Package 'SimEvolEnzCons'

October 28, 2021

Type Package

Title Simulation of Evolution of Enzyme Under Constraints

Version 2.0.0

Date 2021-10-28

Author Charlotte Coton [aut, cre] (<https://orcid.org/0000-0002-5371-0327>),
Gregoire Talbot [aut],
Maud Le Louarn [aut],
Christine Dillmann [aut, ths] (<https://orcid.org/0000-0002-9025-341X>),
Dominique de Vienne [aut, ths]
(<https://orcid.org/0000-0003-3395-7901>)

Maintainer Charlotte Coton <charlotte.coton@universite-paris-saclay.fr>

Description Simulate the evolution of enzyme concentrations under selection for increased flux in a metabolic pathway, with cellular constraints.
Create graphics for the simulation results.
Compute evolutionary equilibrium and Range of Neutral Variations of enzyme concentrations.
This package is part of ``Coton, C., Talbot, G., Le Louarn, M., Dillmann, C., de Vienne, D. (2021) <bioRxiv:10.1101/2021.05.04.442631>".
Version 2.0.0 and more takes account of regulation groups, and is part of a second article ``Coton, C., Dillmann, C., de Vienne, D. (in progress)".

Depends R (>= 3.1.0)

Imports stats, rgl, ade4, graphics, grDevices, RColorBrewer, scatterplot3d

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-10-28 17:30:02 UTC

R topics documented:

`activities` *Activity computation*

Description

Computes the enzyme pseudo-activities

Usage

```
activities(kin_fun,Keq_fun)
```

Arguments

`kin_fun`	Numeric vector of kinetic parameters (catalytic constant kcat divided by Michaelis constant Km)
`Keq_fun`	Numeric vector of equilibrium constants

Details

Computes the pseudo-activities of enzymes, i.e. for each enzyme, the product of its kcat/Km (kinetic parameters) by the product of the upstream reactions equilibrium constants.

In other functions, pseudo-activity is also named activity.

`kin_fun` and `Keq_fun` need to have the same length.

Value

Numeric vector of activities of each enzyme, of same length as `kin_fun` and `Keq_fun`.

References

Lion, S., F. Gabriel, B. Bost, J. Fiévet, C. Dillmann, and D. De Vienne, 2004. An extension to the metabolic control theory taking into account correlations between enzyme concentrations. European Journal of Biochemistry 271:4375–4391.

Examples

```
#Values from CoA metabolism
kin <- c(53/0.29,50/0.78,29,6.22) #kinetic parameters kcat/Km
Keq <- c(1.1e+8,4.9e+3,1.1e+3,0.228) #equilibrium constants
A <- activities(kin,Keq) #activities

# results : A = c(1.827586e+02, 7.051282e+09, 1.563100e+13, 3.687838e+15)
```

---

alpha_ij *Redistribution coefficient computation*

Description

Computes the redistribution coefficients between all enzymes

Usage

```
alpha_ij(E_fun,correl_fun,beta_fun=NULL)
```

Arguments

E_fun	Numeric vector of concentrations
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
beta_fun	Matrix of co-regulation coefficients

Details

The redistribution coefficients are influenced by the applied constraint.

For further details on how the redistribution coefficient is calculated, see Lion *et al.* (2004).

For cases with co-regulations (i.e. correl_fun value is "RegPos", "RegNeg", "CRPos" or "CRNeg"), beta_fun is obligatory. In other cases (i.e. correl_fun value is "SC" or "Comp"), beta_fun is ignored, that is why default is NULL.

When beta_fun is obligatory, it has to be a square matrix of size n, where this number n is the length of E_fun.

Value

Square matrix of size n (where n is the length of E_fun, which is the number of enzymes) of the redistribution coefficients between all enzymes.

References

Lion, S., F. Gabriel, B. Bost, J. Fiévet, C. Dillmann, and D. De Vienne, 2004. An extension to the metabolic control theory taking into account correlations between enzyme concentrations. European Journal of Biochemistry 271:4375–4391.

See Also

See function `is.correl.authorized` to have more information about constraints and on the usage of argument `correl_fun`.

Examples

```
E <- c(30,30,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
correl <- "SC"

alph <- alpha_ij(E,correl,beta)
```

`apparent.activities.Aq`

Apparent activity computation

Description

Computes the group apparent pseudo-activities at equilibrium

Usage

```
apparent.activities.Aq(phi_q,A_fun,B_fun,correl_fun,eiq_eff=NULL)
```

Arguments

`phi_q`	Numeric vector containing the numbers of enzymes in the group (ideally, element of the list returns by `class_group`)
`A_fun`	Numeric vector of activities
`B_fun`	Numeric vector of global co-regulation coefficients. Same length as `E_ini_fun`.
`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system
`eiq_eff`	Numeric vector of the intra-group relative concentrations at effective equilibrium.

Details

Computes the apparent pseudo-activities of regulation groups at theoretical equilibrium (no competition) or effective equilibrium (competition).

When there is competition, `eiq_eff` is needed.

If there is only one regulation group (all enzymes are co-regulated), apparent activity of the group is not useful. When the enzyme is in a singleton (enzyme independent from all other enzymes), the apparent activity of the group is identical to the activity of the enzyme.

In other functions, pseudo-activity is also named activity.

Value

Numeric value of the apparent activity of chosen group.

See Also

Function class_group to classify enzymes in groups.

Examples

```
#Two groups
n <- 3
A <- c(1,10,30)
beta <- diag(1,n)
beta[1,2] <- -0.32
beta[2,1] <- 1/beta[1,2]
B <- compute.B.from.beta(beta)

L_Phi <- class_group(beta)

correl <- "RegNeg"
#apply function of all groups
A_app <- unlist(lapply(L_Phi,apparent.activities.Aq,A,B,correl))
```

---

class_group *Classify enzymes in regulation groups*

Description

Classify the enzymes in their corresponding regulation group from the co-regulation matrix

Usage

```
class_group(beta_fun)
```

Arguments

beta_fun	Matrix of co-regulation coefficients

Details

Enzymes are classified in regulation groups depending on the co-regulation matrix. If the co-regulation coefficient is equal to zero between enzymes, the enzymes are in different regulation group, else there are in the same group.

Enzyme 1 is in the regulation group 1. Other enzymes are classified in ascending order.

Value

Return a list classically named `L_Phi` of length `p` (the number of regulation groups). Each element `q` of `L_Phi` (the regulation group Φ_q) contains the numbers of co-regulated enzymes.

See Also

Function `search_group` to find the group of an enzyme.

Examples

```
#Only one group
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
L_Phi <- class_group(beta)

#total number of regulation groups
p <- length(L_Phi) #gives 1


#Two groups
n <- 3
beta <- diag(1,n)
beta[1,2] <- -0.32
beta[2,1] <- 1/beta[1,2]

L_Phi <- class_group(beta)
p <- length(L_Phi) #gives 2


#with data
data(data_sim_RegNeg_1grpNeg1grpPos)
class_group(data_sim_RegNeg_1grpNeg1grpPos$param$beta)
```

---

coef_rep *Response coefficient computation*

Description

Computes the response coefficients for each enzyme

Usage

```
coef_rep(E_fun,A_fun,correl_fun,beta_fun=NULL)
```

Arguments

E_fun	Numeric vector of concentrations
A_fun	Numeric vector of activities
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
beta_fun	Matrix of co-regulation coefficients

Details

The response coefficients are influenced by the applied constraint.

Response coefficients are an extension of the control coefficients defined in the Metabolic Control Theory (Kacser and Burns, 1973). For further details on how the response coefficient is calculated, see Lion *et al.* (2004).

A_fun and E_fun need to have the same length. The restrictions on beta_fun are explained in alpha_ij.

Value

Numeric vector of the response coefficients of each enzyme

References

Lion, S., F. Gabriel, B. Bost, J. Fiévet, C. Dillmann, and D. De Vienne, 2004. An extension to the metabolic control theory taking into account correlations between enzyme concentrations. European Journal of Biochemistry 271:4375–4391.

See Also

Use function activities to compute enzyme activities.

Examples

```
A <- c(1,10,30)
E <- c(30,30,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
correl <- "SC"

response <- coef_rep(E,A,correl,beta)


# result : response = c(0.88235294, 0.08823529, 0.02941176)
```

coef_sel.continue *Selection coefficient computation*

Description

Computes the selection coefficient using the continuous expression $s_i = R_i * \delta_i / E_i$

Usage

```
coef_sel.continue(i_fun,E_res,A_fun,delta_fun,correl_fun,beta_fun=NULL)
```

Arguments

i_fun	Integer number indicating the enzyme targeted by the mutation. *See details*
E_res	Numeric vector of resident enzyme concentrations
A_fun	Numeric vector of activities
delta_fun	Numeric. Actual effect of a mutation targeting enzyme i_fun, i.e. δ_i. *See details*
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
beta_fun	Matrix of co-regulation coefficients

Details

Computes the selection coefficient using a continuous expression $s_i = R_i * \delta_i / E_i$

Only mutations of concentrations are been considered.

i_fun is the number of the enzyme targeted by the mutation. It is an integer number between 0 and n, which is the total number of enzyme in the pathway. If i_fun is between 1 and n, delta_fun needs to be a single value and function coef_sel.continue computes the selection coefficient of a mutation of actual effect delta_fun targeting i_fun. If i_fun is set to 0, delta_fun needs to be a vector of same length as E_res. Each value of delta_fun is the actual effect of the mutation, and the position of this value in the vector is the target enzyme number. Thus, to see the effect of a mutation of given actual effect on every enzyme, set i_fun to 0 and delta_fun has to be a vector of same length as E_res.

Value

Numeric value of the selection coefficient for the target enzyme.

If i_fun is set to 0, returns the numeric vector of the selection coefficients for the different enzyme.

References

Coton et al. (2021)

See Also

Use function activities to compute enzyme activities.

Examples

```
#### Set
A <- c(1,10,30)
E <- c(30,30,30)
correl <- "CRPos"
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)

### Mutation
mu <- 1
i <- 3
delta <- compute.delta(mu,E,correl,B)

#for enzyme i
coef_sel.continue(i,E,A,delta[i],correl,beta)

#for all enzyme
coef_sel.continue(0,E,A,delta,correl,beta)
```

---

coef_sel.discrete *Selection coefficient computation*

Description

Computes the selection coefficient using the discrete expression $s = (w^m - w^r)/w^r$

Usage

```
coef_sel.discrete(E_res, A_res, E_mut=NULL, A_mut=NULL)
```

Arguments

E_res	Numeric vector of concentrations for the resident
A_res	Numeric vector of activities for the resident. Default value NULL corresponds to no mutation, *i.e.* same value as E_res
E_mut	Numeric vector of concentrations for the mutant
A_mut	Numeric vector of activities for the mutant. Default value NULL corresponds to no mutation, *i.e.* same value as A_res

Details

Computes the selection coefficient between mutant and resident using the discrete expression $s = (w^m - w^r)/w^r$, assuming that fitness is proportional to flux. Here, function `flux` is used to compute the flux, and therefore, the fitness.

