Package 'ChargeTransport'

February 19, 2015

Type Package

Title Charge Transfer Rates and Charge Carrier Mobilities

Version 1.0.2

Date 2014-02-17

Depends R (>= 3.0), parallel (>= 3.0)

Author Julien Idé and Guido Raos

Maintainer Julien Idé <julien.ide.fr@gmail.com>

Description This package provides functions to compute Marcus, Marcus-Levich-Jortner or Landau-Zener charge transfer rates. These rates can then be used to perform kinetic Monte Carlo simulations to estimate charge carrier mobilities in molecular materials. The preparation of this package was supported by the the Fondazione Cariplo (PLENOS project, ref. 2011-0349).

License GPL

Encoding UTF-8

LazyData true

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-06-04 14:51:05

R topics documented:

`activationFreeEnergy` *Activation Free Energy*

Description

Computes the non-adiabatic activation energy ($\Delta E^{\ddagger}$) involved in the calculation of the Landau-Zener and Marcus charge transfer (CT) rates using the following expression:

$$\Delta E^{\ddagger} = \frac{\left(\lambda + \Delta E^{0} + \Delta E_{Field}\right)^{2}}{4\lambda}$$

Usage

```
activationFreeEnergy(lambda, dE0 = 0, dEField = 0)
```

Arguments

`lambda`	a scalar, a vector, a matrix or an array containing the total reorganization energies (**in eV**) used to calculate the activation energy.
`dE0`	a scalar, a vector, a matrix or an array containing the site energy differences (**in eV**) used to calculate the activation energy. By default self-exchange reactions are considered (`dE0=0`).
`dEField`	a scalar, a vector, a matrix or an array containing an additional contribution to the site energy differences due to an external electric field (**in eV**). By default no electric field is applied (`dEField=0`).

Details

The arguments of these function can be scalars, vectors, matrices or arrays. Mixing scalar values with vectors, matrices or arrays is allowed but in all other cases the arguments must have the same dimensions and lengths.

Value

Depending on the dimension of the objects passed to the function a scalar, a vector, a matrix or an array containing the activation energies (**in eV**) is returned.

References

H. Oberhofer and J. Blumberger, *Physical Chemistry Chemical Physics*, 14, 13846–13852, **2012**

B. S. Brunschwig, J. Logan, M. D. Newton and N. Sutin, *Journal of the American Chemical Society*, 102, 5798, **1980**

M. D. Newton and N. Sutin, *Annual Review of Physical Chemistry*, 35, 437, **1984**

M. D. Newton, *Chemical Reviews*, 91, 767, **1991**.

A. Nitzan, Chemical Dynamics in Condensed Phases, Oxford University Press, **2006**

See Also

Marcus, LandauZener, dEField

Examples

```
## Produce a map of the non-adiabatic free energy barrier and
## two other maps of the non-adiabatic free energy barrier corrected by
## the deviation between the adiabatic and non-adiabatic potential energy curves at:
##   - the crossing point (dEad = dEna - abs(J))
##   - both the crossing point and the minimas of the curves (dEad = dEna - Delta)
## We assume that the reorganization energies are the following:
lambdaI <- 0.14  # internal reorganization energy in eV
lambdaS <- 36E-3 # external reorganization energy in eV
lambda <- lambdaI + lambdaS

N  <- 301
J  <- seq( 0  , 65,length.out=N)*1E-3 # eV
dE <- seq(-0.5,0.5,length.out=N)      # eV
G  <- expand.grid(J, dE)
J  <- G[,1]
dE <- G[,2]

dEna      <- activationFreeEnergy(lambda = lambda, dE0 = dE, dEField = 0)
dEad      <- dEna - adiabaticCorrection(J = J, lambda = lambda, dE0 = dE, dEField = 0)
dEadCross <- dEna -  J

dEna      <- matrix(dEna     , nrow = N, ncol = N)
dEad      <- matrix(dEad     , nrow = N, ncol = N)
dEadCross <- matrix(dEadCross, nrow = N, ncol = N)

addAxis <- function(bottom = TRUE, left = FALSE, above = FALSE, right = FALSE){
  useless <- lapply(1:4,axis, labels=FALSE)
  if(bottom) axis(1, labels = TRUE)
  if(left  ) axis(2, labels = TRUE)
  if(above ) axis(3, labels = TRUE)
  if(right ) axis(4, labels = TRUE)
  if(bottom) mtext(side=1,line=1.2, text=expression( abs(J)/eV), cex=par("cex"))
  if(left  ) mtext(side=2,line=1.2, text=expression(Delta*E/eV), cex=par("cex"))
  if(right ) mtext(side=4,line=1.2, text=expression(Delta*E/eV), cex=par("cex"))
  box()
}

layout(matrix(1:3, ncol=3))
par(cex=2, lwd=1.5, pty="s", mgp=c(1.1,0.1,0), tck=0.02, mar=rep(0.7,4), oma=rep(2,4))
contour(unique(J), unique(dE), dEna     , levels = seq(-0.1,0.6,0.1),
        xaxt="n", yaxt="n", labcex=2)
addAxis(TRUE, TRUE, FALSE, FALSE)
title(expression(Delta*E[na]), line=1)
contour(unique(J), unique(dE), dEadCross, levels = seq(-0.1,0.6,0.1),
```

```
        xaxt="n", yaxt="n", labcex=2)
addAxis(TRUE, FALSE, FALSE, FALSE)
title(expression(Delta*E[na]-J), line=1)
contour(unique(J), unique(dE), dEad     , levels = seq(-0.1,0.6,0.1),
        xaxt="n", yaxt="n", labcex=2)
addAxis(TRUE, FALSE, FALSE, TRUE)
title(expression(Delta*E[na]-Delta), line=1)
```

