, 110, 114
- thermo, 3, 4, 6, 8, 14, 17, 30, 34, 47–49, 55, 61–63, 72, 74, 76, 82, 100, 101, 103–105, 109, 114, 119, 122
- thermo.axis (*util.plot*), 112
- thermo.plot.new (*util.plot*), 112
- thermo.postscript (*util.plot*), 112
- title, 21
- tolower, 110
- TP.args (*util.args*), 92
- transfer, 4, 7, 89, 110
- trimfas (*util.fasta*), 102
- Ttr (*util.misc*), 108

- uniroot, 38, 121, 122
- unitize, 47
- unitize (*util.misc*), 108
- units, 118
- util.args, 92
- util.array, 93
- util.blast, 95
- util.character, 97
- util.data, 99
- util.fasta, 41, 102
- util.formula, 104
- util.list, 107
- util.misc, 108
- util.plot, 112
- util.seq, 115
- util.stat, 116
- util.units, 117

water, 6, 31, 32, 41, 74, 76, 77, 83, 88, 92,
119
water.lines (*util.plot*), 112
where.extreme, 45
where.extreme (*revisit*), 68
which.balance, 23
which.balance (*util.misc*), 108
which.pmax (*util.list*), 107
write.blast, 40
write.blast (*util.blast*), 95
write.supcrt (*subcrt*), 73

yeastgfp, 40
yeastgfp (*get.protein*), 48

ZC, 42
ZC (*util.formula*), 104