All input vectors need to have the same length.

If there is no mutation affecting concentrations (resp. activities), mutant concentrations (resp. activities) are identical to resident one. In this case, give to `E_mut` (resp. `A_fun`) the same value as `E_res` (resp. `A_res`), or put the default value `NULL`.

Value

Numeric value for selection coefficient

See Also

Use function `activities` to compute enzyme activities.

Examples

```
### Mutation of E
A <- c(1,10,30)
E <- c(30,30,30)
Em <- mut.E.direct(E,1,1,"SC")

coef_sel.discrete(E,A,Em)


### Mutation of A
E <- c(30,30,30)
kin <- c(1,10,1000)
Keq <- c(10,1,100)
A <- activities(kin,Keq)
kin_m <- mut.kin(kin,3,1)
Am <- activities(kin_m,Keq) #equilibrium constant cannot be modified by mutation

coef_sel.discrete(E,A,E,Am)
```

---

compute.B.from.beta *Global co-regulation coefficient computation*

Description

Computes the global co-regulation coefficients `B` from a matrix of co-regulation coefficients `beta`

Usage

```
compute.B.from.beta(beta_fun)
```

Arguments

beta_fun	Matrix of co-regulation coefficients

Details

beta_fun have same number of rows and columns.

Value

Numeric vector of the n global co-regulation coefficients.

If beta_fun is NULL, compute.B.from.beta returns NULL.

See Also

Use function is.beta.accurate to verify beta_fun conformity.

Examples

```
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
correl <- "RegPos"

is.beta.accurate(beta,3,correl)

B <- compute.B.from.beta(beta)
```

---

compute.beta.from.B *Co-regulation coefficient computation*

Description

Computes the matrix of co-regulation coefficients M_β from a vector of global co-regulation coefficients B

Usage

```
compute.beta.from.B(B_fun,L_Phi_fun=NULL)
```

Arguments

B_fun	Numeric vector of global co-regulation coefficients
L_Phi_fun	List of regulation groups. Default is NULL.

Details

Default value L_Phi_fun = NULL is appropriate only if enzymes are supposed to be all co-regulated, i.e. no β value is null. Return the same result as as a list of one element such as L_Phi_fun = list(1:n), where n is the number of enzymes, which is also the length of B_fun.

L_Phi_fun must be a list of p elements (the number of regulation groups) containing numbers between 1 and n (number of enzymes), where each list element contains the numbers of the enzyme in the group. Each enzyme only occurs in one group. See function class_group to have an idea of L_Phi_fun structure. For compute.beta.from.B, independent enzymes can be not contained in L_Phi_fun, and thus, L_Phi_fun may be smaller than B_fun.

Value

Numeric matrix n*n of the co-regulation coefficients, where n is the length of B_fun.

If beta_fun is NULL, compute.beta.from.B returns NULL.

See Also

Use function is.B.accurate to verify B_fun conformity.

See function class_group to know format of L_Phi_fun.

Examples

```
B <- 1/c(0.5,0.2,0.3)
correl <- "RegPos"

is.B.accurate(B,3,correl)

beta <- compute.beta.from.B(B)

#Seven enzymes and three groups
n <- 7
p <- 3
B <- c(1.1824,  3.695, -8.593023,  1.3, 13,  6.5, 1)
L_Phi <- list(1:3) #firt three enzymes in first group
L_Phi[[2]] <- 4:6
L_Phi[[3]] <- 7 #last enzyme independent

beta <- compute.beta.from.B(B,L_Phi)
```

---

compute.delta *Computation of mutation actual effect*

Description

Computes the actual effect δ of a mutation

Usage

```
compute.delta(nu_fun,E_fun,correl_fun,B_fun=NULL)
```

Arguments

nu_fun	Numeric. Canonical effect of the mutation
E_fun	Numeric vector of concentrations
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
B_fun	Numeric vector of global co-regulation coefficients

Details

Computes the actual effect δ of a mutation depending on its canonical effect ν and on the constraints.

Value

Numeric vector. Each element `i` of the vector is the actual effect δ_i for which enzyme `i` in `E_fun` is targeted by the mutation.

See Also

Use function `is.correl.authorized` to see allowed constraints for `correl_fun`.

Examples

```
mu <- 1
E <- c(30,30,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)
correl <- "RegPos"

compute.delta(mu,E,correl,B)
```

---

data_sim *Simulation results for each constraint*

Description

Six different datasets containing the simulation results for each constraint.

Usage

```
data("data_sim_SC")

data("data_sim_Comp")

data("data_sim_RegPos")

data("data_sim_RegNeg")

data("data_sim_CRPos")

data("data_sim_CRNeg")

data("data_sim_CRNeg_1grpNeg1sgl")

data("data_sim_RegNeg_1grpNeg1grpPos")
```

Format

Each dataset is a list containing six elements:

`$tabR` Dataframe of `nsim*npt` rows and `(3*n+4)` columns. Column `$sim` is the number of current simulation. Each row corresponds to the state at each observation step (i.e. after `pasobs` mutations), and columns are respectively concentrations (`E1` to `En`), kinetic parameters (`kin1` to `kin_n`), total concentration, total kinetic, flux, activities (`A1` to `An`), and simulation number (column `$sim`)

`$tabP_e` Numeric matrix of `npt` rows and `n+1` columns, corresponding to relative concentrations at equilibrium (column 1 to `n`) at each observation step (in rows), and associated simulation number (column `$sim`)

`$tabP_r` Same as `$tabP_e`, but for response coefficients

`$list_init` List of 3 elements, containing initial values of concentrations in `$E0`, kinetic parameters in `$kin0` and activities in `$A0` for each simulation. Each element is a numeric matrix of `nsim` rows and `n` columns

`$list_final` List of 3 elements, containing final values of concentrations in `$E_f`, kinetic parameters in `$kin_f` and activities in `$A_f` for each simulation. Each element is a numeric matrix of `nsim` rows and `n` columns

`$param` List of input parameters:

- `n`: number of enzymes,
- `nsim`: number of simulation,
- `E0`: matrix of initial concentrations, identical to `$list_init$E0`,
- `kin0`: matrix of initial kinetic parameters, identical to `$list_init$kin0`,
- `Keq`: numeric vector of constant equilibrium,
- `beta`: matrix of co-regulation coefficients,
- `B`: numeric vector of global co-regulation coefficients,
- `correl`: character string indicating the constraint abbreviation,

- N: population size,
- pasobs: number of steps between two system observations,
- npt: number of system observations,
- X: parameter for flux computation,
- Etot0: initial total concentration,
- pmutA: probability for activity mutation,
- other parameters are described in simul.evol.enz.multiple

Details

Possible constraints are listed below:

- "SC": independence between all enzymes
- "Comp": competition for resources
- "RegPos": positive regulation
- "RegNeg": negative regulation
- "CRPos": competition plus positive regulation
- "CRNeg": competition plus negative regulation

There is ten simulations by constraint.

Simulations differ by the initial concentrations (manually chosen), but initial concentrations are identical between constraints.

Chosen equilibrium for tabP_e and tabP_r are the theoretical equilibrium for constraints "SC", "Comp" and "RegPos", and the effective one for constraints "RegNeg", "CRPos" and "CRNeg".

These simulation results are exploited in Coton et al. (2021).

New datasets when there are regulation groups

"data_sim_CRNeg_1grpNeg1sgl" contains simulation results when there are one negative group and one singleton with competition.

"data_sim_RegNeg_1grpNeg1grpPos" contains simulation results when there are one negative and one positive groups, without competition.

Source

Function simul.evol.enz.multiple have been used to obtained these datasets. Input parameters are listed in data_sim_xx$param (where xx is the constraint abbreviation). E0 have been randomly chosen. Seed have been set to 1 before the simulations for the first constraint.

References

Coton at al. (2021)

droites *Line of relative enzyme concentrations (co-regulations cases)*

Description

Computes the position of the point of relative concentrations on the line on which they move on in the cases of co-regulations

Usage

```
droite_e(tau_fun,E_ini_fun,B_fun)

droite_E.Reg(tau_fun,E_ini_fun,B_fun)

droite_E.CR(tau_fun,E_ini_fun,B_fun)

droite_tau(E_fun,E_ini_fun,B_fun)
```

Arguments

tau_fun	Numeric value of the position of relative enzyme concentrations on the line
E_ini_fun	Numeric vector of initial concentrations
B_fun	Numeric vector of global co-regulation coefficients. Same length as E_ini_fun.
E_fun	numeric vector of current concentrations on the line. Same length as E_ini_fun.

Details

In the cases of co-regulations, relative enzymes concentrations evolve along a straight line. This line is determined by to factors: initial enzyme concentrations and global co-regulation factors. The driving variable τ is a parameter indicating the position of the relative enzyme concentrations e on this line.

Function droite_e gives the relative concentrations corresponding to the input position tau_fun.

Function droite_E.Reg gives the absolute concentrations corresponding to the input position tau_fun in case of **regulation only** (constraints abbreviation "RegPos" or "RegNeg").

Function droite_E.CR gives the absolute concentrations corresponding to the input position tau_fun in case of **competition and regulation** (constraints abbreviation "CRPos" or "CRNeg").

Function droite_tau gives the position τ corresponding to the input enzyme concentrations E_fun.

Note that if initial relative concentrations E_ini_fun is a multiple of 1/B_fun, the line becomes a point and position τ does not exist.

Value

droite_e returns a numeric vector of relative concentrations

droite_E.Reg returns a numeric vector of absolute concentrations

droite_E.CR returns a numeric vector of absolute concentrations

droite_tau returns a numeric value giving the position on the line

Special results

Initial point

If tau_fun is equal to 0, droite_e returns the value of initial relative concentrations, *i.e.* value of E_ini_fun divided by sum(E_ini_fun); droite_E.Reg and droite_E.CR returns the value of E_ini_fun.

If E_fun is a multiple of E_ini_fun, function droite_tau returns 0 *(initial point)*.

End point

If tau_fun is equal to 1, droite_e returns the reverse value of B_fun; droite_E.CR returns a multiple of the reverse value of B_fun; droite_E.Reg returns Inf.