---

dEField *Site Energy Difference due to an Electric Field*

Description

For a given type of charge carrier, this function computes the site energy differences due to an external electric field ($dE_{Field} = -e\,\overrightarrow{d}\,.\,\overrightarrow{F}$ for a hole and $dE_{Field} = e\,\overrightarrow{d}\,.\,\overrightarrow{F}$ for an electron)

Usage

```
dEField(dx, dy, dz, carrier = "e", Fx = 0, Fy = 0, Fz = 1E5)
```

Arguments

dx, dy, dz	scalars, vectors, matrices or arrays containing respectively the x-, y- and z-components of the inter-sites distances (**in Angstrom**).
carrier	a character string indicating the type of charge carrier for which to compute the site energy differences due to the electric field. Can be either "e" or "h", respectively for electron and hole. By default electron transport (carrier="e") is considered.
Fx, Fy, Fz	scalar values giving the x-, y- and z-component of the electric field (**in V.cm-1**). By default an electric field of 1E5 V.cm-1 is applied along the z-axis.

Details

The arguments of these function can be scalars, vectors, matrices or arrays. Mixing scalar values with vectors, matrices or arrays is allowed but in all other cases the arguments must have the same dimensions and lengths.

Value

Depending on the dimension of the objects passed to the function a scalar, a vector, a matrix or an array containing the site energy differences due to the electric field (**in eV**) is returned.

See Also

energyConversion, Marcus, MarcusLevichJortner, LandauZener, KMC

Examples

```
dz <- rnorm(10,3.5,0.3)
Fnorm=1E5
# Electron transport along a 1D-stack of 10 molecules aligned along the z-axis
dEField(dx=0, dy=0, dz=dz, carrier="e", Fx=0, Fy=0, Fz=Fnorm)

# Hole transport along a 1D-stack of 10 molecules aligned along the z-axis
dEField(dx=0, dy=0, dz=dz, carrier="h", Fx=0, Fy=0, Fz=Fnorm)

# Some disorder in the xy-plan
dx <- rnorm(10,0,1)
dy <- rnorm(10,0,1)
dEField(dx=dx, dy=dy, dz=dz, carrier="e", Fx=0, Fy=0, Fz=Fnorm)

# Change the orientation of the electric field
theta = 45
dEField(dx=dx, dy=dy, dz=dz, carrier="h",
        Fx=0, Fy=Fnorm*sin(theta*pi/180), Fz=Fnorm*cos(theta*pi/180))
```

---

energyConversion *Energy Conversion*

Description

Conversion of energy units.

Usage

```
centimeterMinusOne2electronVolt(x)
centimeterMinusOne2Hartree(x)
centimeterMinusOne2Hertz(x)
centimeterMinusOne2Joule(x)
centimeterMinusOne2Kelvin(x)
centimeterMinusOne2kiloCaloriePerMole(x)
centimeterMinusOne2kiloJoulePerMole(x)

electronVolt2centimeterMinusOne(x)
electronVolt2Hartree(x)
electronVolt2Hertz(x)
electronVolt2Joule(x)
electronVolt2Kelvin(x)
electronVolt2kiloCaloriePerMole(x)
electronVolt2kiloJoulePerMole(x)

Hartree2centimeterMinusOne(x)
Hartree2electronVolt(x)
Hartree2Hertz(x)
Hartree2Joule(x)
```

```
Hartree2Kelvin(x)
Hartree2kiloCaloriePerMole(x)
Hartree2kiloJoulePerMole(x)

Hertz2centimeterMinusOne(x)
Hertz2electronVolt(x)
Hertz2Hartree(x)
Hertz2Joule(x)
Hertz2Kelvin(x)
Hertz2kiloCaloriePerMole(x)
Hertz2kiloJoulePerMole(x)

Joule2centimeterMinusOne(x)
Joule2electronVolt(x)
Joule2Hartree(x)
Joule2Hertz(x)
Joule2Kelvin(x)
Joule2kiloCaloriePerMole(x)
Joule2kiloJoulePerMole(x)

Kelvin2centimeterMinusOne(x)
Kelvin2electronVolt(x)
Kelvin2Hartree(x)
Kelvin2Hertz(x)
Kelvin2Joule(x)
Kelvin2kiloCaloriePerMole(x)
Kelvin2kiloJoulePerMole(x)

kiloCaloriePerMole2centimeterMinusOne(x)
kiloCaloriePerMole2electronVolt(x)
kiloCaloriePerMole2Hartree(x)
kiloCaloriePerMole2Hertz(x)
kiloCaloriePerMole2Joule(x)
kiloCaloriePerMole2Kelvin(x)
kiloCaloriePerMole2kiloJoulePerMole(x)

kiloJoulePerMole2centimeterMinusOne(x)
kiloJoulePerMole2electronVolt(x)
kiloJoulePerMole2Hartree(x)
kiloJoulePerMole2Hertz(x)
kiloJoulePerMole2Joule(x)
kiloJoulePerMole2Kelvin(x)
kiloJoulePerMole2kiloCaloriePerMole(x)
```