If E_fun is a multiple of 1/B_fun, function droite_tau returns 1 *(end point)*.

Line becomes a point

If E_ini_fun is a multiple of 1/B_fun,droite_e returns the value of 1/B_fun; droite_E.Reg returns an error, because concentrations E can be any multiple of 1/B_fun without variation of relative concentrations; droite_E.CR returns E_ini_fun; droite_tau returns an error because τ does not exist in this case.

Line does not exist

If there only one enzyme (length of E_ini_fun is equal to 1), relative concentrations is always equal to 1. droite_e should return 1; droite_E.Reg, droite_E.CR and droite_tau should return an error.

See Also

To compute global co-regulation coefficients B_fun from co-regulation matrix beta_fun, see the example or use function compute.B.from.beta.

Examples

```
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)
E0 <- c(30,30,30)
tau <- 0.5

droite_e(tau,E0,B)

E <- droite_E.Reg(tau,E0,B)
droite_tau(E,E0,B)

E <- droite_E.CR(tau,E0,B)
droite_tau(E,E0,B)
```

`extract.tabEtot` *Table of concentrations*

Description

Creates a table of all absolute concentrations from simulation results, or any table of enzyme concentrations (enzyme by column)

Usage

```
extract.tabEtot(tabR,beta_fun)
```

Arguments

`tabR`	Dataframe of simulation results, which is an output of `simul.evol.enz.multiple`
`beta_fun`	Matrix of co-regulation coefficients

Details

From the result table `tabR` of simulations, extract the values of absolute concentrations of each enzymes, and computes the absolute concentrations E^q in each regulation and the total concentration at each time step.

The input `tabR` is the output `tabR` of function `simul.evol.enz.multiple`, or possibly the output `res_sim` of function `simul.evol.enz.multiple`.

The function also works for any table `tabR` that has n columns (enzyme concentrations in columns), and any number of rows. The last column `"sim"` is not an obligation, but if present, need to be the last column.

Value

Dataframe of $n + p + 2$ columns, where n is the number of enzymes and p is the number of regulation groups. The n first columns are the concentrations of enzymes 1 to n (named `E1,...,En`), the following p are the sum of concentrations in regulations groups from group 1 to p (named `Ep1,...,Epp`), the second last column is the total concentration (named `Etot`) and the last column is the simulation number (named `sim`). The column are named: `E1,...,En,Eq1,... Eqp,Etot,sim`. The row number of the output dataframe is the same as `tabR`. The last column `"sim"` is present only if input `tabR` has a column named `"sim"`.

See Also

Use function `simul.evol.enz.multiple` to run simulations.

Examples

```
#one group
n <- 3
beta <- diag(1,n)
beta[1,2] <- 0.1 ; beta[2,1] <- 1/beta[1,2]
some.E <- matrix(30,ncol=n,nrow=3)
some.E[2,] <- c(20,50,10)
some.E[3,] <- runif(n,1,100)
tabEtot <- extract.tabEtot(some.E,beta)


#With saved simulation
data(data_sim_RegPos)
tabEtot <- extract.tabEtot(data_sim_RegPos$tabR,data_sim_RegPos$param$beta)
```

`flux` *Flux computation*

Description

Computes the flux of a metabolic pathway

Usage

```
flux(E_fun,A_fun,X_fun=1)
```

Arguments

`E_fun`	Numeric vector of concentrations
`A_fun`	Numeric vector of activities
`X_fun`	Numeric value. Default is `1`

Details

Computes the flux of a metabolic pathway according to the Metabolic Control Theory (Kacser and Burns, 1973).

Value

`flux` returns a numeric value

References

Kacser, H. and J. A. Burns, 1973. The control of flux. Symp. Soc. Exp. Biol. 27:65–104.

Kacser, H., J. A. Burns, H. Kacser, and D. A. Fell, 1995. The control of flux : 21 years on. Biochemical Society Transactions 23:341–366.

See Also

Use function activities to compute enzyme activities.

Examples

```
E <- c(1,10,30)
A <- c(30,30,30)
J <- flux(E,A)

#result : J = 26.47059
```

`flux.dome.graphics` *Plot of the flux dome*

Description

Function `flux.dome.graph3D` gives a 3D graph of flux dome for a three-enzyme pathway.

Function `flux.dome.projections` gives a triangular diagram of flux dome, and if `add.reg==TRUE`, it also gives a plot of the curve defined by the intersection of the dome with the upright plan drawn from regulation line.

Usage

```
flux.dome.graph3D(A_fun,Etot_fun,add.reg=FALSE,B_fun=NULL,
E_ini_fun=NULL,X_fun=1,marge_section=0.1,marge_top.dome=0.5)

flux.dome.projections(A_fun,Etot_fun,add.reg=FALSE,B_fun=NULL,
E_ini_fun=NULL,X_fun=1,nbniv=9,niv.palette=NULL,marge_section=0.1,
posi.legend="topleft",new.window=FALSE,cex.pt=1.5,...)
```

Arguments

`A_fun`	Numeric vector of activities
`Etot_fun`	Numeric value of total concentration
`add.reg`	Logical. Add regulation line in plots? Default is `FALSE`.
`B_fun`	Numeric vector of global co-regulation coefficients. Same length as `E_ini_fun`.
`E_ini_fun`	Numeric vector of initial concentrations
`X_fun`	Numeric value. Default is `1`
`marge_section`	Numeric. Height of section up to surface dome. Default is `0.1`
`marge_top.dome`	Numeric. Space between top dome and max of zlim. Default is `0.5`
`nbniv`	Numeric. Number of contour lines, between 3 and 11. Default is `9`
`niv.palette`	Character vector. Color palette to be passed to contour lines.

posi.legend	Contour line legend position. See *details*.
new.window	Logical. Does graphics appear in a new window ? Default is TRUE
cex.pt	Numeric. Size of points in plot(s) drawn by flux.dome.projections
...	Arguments to be passed to plot function, such as lwd or cex.axis

Details

General

***Available only for three-enzyme pathway**, i.e. length of* A_fun *equal to 3.*

Flux dome exists only in case of the competition.

E_ini_fun is rescaled by a cross product to have sum of E_ini_fun equal to Etot_fun.

Only flux dome is returned if add.reg is FALSE.

If add.reg=TRUE, E_ini_fun and B_fun are required. Else, default is NULL.

Function flux.dome.graph.3D

If add.reg is TRUE, a section corresponding to the upright plan drawn from line on which the relative concentrations move when there is co-regulations *(see* droites*)* is added on graph.

Function flux.dome.projections

Projection on plan (e1,e2,e3) is from blue for low flux to red for high flux. Colors are taken from Spectral palette of the package 'RColorBrewer'.

Because contour lines are calibrated to correspond to integer values of flux, it is possible to have a difference between input and output nbniv.

If add.reg is TRUE, line on which the relative concentrations move when there is co-regulations *(see* droites*)* is added on triangular diagram. Also, a new plot of the curve defined by the intersection of the dome with the upright plan drawn from this line is drawn.

Color palette for contour lines

By default, color palette for contour lines is taken in palette "Spectral" of package RColorBrewer, with cool colors for low flux values and warm colors for high ones.

Input your own color palette in niv.palette.

If color numbers in niv.palette is inferior to the number of contour lines nbniv, a warning message is printed and palette is replicated to have enough colors.

posi.legend is the position of the legend for contour line. It is a character vector of length 1 ("topleft", etc.) or a numeric vector of length 2, which is the coordinates of the upper left corner of the legend box. If NULL, the legend will not be shown. The first (resp. second) element of posi.legend is passed to argument x (resp. y). Note that the minimum and maximum coordinates for the return triangular plot are between -0.864 and 0.864 for x-axis, and -0.66 and 1.064 for y-axis. Use locator(1) to adjust position legend.

Value

Function flux.dome.graph3D returns a 3D-graph of flux dome, with section in case of regulation.

Function flux.dome.projections returns a triangular diagram corresponding to the projection of the flux dome on the plan of relative concentrations. In the case of regulation (add.reg=TRUE), a straight line is drawn. If add.reg=TRUE, function flux.dome.projections also returns a plot of the curve defined by the intersection of the dome with the upright plan drawn from regulation line.

See Also

See function droites to compute regulation line.

Examples

```
Etot <- 100
A <- c(1,10,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- compute.B.from.beta(beta)

#or for B :
B <- 1/c(0.2,0.5,0.3)
E0<- c(30,30,30)

flux.dome.graph3D(A,Etot,add.reg=TRUE,B,E0)
flux.dome.projections(A,Etot,add.reg=TRUE,B,E0)


#Position of legend at right
flux.dome.projections(c(1,10,30),100,posilegend=c(0.55,0.9))
```

---

flux.shape.for.all.points

Flux shape computing for all points

Description

flux.shape.for.all.points computes flux and selection coefficient for various points (i.ei. vector of concentrations), giving the flux shape

Usage

```
flux.shape.for.all.points(Etot_fun, A_fun, nu_fun, correl_fun, beta_fun=NULL,
 E_ini_fun=NULL, n_fun=3)
```

Arguments

Etot_fun	Numeric. The total concentration
A_fun	Numeric vector of activities
nu_fun	Numeric value of **canonical** mutation effect
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
beta_fun	Matrix of co-regulation coefficients
E_ini_fun	Numeric vector corresponding to initial concentrations.
n_fun	Numeric. Number of enzymes. Necessarily equal to 3 for this function.

Details

Every enzyme correspond to one dimension in a n-dimensional graph.

For various concentration vectors, this function computes flux and selection coefficient.

Selection coefficient are computed from two different expressions (discrete by `coef_sel.discrete` and continuous by `coef_sel.continue`), for a mutation of canonical size `nu_fun`.

`E_ini_fun` is rescaled by a cross product to have sum of `E_ini_fun` equal to `Etot_fun`.

Value

Invisible list of 3 elements:

- `$J` : Numeric vector of flux.
- `$sel_disc` : Numeric matrix of `n_fun` columns corresponding to discrete selection coefficient, and each row corresponds to one point (i.e. concentration vector). Each column correspond to the "mutated" enzyme.
- `$sel_cont` :Same as `$sel_disc`, but for continuous selection coefficient. Same properties.

See Also

To study shape of flux only from a certain point and for any number of enzymes, see `flux.shape.from.one.point`

`flux.shape.from.one.point` *Flux shape computing from one point*

Description

`flux.shape.from.one.point` computes flux in every dimension (corresponding to each enzyme) from a given point (vector of concentrations)

Usage

```
flux.shape.from.one.point(Etot_fun, A_fun, correl_fun, beta_fun=NULL,
E_ini_fun=NULL, from.eq=TRUE, E_fun=NULL, X_fun=1, with.alpha=FALSE, grp.reg=FALSE)
```

Arguments

`Etot_fun`	Numeric. The total concentration
`A_fun`	Numeric vector of activities
`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system
`beta_fun`	Matrix of co-regulation coefficients
`E_ini_fun`	Numeric vector corresponding to initial concentrations.

from.eq	Logical. Is the analyzed point is the equilibrium point ? If TRUE, flux and selection coefficients are computed from the equilibrium, else E_fun is required. Default is TRUE.
E_fun	Numeric vector of the concentrations at analysed point. If from.eq=TRUE, E_fun is ignored.
X_fun	Numeric value. Default is 1
with.alpha	Logical. For case CR, is the computing method use alpha formula (TRUE) or pass by tau computing (FALSE) ?
grp.reg	Logical. Is there is some regulation groups in beta matrix ? If TRUE, tau will not be computed and give 0.

Details

Every enzyme correspond to one dimension in a n-dimensional graph.

From a given resident point E_fun, each value on dimension i is considered as a possible mutant of enzyme concentration E_i. Every "mutants" are taken between 0 and Etot_fun by 0.01 step. In every dimension, function flux.shape.from.one.point computes flux and selection coefficient (discrete coef_sel.discrete and continuous coef_sel.continue) from this point.

E_fun (resp. E_ini_fun) is rescaled by a cross product to have sum of E_fun (resp. E_ini_fun) equal to Etot_fun.

If from.eq=TRUE, analyzed point is:

- the theoretical equilibrium in case of independence "SC" and competition only "Comp";
- near the theoretical equilibrium (tau=0.95) in case of positive regulation only "RegPos" (due to infinite possible values for concentrations at this point);
- the effective equilibrium in other cases.

Default of E_ini_fun is NULL and corresponds to correl_fun equal to "SC" or "Comp", but in other cases (due to presence of regulation, E_ini_fun is obligatory and needs to have the same length as A_fun).

Value

Invisible list of 6 elements:

- $x : Numeric vector of all values that mutated enzymes can take, between 0 and Etot_fun, by 0.01. Length of (Etot_fun-0)*100.
- $J : Numeric matrix of n columns and (Etot_fun-0)*100 rows. Each column correspond to one direction (*i.e. which enzyme is "mutated"*) and each row to each value of flux in this direction.
- $sel_disc : Numeric matrix corresponding to discrete selection coefficient. Same properties.
- $sel_cont : Numeric matrix corresponding to continuous selection coefficient. Same properties.
- $tau : Numeric matrix corresponding to position τ in case of regulation. Same properties.
- $param : List of input parameters

See Also

To understand differences between discrete and continuous selection coefficients, see function `coef_sel.discrete` and `coef_sel.continue`.

Examples

```
fsfop <- flux.shape.from.one.point(100,c(1,10,30),"SC")
```

`flux.shape.from.one.point.graphics`

Plots of flux shape computing from one point

Description

Graphics for illustrating results of function `flux.shape.from.one.point`

Usage

```
flux.shape.from.one.point.graphics(list_fsfop, N_fun, zoom_delta=NULL,add.eq=TRUE,
gr.s.delta=TRUE,gr.J.E=FALSE,gr.s_cont.E=FALSE,gr.s_disc.E=FALSE,
gr.comp.s=FALSE,gr.tau=FALSE,posi.legend="topleft",...)
```

Arguments

`list_fsfop`	Output of the function `flux.shape.from.one.point`
`N_fun`	Numeric. Population size that influence neutral zone
`zoom_delta`	Zoom on axis of apparent mutation effects. *See details.*
`add.eq`	Logical. Does equilibrium appear on graphics?
`gr.s.delta`	Logical. Add graph of selection coefficient in relation to mutation effect?
`gr.J.E`	Logical. Add graph of flux J in relation to enzyme concentrations?
`gr.s_cont.E, gr.s_disc.E`	Logical. Add graph of selection coefficient (respectively from continuous or discrete expression) in relation to enzyme concentrations?
`gr.comp.s`	Logical. Add graph of comparison of selection coefficient from different expressions (continuous vs discrete)?
`gr.tau`	Logical. Add graph depending on tau?
`posi.legend`	Legend position for activities. See *details*.
`...`	Arguments to be passed in `plot` function, such as `lwd` or `cex`.

Details

See function `flux.shape.from.one.point`.

Gives graphics of:

1. Selection coefficient (discrete expression) in relation to apparent mutation effect δ, which is $E^m - E^r$ (option `gr.s.delta`)
2. Flux in relation to mutant concentration E^m (option `gr.J.E`)
3. Selection coefficient (continuous expression) in relation to mutant concentration (option `gr.s_cont.E`)
4. Selection coefficient (discrete expression) in relation to mutant concentration (option `gr.s_disc.E`)
5. Comparison between the two expressions of selection coefficient (option `gr.comp.s`)
6. If driving variable τ exists: flux/mutant concentrations/discrete selection coefficient in relation to tau (option `gr.tau`)

Each enzyme is represented by one color. If there are three enzymes enzymes or less, enzyme 1 is in red, 2 in green and 3 in blue. For more than three enzymes, colors are taken in palette `"Spectral"` from package '`RColorBrewer`'.

Discrete expression of selection coefficient refers to $s = (J^m - J^r)/J^r$. Available whatever values of enzymes concentrations. See function `coef_sel.discrete`. *Continuous expression* of selection coefficient refers to $s_i = R_E i^J delta_i / E_i$. Available only for small values of delta_i. See function `coef_sel.continue`.

The RNV corresponds to enzyme concentrations such as selection coefficient is between -1/N and 1/N.

Zooms

If `NULL`, default values are applied.

If vector of length 1, symmetry is applied (i.e. -zoom,zoom).

If vector of length 2, zoom values are applied directly.

posi.legend

Indicates coordinates of the upper left corner of the legend. Available only for graph corresponding to `gr.s.delta`. See more in details of function `flux.dome.projections`.

Value

Nothing

See Also

See function `RNV.ranking.order.factor` to see against which parameters the RNV depend.

Examples

```
fsfop <- flux.shape.from.one.point(100,c(1,10,30),"SC")
flux.shape.from.one.point.graphics(fsfop,1000)
```

graph.simul.conc.end *Graphics of simulation ends*

Description

Gives graphics of enzyme concentrations two-by-two at end of simulation

Usage

```
graph.simul.conc.end(all_res_sim,new.window=FALSE,add.eq=TRUE,
which.sim=NULL,which.enz=NULL,...)
```

Arguments

all_res_sim	List, the output of function simul.evol.enz.multiple (results of evolution simulation).
new.window	Logical. Do graphics appear in a new window?
add.eq	Logical. Do equilibrium appear on graph?
which.sim	Numeric vector containing integer numbers between 1 and nsim. Which simulations would you represent? If NULL (default), all simulations would be represented.
which.enz	Numeric vector containing integer numbers between 1 and n. Which enzymes would you represent? If NULL (default), all enzymes would be represented.
...	Arguments to be passed in plot function, such as lwd or cex.

Details

This function shows the concentration of one enzyme against the concentration of another enzyme, for selected enzymes.

Simulation ends are supposed to be near to equilibrium, given the population size and therefore the neutral zone.

See Also

Use function simul.evol.enz.multiple to simulate enzyme evolution.

Use function graph.simul.by.time.by.sim to see enzyme concentrations through time.

Examples

```
data(data_sim_CRNeg_1grpNeg1sgl)
graph.simul.conc.end(data_sim_CRNeg_1grpNeg1sgl)
```

graph.simul.triangle.diagram.e *Triangular diagram of relative concentrations for simulations of enzyme evolution*

Description

Graphics of enzyme evolution simulations obtained by function `simul.evol.enz.multiple`. Function `graph.simul.triangl.diagram.e` gives a triangular diagram of relative concentrations.

Usage

```
graph.simul.triangle.diagram.e(all_res_sim,which.enz=c(1,2,3),which.sim=NULL,
new.window=FALSE,add.eq=TRUE,add.line.eq.eff=FALSE,add.curve.lines=FALSE,
nbniv=9,niv.palette=NULL,posi.legend="topleft",cex.ini=1,cex.th=1.2,cex.eff=0.6)
```

Arguments

`all_res_sim`	List, the output of function `simul.evol.enz.multiple` (results of evolution simulation).
`which.enz`	Numeric. Which enzymes would be represented? Default is the first three enzymes, i.e. `c(1,2,3)`.
`which.sim`	Numeric vector containing integer numbers between 1 and `nsim`. Which simulations would you represent? If `NULL` (default), all simulations would be represented.
`new.window`	Logical. Do graphics appear in a new window?
`add.eq`	Logical. Do equilibrium appear on graph?
`add.line.eq.eff`	Logical. Add line of effective equilibrium for all initial concentrations? Default is `FALSE`. *See details.*
`add.curve.lines`	Logical. Add curve lines for flux dome? Default is `FALSE`. *See details.*
`nbniv`	Numeric. Number of contour lines, between 3 and 11. Default is `9`
`niv.palette`	Character vector. Color palette to be passed to contour lines.
`posi.legend`	Contour line legend position. See *details*.
`cex.ini, cex.th, cex.eff`	Numeric. Size of remarkable points (respectively initial, theoretical and effective relative concentrations).

Details

WARNING! If there is more than three enzymes in simulations ($n > 3$), be careful for interpretations. Indeed, triangular diagram is projection of a plane for which sum of coordinates is equal to 1. In other words, if $n > 3$, represented points are not strictly relative concentrations such as $e_i = E_i/Etot$, but are $x_i = E_i/(E_i + E_j + E_k)$ for the three represented enzymes `i`, `j` and `k`.

Triangular diagram cannot be developed if there is less than three enzymes ($n < 3$ or `length(which.enz)<3`). Indeed, it is a three-dimensional graph.

If there is only one simulation (length of `which.sim` equal to 1), color is black. Else, color are taken in palette `rainbow`.

Option `add.line.eq.eff=TRUE` adds a line of effective equilibrium for all possible initial concentrations. This line is only available for certain constraints, which are `"CRPos"` and `"CRNeg"`. Code is written only for three enzymes `n=3`, and requires to use same initial kinetic parameters for all simulations (set `same.kin0=TRUE`) and no mutation of activities (set `pmutA=0`). Also, same initial total concentration for all enzyme is required. Easiest way is to set `is.random.E0=TRUE`. See function `predict_eff_allE0`.