Arguments

x	an R object containing numerical values to be converted

Details

These functions use the `universalConstants` data set to convert energy values from a given unit to an other. Their are first converted to Joule and then to the desired unit. With care, the functions may be applied also to otherquantities (e.g. temperatures, wavenumbers, etc...)

Value

Return `x` multiplied by conversion factors.

See Also

`universalConstants`

Examples

```
# Return the Planck constant in eV.s
Joule2electronVolt(universalConstants["h","Value"])

# Return the Boltzmann constant in eV.K-1
Joule2electronVolt(universalConstants["kb","Value"])

# Convert electron Volt into cm-1 (E = h.c.nu_bar)
electronVolt2centimeterMinusOne(0.2)

# Convert electron Volt into Hz (E = h.nu)
electronVolt2Hertz(0.2)

# Convert Kelvin into electron Volt (E = kb.T)
Kelvin2electronVolt(300)
```

`KMC` *Charge Transport Simulation*

Description

Performs a kinetic Monte Carlo (KMC) simulation to propagate a single charge carrier within a given percolation network. Either the Bortz-Kalos-Lebowitz (BKL) or the First Reaction Method (FRM) algorithm can be used to propagate the charge carrier. The function is parallelized over the number of KMC simulations by using the `parLapply` function of the "parallel" package.

Usage

```
KMC(cl, con, rates, dx, dy, dz, type = "BKL", nSimu = 10, nHops = 1E7, seed = NULL)
```

Arguments

cl	a cluster object as returned by makeCluster.
con	a two-column matrix containing the "connectivity" of the percolation network. Each row contains the labels/indexes of a pair of sites within the percolation network.
rates	a matrix containing the charge transfer rates for each pair of sites (columns) and each frame of a molecular dynamics (rows).
dx, dy, dz	matrices containing respectively the x-, y- and z-components of the inter-site distances for each pair of sites (columns) and each frame of a molecular dynamics (rows).
type	a character string specifying the type of KMC simulation to perform. Can be either the FRM (First Reaction Method) or BKL (Bortz-Kalos-Lebowitz) algorithm.
nSimu	an integer indicating the number of KMC simulations to be performed (Usually a large number). If possible it has to be a multiple of the number of nodes used for the parallelization.
nHops	an integer indication the number of hopping events to be performed before stopping each KMC simulation.
seed	an integer used to initialize the random number generator. If NULL, a random number is sampled from .Random.seed.

Details

The labels/indexes of the pair of sites defining the pathways forming the percolation network stored in con can either be integers or character strings. The dimensions of the different arguments must be consistent: rates, dx, dy, dz, must have the same dimensions and the number of rows of con must be equal to the number of columns of rates, dx, dy, dz. The first dimension (rows) of rates, dx, dy, dz allow to treat different frames of a molecular dynamics together, assuming that the connectivity of the percolation network is the same for each frame. If the connectivity slightly change during a molecular dynamics, the user can first carefully analysed the trajectory to define a suitable 'fixed' connectivity (For example, if sites 1 and 2 are neigbours at a point of the dynamics, then, the 1-2 and 2-1 rates have to be computed for all the frames). The dimnames of rates are used to set the dimnames of the different matrices/array returned by the function.

Value

A list of class 'KMC' with the following components:

distx, disty, distz	matrices containing for each KMC simulation (columns) and each frame (rows) the distance cover by the charge carrier respectively along the x-, y- and z-axis.
time	a matrix containing for each KMC simulation (columns) and each frame (rows) the drift time of the charge carrier.
nhop	a three-dimensions array containing the number of hopping events that have occured along each pathway of the percolation network (second dimension) for each frame (first dimension) and each KMC simulation (third dimension).

References

A. B. Boltz, M. H. Kalos, J. L. Lebowitz, *Journal of Computational Physics*, 17:10-18, **1975**
D. T. Gillespie, *Journal of Computational Physics* 22:403-434, **1976**

See Also

Marcus, MarcusLevichJortner, LandauZener

Examples

```
##########################################################################
## Electron transport within a periodic 1D-stack of 4 molecules:
##
##                        ... 4 | 1 2 3 4 | 1 ...
##
##########################################################################
################ Preparation of the percolation network ################
nMols <- 4
## The charge can hop toward the right neigbour
con <- matrix(c(1:nMols,2:nMols,1), ncol=2)
## or the left neigbour
con <- rbind(con,con[,2:1])
## The number of pathways forming the percolation network is then:
nPaths <- 2*nMols
path.names <- paste0("P",1:nPaths)
dimnames(con) <- list(path.names, c("mol.i","mol.f"))
print(con)
##########################################################################

############## Preparation of some data for the simulation ##############
## Lets consider a slight deformation of the stack along the z-axis over
## 11 frames of a molecular dynamics
nFrames <- 11
frame.names <- paste0("F", 1:nFrames)
dx <- matrix(0, ncol=nPaths, nrow=nFrames,
             dimnames=list(frame.names, path.names))
dy <- dx
# dz when hoping to the right
dz <- matrix(seq(3.0,4.0,length.out=nMols), ncol=nMols, nrow=nFrames)
# and when hoping to the left
dz <- cbind(dz, -dz)
dimnames(dz) <- dimnames(dx)
## The electronic couplings will slightly decrease
J  <- matrix(seq(10, 110, length.out=nFrames),
             ncol=nMols, nrow=nFrames)*1E-3
J  <- cbind( J, J)
dimnames(J) <- dimnames(dx)
## By symmetry all the sites have the same energy
dE <- 0
## We apply an electric field along the stack (z-axis)
Fn <- 1E5
F <- c(0,0,Fn)
```

```
## Lets consider the case of electron transport
carr <- "e"
## This introduce an addition term in the site energy differences.
dEFz <- dEField(dx,dy,dz,carr,F[1],F[2],F[3])
## Lets assum that:
lambda <- 0.14 # eV
#############################################################################

############### Calculation of the charge transfer rates ###############
## Using the Marcus expression:
k <- Marcus(J,lambda,dE,dEFz)
format(k, digits=2, scientific=TRUE)
#############################################################################

################### Execution of the KMC simulation ###################
cl <- makeCluster(2, outfile="") # Slave nodes output on master stdout
simuFz <- KMC(cl=cl, con=con, rates=k, dx=dx, dy=dy, dz=dz,
              type="BKL", nSimu=2, nHops=1E5)
#############################################################################

###### More simulations applying the electric field along x or y #######
F <- c(Fn,0,0)
dEFx <- dEField(dx,dy,dz,carr,F[1],F[2],F[3])
k <- Marcus(J,lambda,dE,dEFx)
simuFx <- KMC(cl=cl, con=con, rates=k, dx=dx, dy=dy, dz=dz,
              type="BKL", nSimu=2, nHops=1E5)
F <- c(0,Fn,0)
dEFy <- dEField(dx,dy,dz,carr,F[1],F[2],F[3])
k <- Marcus(J,lambda,dE,dEFy)
simuFy <- KMC(cl=cl, con=con, rates=k, dx=dx, dy=dy, dz=dz,
              type="BKL", nSimu=2, nHops=1E5)
#############################################################################

################# Calculation of the mobility tensor ##################
driftVelocity <- function(simu){
  V <- c(
         x=1E-8*mean(simu$distx/simu$time),
         y=1E-8*mean(simu$disty/simu$time),
         z=1E-8*mean(simu$distz/simu$time))
  return(V) # Return the average drift velocity in cm.s-1
}
mu.Fx <- -driftVelocity(simuFx)/Fn # The minus is for electron transport
mu.Fy <- -driftVelocity(simuFy)/Fn # The minus is for electron transport
mu.Fz <- -driftVelocity(simuFz)/Fn # The minus is for electron transport
mu <- cbind(mu.Fx, mu.Fy, mu.Fz)
eigen(mu)
#############################################################################
```

Description

Computes charge transfer (CT) rates using the semi-classic Landau-Zener expression obtained by combining transition state theory with the Landau-Zener treatment for non-adiabatic transitions (it includes an adiabatic correction to the Marcus expression):

$$k = \kappa_{el}\nu_n\Gamma_n exp\left[-\beta\left(\Delta E^{\ddagger} - \Delta\right)\right]$$

where κ_{el} and Γ_n are respectively the thermally averaged electronic transmission coefficient and nuclear tunneling factor. ν_n is the frequency of the effective vibrational mode assisting the reaction. $\Delta E^{\ddagger}$ is the non-adiabatic activation energy (see `activationFreeEnergy`) and Δ is a correction factor relating $\Delta E^{\ddagger}$ to the adiabatic activation energy ($= \Delta E^{\ddagger} - \Delta$). ($\beta = 1/k_B T$ where k_B is the Boltzmann constant and T the temperature).