Option `add.curve.lines=TRUE` adds contour lines of flux dome. Contour lines are only available in case of competition, which are `"Comp"`, `"CRPos"` and `"CRNeg"`. It is only available for three enzymes `n=3`, and other conditions are the same than option `add.line.eq.eff=TRUE`. `nbniv` and `niv.palette` adjust number and color of contour lines respectively. See function `flux.dome.projections`.

In triangular diagram, left axis correspond to first value of `which.enz` for enzyme number, bottom axis to its second value and right axis to its third value. If options `add.line.eq.eff=TRUE` or `add.curve.lines=TRUE` are chosen and if all tests have been successfully passed, `which.enz` is automatically set to `c(1,2,3)`.

Value

Invisible matrix `why_no_lines` who explain why line of effective equilibrium or curve lines not appear even the corresponding options are set to `TRUE`. If `why_no_lines` is `NULL`, options `add.line.eq.eff` and `add.curve.lines` have been set to `FALSE` (default).

Function `graph.simul.triangle.diagram.e` returns point coordinates on triangular diagram.

See Also

See `simul.evol.graph.methods` for other plots of enzyme evolution simulation.

See `flux.dome.projections` to have details about contour lines.

Examples

```
#Saved simulation
data(data_sim_RegNeg)
graph.simul.triangle.diagram.e(data_sim_RegNeg, new.window=TRUE, add.line.eq.eff=TRUE)


#add curve lines
data(data_sim_Comp)
graph.simul.triangle.diagram.e(data_sim_Comp,new.window=TRUE,add.curve.lines=TRUE)

#all options
data(data_sim_CRNeg)
graph.simul.triangle.diagram.e(data_sim_CRNeg,new.window=TRUE,add.curve.lines=TRUE,
add.line.eq.eff=TRUE)
```

```
 #New simulation
# case for 3 enzymes
n <- 3
E0 <- c(30,30,30)
kin <- c(1,10,30)
Keq <- c(1,1,1)
nsim <- 2 # 2 simulations
N <- 1000
beta <- diag(1,n)
beta[upper.tri(beta)] <- c(0.32,0.32*(-0.43),-0.43)
#beta_12 = 0.32, beta_13 = beta_12 x beta_23, beta_23 = -0.43
t_beta <- t(beta) #because R fills matrix column by column
beta[lower.tri(beta)] <- 1/t_beta[lower.tri(t_beta)] #beta_ji = 1/beta_ij
if (n==3) {beta[lower.tri(beta)] <- 1/beta[upper.tri(beta)]} #only available if n=3
correl <- "RegNeg"

evol_sim <- simul.evol.enz.multiple(E0,kin,Keq,nsim,N,correl,beta,npt=250,
is.random.E0=TRUE,same.E0=FALSE)
graph.simul.triangle.diagram.e(evol_sim, new.window=TRUE, add.line.eq.eff=TRUE)

#add curve lines
nsim <- 3
correl <- "Comp"
evol_sim <- simul.evol.enz.multiple(E0,kin,Keq,nsim,N,correl,beta,npt=250,
is.random.E0=TRUE,same.E0=FALSE)
graph.simul.triangle.diagram.e(evol_sim,new.window=TRUE,add.curve.lines=TRUE)

#all options
correl <- "CRNeg"
evol_sim <- simul.evol.enz.multiple(E0,kin,Keq,nsim,N,correl,beta,npt=250,
is.random.E0=TRUE,same.E0=FALSE)
graph.simul.triangle.diagram.e(evol_sim,new.window=TRUE,add.curve.lines=TRUE,add.line.eq.eff=TRUE)

#several enzyme
n <- 5
E0 <- c(30,30,30,30,30)
kin <- c(1,10,30,100,1000)
Keq <- c(1,1,1,1,1)
correl <- "SC"
evol_sim <- simul.evol.enz.multiple(E0,kin,Keq,nsim,N,correl,beta,npt=250)
graph.simul.triangle.diagram.e(evol_sim,new.window=TRUE)
```

`group_types` *Which types are the different regulation groups?*

Description

Determines the type of all regulation groups from the matrix of co-regulation coefficients

Usage

```
group_types(beta_fun)
```

Arguments

beta_fun	Matrix of co-regulation coefficients

Details

Regulation groups exist in three types :

- positive group, when all regulation coefficients are positive
- negative group, when it exists at least one negative regulation coefficients in the group
- singletons, when enzyme is lone in the group, and is therefore independent from all others enzymes

The position of the groups is determined from the list of regulation, computed with function `class_group`.

Value

Return a list of three elements that contains the numbers of the regulation groups:

- `$grp_pos`: numeric vector containing the position of positive groups
- `$grp_single`: numeric vector containing the position of singletons
- `$grp_neg`: numeric vector containing the position of negative groups

If there is no group of a type, the corresponding element returns `NULL` instead of a numeric vector.

See Also

Function `class_group` to compute the list of regulation groups

Examples

```
#Only one group
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
L_Phi <- class_group(beta)
group_types(beta) #gives c(1), NULL and NULL


#Two groups
n <- 3
beta <- diag(1,n)
beta[1,2] <- -0.32
beta[2,1] <- 1/beta[1,2]
```

```
L_Phi <- class_group(beta)
group_types(beta) #gives NULL, c(2) and c(1)

#with data
data(data_sim_RegNeg_1grpNeg1grpPos)
group_types(data_sim_RegNeg_1grpNeg1grpPos$param$beta)
```

`is.B.accurate` *Verification of B accuracy*

Description

Verifies if the vector `B_fun` of global co-regulation coefficients is accurate for other functions

Usage

```
is.B.accurate(B_fun, n_fun, correl_fun)
```

Arguments

`B_fun`	Numeric vector of global co-regulation coefficients
`n_fun`	Number of enzymes in the system
`correl_fun`	Character string indicating the constraint applied on the system

Details

Different tests are performed on parameter `B_fun` to verify its accuracy.

- Is there regulation? If yes, `B_fun` is necessary.
- Do `B_fun` have a correct length? Compare `length(B_fun)` and number of enzymes `n_fun`. If difference, stops.
- Is there negative regulation in `correl_fun`? If yes, does `B_fun` include a negative regulation? If difference, stops.
- Sum of `1/B_fun` need to be equal an integer, which is the number of regulation groups.

Value

Return `TRUE` if all conditions are respected, else stop

See Also

To verify matrix of co-regulation coefficients, see function `is.beta.accurate`.

Examples

```
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)
is.B.accurate(B,3,"CRPos")
```

`is.beta.accurate` *Verification of beta matrix accuracy*

Description

Verifies if the the matrix `beta_fun` of co-regulation coefficients is accurate for other functions

Usage

```
is.beta.accurate(beta_fun, n_fun, correl_fun)
```

Arguments

`beta_fun`	Numeric matrix of co-regulation coefficients
`n_fun`	Number of enzymes in the system
`correl_fun`	Character string indicating the constraint applied on the system

Details

Different tests are performed on matrix `beta_fun` to verify its accuracy.

- Is there regulation in `correl_fun`? If yes, `beta_fun` is necessary.
- Does `beta_fun` have a correct size? Compare `nrow(beta_fun)` and `ncol(beta_fun)` to number of enzymes `n_fun`. If difference, stops.
- Is there negative regulation in `correl_fun`? If yes, does `beta_fun` include a negative regulation? If difference, stops.
- Each element of `beta_fun` diagonal needs to be equal to 1.

Value

Return `TRUE` if all conditions are respected, else stop

See Also

To verify vector of global co-regulation coefficients, see function `is.B.accurate`.

Examples

```
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
is.beta.accurate(beta,3,"CRPos")
```

`is.correl.authorized` *Verification of constraint name*

Description

Verifies if the constraint is authorized for other functions or not

Usage

```
is.correl.authorized(correl_fun)
```

Arguments

`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system

Details

Verifies if constraint is included in the six allowed cases.

Possible constraints are listed below:

- `"SC"`: independence between all enzymes
- `"Comp"`: competition for resources
- `"RegPos"`: positive regulation
- `"RegNeg"`: negative regulation
- `"CRPos"`: competition plus positive regulation
- `"CRNeg"`: competition plus negative regulation

Value

Returns TRUE if `correl_fun` is an authorized case, else stop

See Also

Need help to correctly write `correl_fun`? Use function `name.correl`.

Examples

```
is.correl.authorized("SC")
#returns TRUE
```

`mut.E.direct` *Direct mutation of enzyme concentrations*

Description

Computes the mutant value of enzyme concentrations by a direct method.

Usage

```
mut.E.direct(E_res_fun,i_fun,nu_fun,correl_fun,beta_fun=NULL)
```

Arguments

`E_res_fun`	Numeric vector of enzyme concentrations (resident)
`i_fun`	Integer number indicating the enzyme targeted by the mutation
`nu_fun`	Numeric value of **canonical** mutation effect
`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system
`beta_fun`	Matrix of co-regulation coefficients

Details

This mutation method is named *direct*, because we used canonical mutation effect to compute mutant values

This function is the one used in evolution simulation.

Value

Numeric vector corresponding to mutant value of enzyme concentrations

See Also

See function `mut.E.indirect` for an indirect computation method of mutation.

Examples

```
E <- c(30,30,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
correl <- "RegPos"
mu <- 1 #canonical size of mutation
i <- 3 #enzyme directly targeted by mutation

mut.E.direct(E,i,mu,correl,beta)
```

---

mut.E.indirect *Indirect mutation of enzyme concentrations*

Description

Computes the mutant value of enzyme concentrations by an indirect method.

Usage

```
mut.E.indirect(delta_fun,E_res,alpha_fun,i_fun)
```

Arguments

delta_fun	Numeric value of the actual effect of a mutation targeting enzyme i_fun, i.e. δ_i
E_res	Numeric vector of resident enzyme concentrations
alpha_fun	Numeric matrix of redistribution coefficients
i_fun	Integer number indicating the enzyme targeted by the mutation

Details

This mutation method is named *indirect*, because redistribution coefficient matrix M_α and actual mutation effect are used to compute mutant values rather than canonical mutation effect. Expression is : $E_j^m = E_j^r + \alpha_i j * \delta_i$

Constraints between enzymes are implicitly described in redistribution coefficients matrix.

Value

Numeric vector corresponding to mutant value of enzyme concentrations

See Also

Use function compute.delta to compute the **apparent** mutation effect.

Use function alpha_ij to compute matrix of redistribution coefficients.

See function mut.E.direct for a direct computation method of mutation.