The electronic transmission coefficient accounting for multiple crossings at the intersection region is given by:

$$\kappa_{el} = \frac{2P_{LZ}}{1 + P_{LZ}}$$

$$P_{LZ} = 1 - exp\left(-2\pi\gamma\right)$$

$$2\pi\gamma = \frac{\pi^{3/2}\left|J\right|^2}{h\nu_n\sqrt{\lambda k_B T}}$$

where P_{LZ} is the Landau-Zener transition probability for a single surface crossing event, λ is the reorganization energy and J is the electronic coupling. The activation free energy and the adiabatic correction factor are given by:

$$\Delta E^{\ddagger} = \frac{\left(\lambda + \Delta E^0 + \Delta E_{Field}\right)^2}{4\lambda}$$

$$\Delta = \left|J\right| + \frac{\lambda + \Delta E^0 + \Delta E_{Field}}{2} - \sqrt{\frac{\left(\lambda + \Delta E^0 + \Delta E_{Field}\right)^2}{4} + \left|J\right|^2}$$

The parameter $2\pi\gamma$ determines the adiabaticity of the reaction. If $2\pi\gamma << 1$ the reaction is non-adiabatic and the Landau-Zener becomes the Marcus expression. In the opposite limit $2\pi\gamma >> 1$ the ET is adiabatic and P_{LZ} and κ_{el} approach unity. Then, the rate expression is the same as for standard chemical reactions in the classical transition state approximation:

$$k_{ad} = \nu_n exp\left[-\beta\left(\Delta E^{\ddagger} - \Delta\right)\right]$$

Usage

```
LandauZener(J, lambda, nuN, dE0 = 0, dEField = 0, temp = 300, gammaN = 1)

electronicTransmissionCoefficient(J, lambda, nuN, temp = 300)

LandauZenerProbability(J, lambda, nuN, temp = 300)

adiabaticCorrection(J, lambda, dE0 = 0, dEField = 0)
```

Arguments

J	a scalar, a vector, a matrix or an array containing the electronic couplings (**in eV**) used to calculate the CT rates.
lambda	a scalar, a vector, a matrix or an array containing the total reorganization energies (**in eV**) used to calculate the CT rates.
nuN	a scalar giving the frequency of the effective vibrational mode assisting the CT process (**in eV**) used to calculate the CT rates.
dE0	a scalar, a vector, a matrix or an array containing the site energy differences (**in eV**) used to calculate the CT rates. By default self-exchange reactions are considered (dE0=0).
dEField	a scalar, a vector, a matrix or an array containing an additional contribution to the site energy differences due to an external electric field (**in eV**). By default no electric field is applied (dEField=0).
temp	a scalar giving the temperature (**in Kelvin**) at which to evalute the CT rates. By default CT rates are evaluated at room temperature (temp=300).
gammaN	a scalar giving the nuclear tunneling factor. By default the transition is assumed to occur without any tunneling effect (gammaN=1).

Details

The arguments of these function can be scalars, vectors, matrices or arrays. Mixing scalar values with vectors, matrices or arrays is allowed but in all other cases the arguments must have the same dimensions and lengths. Using matrices or arrays is useful to compute simultaneously several charge transfer rates for different pairs of molecules, structures ...

Value

Depending on the dimension of the objects passed to the function a scalar, a vector, a matrix or an array containing the Landau-Zener CT rates (**in s-1**) is returned.

References

H. Oberhofer and J. Blumberger, *Physical Chemistry Chemical Physics*, 14, 13846–13852, **2012**

B. S. Brunschwig, J. Logan, M. D. Newton and N. Sutin, *Journal of the American Chemical Society*, 102, 5798, **1980**

M. D. Newton and N. Sutin, *Annual Review of Physical Chemistry*, 35, 437, **1984**

M. D. Newton, *Chemical Reviews*, 91, 767, **1991**.

A. Nitzan, Chemical Dynamics in Condensed Phases, Oxford University Press, **2006**

See Also

energyConversion, dEField, activationFreeEnergy, Marcus, MarcusLevichJortner, KMC