Examples

```
E <- c(30,30,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)
correl <- "RegPos"
mu <- 1 #canonical size of mutation

alph <- alpha_ij(E,correl,beta)
delta <- compute.delta(mu,E,correl,B)

i <- 3 #enzyme directly targeted by mutation
mut.E.indirect(delta[i],E,alph,i)
```

`mut.E.old` *Old method for mutation of enzyme concentrations*

Description

Computes the mutant value of enzyme concentrations with the old method (see Lion *et al.* 2004).

Usage

```
mut.E.old(E_fun,i_fun,nu_fun,correl_fun,beta_fun=NULL,typ_E=1)
```

Arguments

`E_fun`	Numeric vector of enzyme concentrations (resident)
`i_fun`	Numeric value corresponding to number of the enzyme targeted by the mutation
`nu_fun`	Numeric value of **canonical** mutation effect
`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system
`beta_fun`	Matrix of co-regulation coefficients
`typ_E`	Numeric for mutation method. Authorized values: 1 or 2. Default is 1.

Details

This mutation method is named *old*, because it was used the first one used for evolution simulation. Some improvements has been made since.

The main difference with mutation function `mut.E.direct` is the method to compute mutant values under competition constraint. In this function `mut.E.old`, canonical effect ν is redistributed only on enzymes different from the target one, whereas in `mut.E.direct`, mutation canonical effect is redistributed between **all** enzymes, including the target one.

Moreover, computation method in `mut.E.direct` is simplified.

Last, this *old* method includes two model of mutations: additive and multiplicative.

- Additive method (typ_E=1): mutant concentration is the sum of the resident one plus size of mutation ν
- Multiplicative method (typ_E=2): mutant concentration is the product of the resident one and $1 + \nu$

Default is method 1.

In mut.E.direct, only method 1 is kept.

Value

Numeric vector corresponding to mutant value of enzyme concentrations

References

Lion, S., F. Gabriel, B. Bost, J. Fiévet, C. Dillmann, and D. De Vienne, 2004. An extension to the metabolic control theory taking into account correlations between enzyme concentrations. European Journal of Biochemistry 271:4375–4391.

See Also

See function mut.E.direct for a direct computation method of mutation.

Examples

```
E <- c(30,30,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
correl <- "RegPos"
mu <- 1 #canonical size of mutation
i <- 3 #enzyme directly targeted by mutation

mut.E.old(E,i,mu,correl,beta)
```

---

mut.kin *Mutation of kinetic parameters*

Description

Computes the mutant value of enzyme kinetic parameters

Usage

```
mut.kin(kin_fun,i_fun,nu_fun,typ_A=1)
```

Arguments

kin_fun	Numeric vector of enzyme kinetic parameter, which are the resident values
i_fun	Numeric value indicating the number of the enzyme targeted by the mutation
nu_fun	Numeric value of mutation size
typ_A	Numeric for mutation method. Default is 1. *See details in* mut.kin.

Details

This function used three mutation methods:

- Additive method (typ_A=1): mutant kinetic parameter is the sum of the resident one plus size of mutation ν
- Multiplicative method (typ_A=2): mutant kinetic parameter is the product of the resident one and $1 + \nu$
- Random method (typ_A=3): mutant kinetic parameter is equal to the mutation size

The method 1 is the default one.

Value

Numeric vector of mutant values of enzyme kinetic parameters

See Also

See function mut.E.direct to compute mutation for enzyme concentrations.

See function activities to compute "activities" from enzyme kinetic parameters.

Examples

```
kin <- c(1,10,1000)
mu <- 1 #size of mutation
i <- 3 #enzyme directly targeted by mutation

mut.kin(kin,i,mu)
```

---

name.correl *Help to name parameter 'correl'*

Description

Give the correct abbreviation of the applied constraint, used for parameter correl_fun

Usage

```
name.correl(is.comp,is.reg,beta_fun=NULL)
```

Arguments

is.comp	Logical. Is there competition for resources ?
is.reg	Logical. Is there regulation between enzymes ?
beta_fun	Numeric matrix of co-regulation coefficients. Default is NULL, but needed if there is regulation (is.reg==TRUE).

Details

Explored constraints are competition and/or regulation.

If you choose to put regulation, matrix of co-regulation coefficient beta **or** vector of global co-regulation coefficients B are needed.

See function is.correl.authorized to know possible value of parameter correl_fun.

Value

A character string, used for parameter correl_fun. See possible values in is.correl.authorized.

Examples

```
#Independence
name.correl(is.comp=FALSE,is.reg=FALSE)
#returns "SC"

#Regulation
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
name.correl(is.comp=FALSE,is.reg=TRUE,beta_fun=beta)
#returns "RegPos"
```

---

odd.discrete.sel.coef *Equation of null coefficient selection*

Description

(Utilitary function). Gives the result of the equation corresponding to discrete coefficient selection and indirect mutation of E

Usage

```
odd.discrete.sel.coef(delta_fun,i_fun,E_res,A_fun,alpha_fun,sel_coef_fun)
```

Arguments

delta_fun	Numeric value of the actual effect of a mutation targeting enzyme i_fun, i.e. δ_i
i_fun	Integer number indicating the enzyme targeted by the mutation
E_res	Numeric vector of resident enzyme concentrations
A_fun	Numeric vector of activities
alpha_fun	Numeric matrix of redistribution coefficients
sel_coef_fun	Numeric value of selection coefficient

Details

Gives the result of the equation corresponding to discrete coefficient selection and indirect mutation of E. Corresponding equation is $\sum(\frac{1}{A_j E_j}(\frac{1}{1+s} - \frac{E_j}{E_j+\alpha_{ij}\delta_i}))$

Null this equation corresponds to search the δ_i corresponding to the given resident concentrations and selection coefficient.

This function is used to find δ at bounds of the Range of Neutral Variation, where selection coefficient s is equal to $+/-1/N$.

Value

A numeric value

See Also

See function alpha_ij to compute the redistribution coefficients.

---

predict_eff *Prediction of effective equilibrium*

Description

Gives the effective equilibrium for relative concentrations

Usage

```
predict_eff(E_ini_fun,B_fun,A_fun,correl_fun, tol=0.00000001)
```

Arguments

E_ini_fun	Numeric vector of initial concentrations
B_fun	Numeric vector of global co-regulation coefficients. Same length as E_ini_fun.
A_fun	Numeric vector of activities
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
tol	Tolerance for function uniroot

Details

Gives values at effective equilibrium for relative concentrations and corresponding driving variable τ. This equilibrium corresponds to null derivative of relative concentrations, with a maximum for flux.

Effective equilibrium is found by searching the zero for response coefficients. The R function uses in this objective is `uniroot`.

Note that sum of `1/B_fun` need to be equal to 1.

When there are regulation groups, preferably use `predict_grp`.

Value

List of three elements:

- `$pred_e`: numeric vector of relative concentrations at effective equilibrium. Same length as `A_fun`
- `$pred_tau`: numeric value of driving variable *tau* at effective equilibrium
- `$pred_E`: numeric vector of absolute concentrations at effective equilibrium. Same length as `A_fun`

Special results

In case of independence (`correl_fun="SC"`) or positive regulation (`correl_fun="RegPos"`), there is no effective equilibrium, and function `predict_eff` stops.

In case of competition (`correl_fun="Comp"`), effective and theoretical equilibria are confounded. Function `predict_eff` also stops, so use preferably function `predict_th` to compute equilibrium.

If `E_ini_fun` is a multiple of `1/B_fun`, effective equilibrium is confounded with theoretical equilibrium and initial point (see `droites` for details). Function `predict_eff` returns `E_ini_fun` for `$pred_E` and 0 for `$pred_tau`, with a warning message.

References

Coton et al. (2021)

See Also

Use function `activities` to compute enzyme activities.

Use function `is.correl.authorized` to see allowed constraints for `correl_fun`.

Use function `predict_th` to compute theoretical equilibrium.

Use function `predict_grp` to predict equilibria when there are co-regulation groups.

Examples

```
###### In presence of competition plus regulation
A <- c(1,10,30)
E0 <- c(30,30,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)
```

```
eq_eff <- predict_eff(E0,B,A,"CRPos")

eq_eff$pred_e
eq_eff$pred_tau
eq_eff$pred_E
```

`predict_eff_allE0` *Prediction of effective equilibrium for all possible initial relative concentrations*

Description

Gives the effective equilibrium for relative concentrations for various initial concentrations

Usage

```
predict_eff_allE0(B_fun,A_fun,correl_fun,Etot_fun=100,X_fun=1, tol=0.00000001)
```

Arguments

`B_fun`	Numeric vector of global co-regulation coefficients. Same length as `E_ini_fun`.
`A_fun`	Numeric vector of activities
`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system
`Etot_fun`	Numeric value of total concentration
`X_fun`	Numeric value. Default is `1`
`tol`	Tolerance for function `uniroot`

Details

Effective equilibrium is computed with function `predict_eff`.

WARNING: Function `predict_eff_allE0` ***is only available for three enzymes!*** *Length of* `A_fun` *and* `B_fun` *need to be 3.*

Each relative concentration is taken between 0 and 1 by 0.01, then triplet of relative concentrations are sorted to have a sum equal to 1. Then relative concentrations are multiplied by `Etot_fun` to have initial concentrations.

For parameter `correl_fun`, authorized input are `"RegNeg"`, `"CRPos"` and `"CRNeg"`.

Value

Invisible list of 3 elements:

- `$all_eq_eff`: Dataframe of 4 851 rows and eight columns (named `e1,e2,e3,tau,E1,E2,E3,J`) for effective equilibrium from possible initial concentrations. Each row corresponds to a set of initial concentrations, and columns are respectively relative concentrations (`$e1,$e2,$e3`), driving variable τ (`$tau`), absolute concentrations `$E1,$E2,$E3` and flux `$J` at effective equilibrium;
- `$all_E0`: Dataframe of 4 851 rows and three columns corresponding to initial concentrations. Each row is a triplet of initial concentrations;
- `$param`: List of input parameters

---

predict_grp *Prediction of equilibrium with regulation groups*

Description

Gives the equilibrium for intra-group, inter-group and total relative concentrations at equilibrium

Usage

```
predict_grp(E_ini_fun,beta_fun,A_fun,correl_fun, tol=0.00000001)
```

Arguments

`E_ini_fun`	Numeric vector of initial concentrations
`beta_fun`	Matrix of co-regulation coefficients
`A_fun`	Numeric vector of activities
`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system
`tol`	Tolerance for function `uniroot`

Details

Gives values at effective equilibrium for intra-group e_i^q, inter-group e^q and total e_i relative concentrations, and group driving variable τ^q, and also for absolute concentrations E^i and concentrations sum in groups E^q. The equilibrium corresponds to null derivative for relative concentrations.

However, does not compute the theoretical intra-group equilibrium when there is competition, which is $e_i^q = 1/B_i$.

Value

List of seven elements:

- `$pred_eiq`: numeric vector of intra-group relative concentrations e_i^q at equilibrium. Same length as `A_fun`.
- `$pred_eq`: numeric vector of inter-group relative concentrations e^q at equilibrium. The length is the number of regulation groups.
- `$pred_ei`: numeric vector of total relative concentrations e_i at equilibrium. Same length as `A_fun`.
- `$pred_tau`: numeric vector of driving variable tau^q at equilibrium. The length is the number of regulation groups.
- `$pred_Ei`: numeric vector of enzyme absolute concentrations E_i at equilibrium. Same length as `A_fun`.
- `$pred_Eq`: numeric vector of sum of absolute concentrations in groups E^q at equilibrium. The length is the number of regulation groups.
- `$pred_Etot`: numeric value of total concentration at equilibrium.

Special results

When there are more than one positive or negative group and singletons with competition (`"CRPos"` or `"CRNeg"`), the equilibria are not predictable.

See Also

Use function `activities` to compute enzyme activities.

Use function `is.correl.authorized` to see allowed constraints for `correl_fun`.

Use function `predict_th` (resp. `predict_eff`) to compute theoretical (resp. effective) equilibrium when there is no regulation groups (enzymes are all independent or all co-regulated).

Examples

```
#### For independancy "SC"
A <- c(1,10,30)
E0 <- c(30,30,30)
beta <- diag(1,3)

eq <- predict_grp(E0,beta,A,"SC")
#same results for pred_e and pred_ei
eq_th <- predict_th(A,"SC")

###### In presence of regulation, all enzyme co-regulated
A <- c(1,10,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)

eq_grp <- predict_grp(E0,beta,A,"CRPos")
#same results for pred_e and pred_ei
eq_eff <- predict_eff(E0,B,A,"CRPos")
```

```
#Two groups: one negative group + one singleton
n <- 3
beta <- diag(1,n)
beta[1,2] <- -0.32
beta[2,1] <- 1/beta[1,2]

eq_grp <- predict_grp(E0,beta,A,"RegNeg")
eq_grp <- predict_grp(E0,beta,A,"CRNeg")


#Two groups: one positive group + one singleton
n <- 3
beta <- diag(1,n)
beta[1,2] <- 0.43
beta[2,1] <- 1/beta[1,2]

eq_grp <- predict_grp(E0,beta,A,"RegPos")
eq_grp <- predict_grp(E0,beta,A,"CRPos")



#With saved simulation
data(data_sim_RegPos)
n <- data_sim_RegPos$param$n
num_s <- 1
pred_eq <- predict_grp(data_sim_RegPos$list_init$E0[num_s,1:n],
data_sim_RegPos$param$beta,data_sim_RegPos$list_init$A0[num_s,1:n],data_sim_RegPos$param$correl)

data(data_sim_RegNeg_1grpNeg1grpPos)
pred_eq <- predict_grp(data_sim_RegNeg_1grpNeg1grpPos$list_init$E0[num_s,],
data_sim_RegNeg_1grpNeg1grpPos$param$beta,c(1,10,30,50),"RegNeg")
```

---

predict_th *Prediction of theoretical equilibrium*

Description

Gives the theoretical equilibrium for relative concentrations

Usage

```
predict_th(A_fun,correl_fun,B_fun=NULL)
```

Arguments

A_fun	Numeric vector of activities
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
B_fun	Numeric vector of global co-regulation coefficients

Details

Gives values at theoretical equilibrium for relative concentrations and response coefficients. This equilibrium corresponds to null derivative for relative concentrations, without conditions on flux.

When there are regulation groups, preferably use predict_grp.

Value

List of two elements:

- $pred_e: numeric vector of relative concentrations at theoretical equilibrium. Same length as A_fun
- $pred_r: numeric vector of response coefficients at theoretical equilibrium. Same length as A_fun

Special results

In case of negative regulation (correl_fun = "RegNeg" or "CRNeg"), relative concentrations would be negative.

In case of competition plus regulation (correl_fun = "CRPos" or "CRNeg"), response coefficients is not defined and $pred_r returns NaN.

See Also

Use function activities to compute enzyme activities.

Use function is.correl.authorized to see allowed constraints for correl_fun.

Use function predict_grp to predict equilibria when there are regulation groups.

Examples

```
#### For independancy "SC" or competition "Comp"
A <- c(1,10,30)

eq_th <- predict_th(A,"SC")

eq_th$pred_e
eq_th$pred_r

###### In presence of regulation
A <- c(1,10,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)
```

```
eq_th <- predict_th(A,"CRPos",B)

eq_th$pred_e
eq_th$pred_r
```

---

range_delta *Bounds of* delta_i

Description

Computes the bounds of the actual mutation effect δ_i such as all mutant concentrations are between 0 and total concentration, for a mutation targeting enzyme i

Usage

```
range_delta(E_res,alpha_fun,i_fun,tol_fun=0.0001)
```

Arguments

E_res	Numeric vector of resident enzyme concentrations
alpha_fun	Numeric matrix of redistribution coefficients
i_fun	Integer number indicating the enzyme targeted by the mutation
tol_fun	Numeric and positive value. Accuracy for delta bounds. Default is 0.0001

Details

This function range.delta computes the bounds of δ_i such as all mutant concentrations are between 0 and total concentration $Etot$, for a mutation targeting enzyme i_fun. Mutant concentrations are equal to resident concentrations plus $\alpha_i j * \delta_i$ ($see function$mut.E.indirect). For any enzyme j, mutant value is $E_j^r + \alpha_i j * \delta_i$.

The inferior (resp. superior) bound of δ_i corresponds to minimal (resp. maximal) value of δ_i such as all mutant concentrations are superior or equal to 0 **and** inferior or equal to $Etot$, with at least one mutant concentration equal to 0 or $Etot$.

tol_fun is the accuracy (or allowed tolerance) for δ bounds. It allows to avoid asymptote problem when computing the RNV.

Value

Numeric vector of the inferior and the superior bounds of actual mutation effect δ_i

See Also

See function alpha_ij to compute matrix of redistribution coefficients alpha_fun.

Examples

```
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
Er <- c(30,30,30)
correl <- "CRPos"
alpha <- alpha_ij(Er,correl,beta)

#mutant enzyme
i <- 1

range_delta(Er,alpha,i)
```

`range_tau` *Bounds of* tau

Description

Computes the bounds of the driving variable (or position) τ on the straight line of relative concentrations such as thereof are between 0 and 1

Usage

```
range_tau(E_ini_fun,B_fun)
```

Arguments

`E_ini_fun`	Numeric vector of initial concentrations
`B_fun`	Numeric vector of global co-regulation coefficients. Same length as `E_ini_fun`.

Details

Relative concentrations is defined in [0,1]. This function `range_tau` computes the bounds of τ such as all relative concentrations are between 0 and 1.

The inferior (resp. superior) bound of τ corresponds to minimal (resp. maximal) value of τ such as all relative concentrations are superior or equal to 0 **and** inferior or equal to 1, with at least one relative concentration equal to 0 or 1.

Value

Numeric vector of the inferior and the superior bounds of driving variable τ

See Also

See details of function `droite_e` for detailed explanations on τ.

Examples

```
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- apply(beta,1,sumbis)
E0 <- c(30,30,30)

range_tau(E0,B)
```

`RNV.compute.elements` *Compute RNV and associated elements for all enzymes* (Deprecated)

Description

(Deprecated). This function computes different elements at RNV, for different resident concentrations, based on function `RNV.delta.all.enz` For simulations, preferably use `RNV.for.simul`.

Usage

```
RNV.compute.elements(mat_E,mat_A,N_fun,correl_fun,beta_fun=NULL,
end.mean=TRUE,add.RNV.J=FALSE,mat_J=NULL,X_fun=1)
```

Arguments

`mat_E`	Numeric matrix of concentrations. Columns correspond to enzyme number, and rows correspond to different resident.
`mat_A`	Numeric matrix of activities. Same dimensions as `mat_E`.
`N_fun`	Numeric. Population size
`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system
`beta_fun`	Matrix of co-regulation coefficients
`end.mean`	Logical. If `FALSE`, compute RNV size mean of all resident. If `TRUE`, compute RNV size mean for last half of resident only.
`add.RNV.J`	Logical. Add value of flux at RNV ? If `TRUE`, possibility set also `mat_J`.
`mat_J`	Numeric matrix of flux. One column and same rows number as `mat_E`. Optional.
`X_fun`	Numeric value. Default is `1`

Details

The *Range of Neutral Variations* (RNV) are mutant concentration values such as coefficient selection is between $1/(2N)$ and *-1/(2N)*.

Inferior (resp. superior) bound of RNV corresponds to selection coefficient equal to $-1/(2N)$ (resp. $1/(2N)$).

Function `RNV.compute.elements` computes the δ_i at RNV bounds (where i is the enzyme targeted by the mutation), but also mutant concentrations at RNV bounds and the RNV size.

The RNV size is the absolute value of $\delta_i^s up$ minus $\delta_i^i nf$.

Depending on applied constraint `correl_fun`, it exists 1 or 2 RNV. In case of independence (`"SC"`) or positive regulation between all enzymes (`"RegPos"`), flux has no limit and there is only one RNV. In other cases (competition and/or negative regulation), because flux can reach a maximum, there is two RNVs: a "near" one, for small mutations, and a "far" one for big mutations that put mutant in the other side of flux dome.

This function `RNV.compute.elements` is designed to compute RNV of *one* simulation launched by `simul.evol.enz.multiple`. For example, for simulation 1, put `mat_E=tabR[tabR$sim==1,1:n]` and `mat_A=tabR[tabR$sim==1,(2*n+4):(3*n+3)]`. To reduce computation time, put `mat_J=tabR[tabR$sim==1,(2*n+3)` for flux. It also works for different resident values of concentrations and activities.