Examples

```
## Produce a map of the decimal logarithm of the Marcus,
## Marcus-Levich-Jortner and Landau-Zener rate expressions for:
nuN <- 1445 # effective vibrational mode wavenumber in cm-1
lambdaI <- 0.14  # internal reorganization energy in eV
lambdaS <- 36E-3 # external reorganization energy in eV

N  <- 301
J  <- seq( 0  , 65,length.out=N)*1E-3 # eV
dE <- seq(-0.5,0.5,length.out=N)      # eV
G  <- expand.grid(J, dE)
J  <- G[,1]
dE <- G[,2]

kMLJ    <- MarcusLevichJortner(
           J = J, lambdaI = lambdaI, lambdaS = lambdaS,
           hBarW = centimeterMinusOne2electronVolt(nuN), dE0 = dE)
kMarcus <- Marcus(
           J = J, lambda = lambdaI+lambdaS, dE0 = dE)
kLZ     <- LandauZener(
           J = J, lambda = lambdaI+lambdaS,
           nuN = centimeterMinusOne2Hertz(nuN), dE0 = dE)

kMLJ    <- matrix(kMLJ   , nrow = N, ncol = N)
kMarcus <- matrix(kMarcus, nrow = N, ncol = N)
kLZ     <- matrix(kLZ    , nrow = N, ncol = N)

addAxis <- function(bottom = TRUE, left = FALSE, above = FALSE, right = FALSE){
  useless <- lapply(1:4,axis, labels=FALSE)
  if(bottom) axis(1, labels = TRUE)
  if(left  ) axis(2, labels = TRUE)
  if(above ) axis(3, labels = TRUE)
  if(right ) axis(4, labels = TRUE)
  if(bottom) mtext(side=1,line=1.2, text=expression( abs(J)/eV), cex=par("cex"))
  if(left  ) mtext(side=2,line=1.2, text=expression(Delta*E/eV), cex=par("cex"))
  if(right ) mtext(side=4,line=1.2, text=expression(Delta*E/eV), cex=par("cex"))
  box()
}

layout(matrix(1:3, ncol=3))
par(cex=2, lwd=1.5, pty="s", mgp=c(1.1,0.1,0), tck=0.02, mar=rep(0.7,4), oma=rep(2,4))
contour(unique(J), unique(dE), log10(kMLJ   ),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, TRUE, FALSE, FALSE)
title("Marcus-Levich-Jortner", line=1)
contour(unique(J), unique(dE), log10(kMarcus),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, FALSE, FALSE, FALSE)
title("Marcus", line=1)
contour(unique(J), unique(dE), log10(kLZ    ),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, FALSE, FALSE, TRUE)
```

```
title("Landau-Zener", line=1)
```

`Marcus` *Marcus Charge Transfer Rates*

Description

Computes charge transfer (CT) rates using the following semi-classical Marcus expression:

$$k = \frac{2\pi}{\hbar} J^2 \sqrt{\frac{1}{4\pi\lambda k_B T}} exp\left(-\frac{\left(\lambda + \Delta E^0 + \Delta E_{Field}\right)^2}{4\lambda k_B T}\right)$$

where J and $\Delta E^0 = E_{final} - E_{initial}$ are respectively the electronic coupling and the site energy difference between the initial and final electronic states involved in the charge transfer reaction, and λ is the total reorganization energy. ΔE_{Field} is an additional contribution to the site energy difference due to an external electric field and T is the temperature.

Usage

```
Marcus(J, lambda, dE0 = 0, dEField = 0, temp = 300)
```

Arguments

`J`	a scalar, a vector, a matrix or an array containing the electronic couplings (**in eV**) used to calculate the CT rates.
`lambda`	a scalar, a vector, a matrix or an array containing the total reorganization energies (**in eV**) used to calculate the CT rates.
`dE0`	a scalar, a vector, a matrix or an array containing the site energy differences (**in eV**) used to calculate the CT rates. By default self-exchange reactions are considered (`dE0=0`).
`dEField`	a scalar, a vector, a matrix or an array containing an additional contribution to the site energy differences due to an external electric field (**in eV**). By default no electric field is applied (`dEField=0`).
`temp`	a scalar giving the temperature (**in Kelvin**) at which to evalute the CT rates. By default CT rates are evaluated at room temperature (temp=300).

Details

The arguments of these function can be scalars, vectors, matrices or arrays. Mixing scalar values with vectors, matrices or arrays is allowed but in all other cases the arguments must have the same dimensions and lengths. Using matrices or arrays is useful to compute simultaneously several charge transfer rates for different pairs of molecules, structures ...

Value

Depending on the dimension of the objects passed to the function a scalar, a vector, a matrix or an array containing the Marcus CT rates (**in s-1**) is returned.

References

R.A. Marcus, *Journal of Chemical Physics*, 24:966, **1956**

See Also

energyConversion, dEField, MarcusLevichJortner, LandauZener, KMC

Examples

```
## Produce a map of the decimal logarithm of the Marcus,
## Marcus-Levich-Jortner and Landau-Zener rate expressions for:
nuN <- 1445 # effective vibrational mode wavenumber in cm-1
lambdaI <- 0.14  # internal reorganization energy in eV
lambdaS <- 36E-3 # external reorganization energy in eV

N  <- 301
J  <- seq( 0  , 65,length.out=N)*1E-3 # eV
dE <- seq(-0.5,0.5,length.out=N)      # eV
G  <- expand.grid(J, dE)
J  <- G[,1]
dE <- G[,2]

kMLJ    <- MarcusLevichJortner(
           J = J, lambdaI = lambdaI, lambdaS = lambdaS,
           hBarW = centimeterMinusOne2electronVolt(nuN), dE0 = dE)
kMarcus <- Marcus(
           J = J, lambda = lambdaI+lambdaS, dE0 = dE)
kLZ     <- LandauZener(
           J = J, lambda = lambdaI+lambdaS,
           nuN = centimeterMinusOne2Hertz(nuN), dE0 = dE)

kMLJ    <- matrix(kMLJ   , nrow = N, ncol = N)
kMarcus <- matrix(kMarcus, nrow = N, ncol = N)
kLZ     <- matrix(kLZ    , nrow = N, ncol = N)

addAxis <- function(bottom = TRUE, left = FALSE, above = FALSE, right = FALSE){
  useless <- lapply(1:4,axis, labels=FALSE)
  if(bottom) axis(1, labels = TRUE)
  if(left  ) axis(2, labels = TRUE)
  if(above ) axis(3, labels = TRUE)
  if(right ) axis(4, labels = TRUE)
  if(bottom) mtext(side=1,line=1.2, text=expression( abs(J)/eV), cex=par("cex"))
  if(left  ) mtext(side=2,line=1.2, text=expression(Delta*E/eV), cex=par("cex"))
  if(right ) mtext(side=4,line=1.2, text=expression(Delta*E/eV), cex=par("cex"))
  box()
}

layout(matrix(1:3, ncol=3))
par(cex=2, lwd=1.5, pty="s", mgp=c(1.1,0.1,0), tck=0.02, mar=rep(0.7,4), oma=rep(2,4))
contour(unique(J), unique(dE), log10(kMLJ   ),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, TRUE, FALSE, FALSE)
```

```
title("Marcus-Levich-Jortner", line=1)
contour(unique(J), unique(dE), log10(kMarcus),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, FALSE, FALSE, FALSE)
title("Marcus", line=1)
contour(unique(J), unique(dE), log10(kLZ    ),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, FALSE, FALSE, TRUE)
title("Landau-Zener", line=1)
```

`MarcusLevichJortner` *Marcus-Levich-Jortner Charge Transfer Rates*

Description

Compute charge transfer (CT) rates using the Marcus-Levich-Jortner expression:

$$k = \frac{2\pi}{\hbar} J^2 \sqrt{\frac{1}{4\pi\lambda_s k_B T}} \times$$

$$\sum_{n=0}^{\infty} exp(-S)\frac{S^n}{n!} exp\left(-\frac{\left(\Delta E^0 + \Delta E_{Field} + \lambda_s + n\hbar\omega\right)^2}{4\lambda_s k_B T}\right)$$

where J and $\Delta E^0 = E_{final} - E_{initial}$ are respectively the electronic coupling and the site energy difference between the initial and final electronic states involved in the CT reaction, and λ_s is the external (intermolecular) reorganization energy. $S = \lambda_i/\hbar\omega$ is the electron-phonon coupling (Huang-Rhys factor), directly related to the internal (intramolecular) reorganization energy (λ_i) and to the energy of the effective vibrational mode ($\hbar\omega$) assisting the CT process. ΔE_{Field} is an additional contribution to the site energy difference due to an external electric field and T is the temperature.

Usage

```
MarcusLevichJortner(J, lambdaI, lambdaS, hBarW,
                    dE0 = 0, dEField = 0, temp = 300, nVib = 50)
```

Arguments

`J`	a scalar, a vector, a matrix or an array containing the electronic couplings (**in eV**) used to calculate the CT rates.
`lambdaI`	a scalar, a vector, a matrix or an array containing the internal reorganization energies (**in eV**) used to calculate the CT rates.
`lambdaS`	a scalar, a vector, a matrix or an array containing the external reorganization energies (**in eV**) used to calculate the CT rates.
`hBarW`	a scalar giving the energy of the effective vibrational mode assisting the charge transfer process (**in eV**) used to calculate the CT rates.

dE0	a scalar, a vector, a matrix or an array containing the site energy differences (**in eV**) used to calculate the CT rates. By default self-exchange reactions are considered (dE0=0).
dEField	a scalar, a vector, a matrix or an array containing an additional contribution to the site energy differences due to an external electric field (**in eV**). By default no electric field is applied (dEField=0).
temp	a scalar giving the temperature (**in Kelvin**) at which to evalute the CT rates. By default CT rates are evaluated at room temperature (temp=300).
nVib	an integer indicating the number of vibrational states used to compute the Franck-Condon factor. By default 50 vibrational states are considered.

Details

The arguments of these function can be scalars, vectors, matrices or arrays. Mixing scalar values with vectors, matrices or arrays is allowed but in all other cases the arguments must have the same dimensions and lengths. Using matrices or arrays is usefull to compute simultaneously several charge transfer rates for different pairs of molecules, structures ...

Value

Depending on the dimension of the objects passed to the function a scalar, a vector, a matrix or an array containing the Marcus-Levich-Jortner CT rates (**in s-1**) is returned.

References

R.A. Marcus, *Journal of Chemical Physics*, 24:966, **1956**

R.A. Marcus, *Reviews of Modern Physics*, 65:599, **1993**

S.A. Rice and M.T. Vala, *Journal of Chemical Physics*, 42:73, **1965**

See Also

energyConversion, dEField, Marcus, LandauZener, KMC

Examples

```
## Produce a map of the decimal logarithm of the Marcus,
## Marcus-Levich-Jortner and Landau-Zener rate expressions for:
nuN <- 1445 # effective vibrational mode wavenumber in cm-1
lambdaI <- 0.14  # internal reorganization energy in eV
lambdaS <- 36E-3 # external reorganization energy in eV

N  <- 301
J  <- seq( 0  , 65,length.out=N)*1E-3 # eV
dE <- seq(-0.5,0.5,length.out=N)      # eV
G  <- expand.grid(J, dE)
J  <- G[,1]
dE <- G[,2]
```

```
kMLJ    <- MarcusLevichJortner(
           J = J, lambdaI = lambdaI, lambdaS = lambdaS,
           hBarW = centimeterMinusOne2electronVolt(nuN), dE0 = dE)
kMarcus <- Marcus(
           J = J, lambda = lambdaI+lambdaS, dE0 = dE)
kLZ     <- LandauZener(
           J = J, lambda = lambdaI+lambdaS,
           nuN = centimeterMinusOne2Hertz(nuN), dE0 = dE)

kMLJ    <- matrix(kMLJ   , nrow = N, ncol = N)
kMarcus <- matrix(kMarcus, nrow = N, ncol = N)
kLZ     <- matrix(kLZ    , nrow = N, ncol = N)

addAxis <- function(bottom = TRUE, left = FALSE, above = FALSE, right = FALSE){
  useless <- lapply(1:4,axis, labels=FALSE)
  if(bottom) axis(1, labels = TRUE)
  if(left  ) axis(2, labels = TRUE)
  if(above ) axis(3, labels = TRUE)
  if(right ) axis(4, labels = TRUE)
  if(bottom) mtext(side=1,line=1.2, text=expression( abs(J)/eV), cex=par("cex"))
  if(left  ) mtext(side=2,line=1.2, text=expression(Delta*E/eV), cex=par("cex"))
  if(right ) mtext(side=4,line=1.2, text=expression(Delta*E/eV), cex=par("cex"))
  box()
}

layout(matrix(1:3, ncol=3))
par(cex=2, lwd=1.5, pty="s", mgp=c(1.1,0.1,0), tck=0.02, mar=rep(0.7,4), oma=rep(2,4))
contour(unique(J), unique(dE), log10(kMLJ   ),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, TRUE, FALSE, FALSE)
title("Marcus-Levich-Jortner", line=1)
contour(unique(J), unique(dE), log10(kMarcus),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, FALSE, FALSE, FALSE)
title("Marcus", line=1)
contour(unique(J), unique(dE), log10(kLZ    ),
        zlim = c(1,15), levels = -15:15, xaxt="n", yaxt="n", labcex=3)
addAxis(TRUE, FALSE, FALSE, TRUE)
title("Landau-Zener", line=1)
```

`universalConstants` *universal Constants*

Description

This data set provides various universal constants

Usage

```
universalConstants
```

Format

A data frame containing for each universal constant the following information.

`Quantity` a character string describing the constant.

`Value` the value of the constant (numerical).

`Unit` a character string indicating the unit of the constant.

Constants such as the speed of light, the Planck or Boltzmann constants are available.

Source

http://www.ebyte.it/library/educards/constants/ConstantsOfPhysicsAndMath.html

Examples

```
# List all the constants
print(universalConstants)

# Data for the speed of light
universalConstants["c",]

# Return the speed of light in m.s-1
universalConstants["c","Value"]

# Return the Planck constant in J.s
universalConstants["h","Value"]

# Return the Planck constant in eV.s
Joule2electronVolt(universalConstants["h","Value"])
```

Index