Value

List of 7 elements:

- `$RNV_delta` : list of `n` elements, one by enzyme. Each element contains a numeric matrix of `nb_resid` rows and two or four columns. For each enzyme i (a list element), each row corresponds to one resident and columns are actual mutation effect δ_i corresponding respectively to inferior bound (col 1) and superior bound (col 2) of RNV (x2 if there is a second RNV). Each row is a result of `RNV.delta.all.enz`.
- `$RNV_enz` : same structure as `$RNV_delta`, but for mutant enzyme concentrations at RNV limits, *i.e.* $E_i + \delta_i$ for each target enzyme i ;
- `$RNV_size` : list of one or two elements (depending on RNV number). Each element contains a matrix of `n` columns (one by enzyme) and `nb_resid` rows. Each cell is the RNV size for current enzyme (in column) and current resident (in row). The RNV size is absolute value of $\delta_i^s up$ minus $\delta_i^i nf$. If there is no superior bounds but there is two RNV, RNV size is obtained by the difference of the two $\delta_i^i nf$.
- `$RNV_size_divEtot` : same structure as `$RNV_size`, but for RNV size divided by total concentration of corresponding resident.
- `$RNV_proxy` : numeric matrix of one or two rows (depending on RNV number) and `n` columns (one by enzyme). Each cell id the mean of RNV size. If `end.mean=TRUE`, the mean is computed from last half of resident.
- `$RNV_proxy_divEtot` : same structure as `$RNV_proxy`, but mean of RNV size is divided by mean of total concentration.
- `$RNV_flux` : numeric matrix of two columns (inferior and superior limits of neutral zone) and `nb_resid` rows. Each cell is the flux value at neutral zone limits.

Note that `n` is the number of enzymes. `nb_resid` is the number of resident and `s` also the row number of `mat_E` and `mat_A`.

See Also

See function `RNV.delta.all.enz` to see how actual mutation effect δ is computed.

For simulations, preferably use `RNV.for.simul`. Code is almost the same for the two functions, but differs in input.

Examples

```
#for 2 resident genotypes and 3 enzymes
Er <- matrix(30,ncol=3,nrow=2)
A <- matrix(c(1,10,30),byrow=TRUE,ncol=3,nrow=2)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- compute.B.from.beta(beta)
correl <- "CRPos"
N <- 1000

#second resident = theoretical equilibrium of first resident
Er[2,] <- 100*predict_th(A[1,],correl,B)$pred_e

RNV.compute.elements(Er,A,N,correl,beta,add.RNV.J=TRUE)
```

RNV.delta.all.enz *Delta at RNV for all enzymes*

Description

This function computes actual mutation effect δ at RNV for each enzyme

Usage

```
RNV.delta.all.enz(E_res_fun,A_fun,N_fun,correl_fun,beta_fun=NULL,tol_fun=0.0001)
```

Arguments

E_res_fun	Numeric vector of enzyme concentrations (resident)
A_fun	Numeric vector of activities
N_fun	Numeric. Population size
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
beta_fun	Matrix of co-regulation coefficients
tol_fun	Numeric and positive value. Accuracy for delta bounds. Default is 0.0001

Details

The *Range of Neutral Variations* (RNV) are mutant concentration values such as coefficient selection is between $1/(2N)$ and $-1/(2N)$.

Inferior (resp. superior) bound of RNV corresponds to selection coefficient equal to $-1/(2N)$ (resp. $1/(2N)$).

Function RNV.delta.all.enz computes the actual mutation effect $delta_i$ at RNV bounds, where i is the enzyme targeted by the mutation.

Depending on applied constraints correl_fun, it exists 1 or 2 RNV. In case of independence ("SC") or positive regulation between all enzymes ("RegPos"), flux has no limit and there is only one RNV. In other cases (competition and/or negative regulation), because flux can reach a maximum, there is two RNV: a "near" one, for small mutations, and a "far" one for big mutations that put mutants in the other side of flux dome.

Known bug

Due to use of range_delta to limit search area, output δ is computed to have mutant concentration between 0 and sum(E_res_fun) (resident total concentration). If δ is too high (mutant concentration over resident total concentration), output δ could be NA rather than a numeric value. This case might happen when N_fun is too low and correl_fun="SC" or "RegPos" (no limit on total concentration),

Value

List of *n* elements, one for each enzyme *i* considering that i is targeted by the mutation.

For each element of the list, we have a vector of length 2 or 4, depending on applied constraint correl_fun. Values of this vector are:

1. *delta_i* value for inferior bounds of the near RNV
2. *delta_i* value for superior bounds of the near RNV
3. *delta_i* value for inferior bounds of the far RNV (if exists)
4. *delta_i* value for superior bounds of the far RNV (if exists)

If superior bound is not accessible, value is NA.

Note that n is the number of enzymes, which is the length of E_ini_fun.

References

Coton et al. (2021)

See Also

$delta$ at RNV bounds is obtained by nullify the expression in odd.discrete.sel.coef.

Examples

```
Er <- c(30,30,30)
A <- c(1,10,30)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
correl <- "CRPos"
N <- 1000

RNV.delta.all.enz(Er,A,N,correl,beta)

correl <- "SC"
RNV.delta.all.enz(Er,A,N,correl)
```

RNV.for.simul *Compute RNV for simulation*

Description

Computes different elements at RNV for evolution simulation

Usage

```
RNV.for.simul(res_sim,n_fun,N_fun,correl_fun,beta_fun=NULL,
end.mean=TRUE)
```

Arguments

res_sim	Dataframe corresponding to result of one simulation. *See details*.
n_fun	Integer number indicating the number of enzymes.
N_fun	Numeric. Population size
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
beta_fun	Matrix of co-regulation coefficients
end.mean	Logical. If FALSE, compute RNV size mean for all rows of res_sim. If TRUE, compute RNV size mean for last half of res_sim rows.

Details

This function is designed to computes RNV of *one* simulation launched by simul.evol.enz.one. Input res_sim is a result of simul.evol.enz.one. If you used simul.evol.enz.multiple, input tabR for parameter res_sim. If input is several simulations, remember that number of rows for one simulation is npt. *See below.*

The *Range of Neutral Variations* (RNV) are mutant concentration values such as coefficient selection is between $1/(2N)$ and $-1/(2N)$.

Inferior (resp. superior) bound of RNV corresponds to selection coefficient equal to $-1/(2N)$ (resp. $1/(2N)$).

Function RNV.for.simul computes the actual mutation effect δ_i at RNV bounds (where i is the enzyme targeted by the mutation), but also mutant concentration at RNV bounds and the RNV size.

Depending on applied constraint correl_fun, it exists 1 or 2 RNV. In case of independence ("SC") or positive regulation between all enzymes ("RegPos"), flux has no limit and there is only one RNV. In other cases (competition and/or negative regulation), because flux can reach a maximum, there is two RNV: a "near" one, for small mutations, and a "far" one for big mutations that put mutants on the other side of flux dome.

RNV size

The RNV size is the absolute value of $\delta_i^s up$ minus $\delta_i^i nf$. If there is no superior bounds but there is two RNVs, RNV size is obtained by the difference of the two $\delta_i^i nf$.

The mean of RNv size is the mean of every resident for each enzyme. If `end.mean=TRUE`, only last half of resident (the last half of `res_sim` rows) are used to compute RNV mean.

Use of `res_sim`

`res_sim` is a numeric matrix of `(3*n+4)` columns and at least 2 rows. Respective columns are: concentrations (`1:n`), kinetic parameters (`n+1:2n`), total concentration (`2n+1`), total kinetic (`2n+2`), flux/fitness (`2n+3`) and activities (`2n+4:3n+3`), corresponding to *(E1 to En, kin_1 to kin_n, Etot, kin_tot, J, A1 to An). See function* `simul.evol.enz.one`.

`res_sim` is normally output `$res_sim` of function `simul.evol.enz.one`. Output `$tabR` of function `simul.evol.enz.multiple` is also possible, by selecting a simulation with `x$tabR[x$tabR$sim==i,]` where `i` is simulation number. If input is several simulations, remember that number of rows for one simulation is `npt`.

Other parameters (`n_fun,N_fun,correl_fun,beta_fun`) are available in output `$param` of simulation functions.

Value

Invisible list of 7 elements:

- `$RNV_delta`: list of `n` elements, one by enzyme. Each element contains a numeric matrix of `nb_resid` rows and two or four columns. For each enzyme `i` (a list element), each row corresponds to one resident and columns are actual mutation size δ_i corresponding respectively to inferior bound (col 1) and superior bound (col 2) of RNV (x2 if there is a second RNV). Each row is in fact a result of `RNV.delta.all.enz`.
- `$RNV_enz`: same structure as `$RNV_delta`, but for mutant enzyme concentrations at RNV limits, *i.e.* $E_i + \delta_i$ for each target enzyme `i`.
- `$RNV_size`: list of one or two elements (depending RNV on RNV number). Each element contains a matrix of `n` columns (one by enzyme) and `nb_resid` rows. Each cell is the RNV size *(see details)*.
- `$RNV_size_divEtot`: same structure as `$RNV_size`, but for RNV size divided by total concentration of corresponding resident.
- `$RNV_proxy`: numeric matrix of one or two rows (depending on RNV number) and `n` columns. Each cell is the mean of RNV size *(see details)*.
- `$RNV_proxy_divEtot`: same structure as `$RNV_proxy`, and contains mean of RNV size divided by total concentration.
- `$RNV_flux`: numeric matrix of two columns (inferior and superior limits of neutral zone) and `nb_resid` rows. Each cell is the flux value at neutral zone limits.
- `$nb_RNV`: number of RNV. If constraint is `"SC"` or `"RegPos"`, there is one RNV, else two.
- `$limits_NZ`: numeric vector of the two limits of neutral zone

Note that `n` is the number of enzymes. `nb_resid` is the number of resident and is also the rows number of `res_sim`.

See Also

See function RNV.delta.all.enz to see how δ is computed.

Use function simul.evol.enz.one to launch a simulation, or simul.evol.enz.multiple for several simulations.

Examples

```
 #### Construction of false simulation
#for 2 resident genotypes and 3 enzymes
n <- 3
Er <- c(30,30,30)
kin <- c(1,10,30)
Keq <- c(1,1,1)
A <- activities(kin,Keq)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
B <- compute.B.from.beta(beta)
correl <- "RegPos"
N <- 1000

#on one line
first_res <- cbind(t(Er),t(kin),sum(Er),sum(kin),flux(Er,A),t(A))

#second resident = theoretical equilibrium of first resident
Eq_th <- 100*predict_th(A,correl,B)$pred_e
second_res <- cbind(t(Eq_th),t(kin),sum(Eq_th),sum(kin),flux(Eq_th,A),t(A))

false_sim <- rbind(first_res,second_res)

RNV_elements <- RNV.for.simul(false_sim,n,N,correl,beta)

RNV_elements$RNV_delta #apparent mutation size at RNV for enzymes 1, 2 and 3
RNV_elements$RNV_enz #concentrations
RNV_elements$RNV_size #RNV size
RNV_elements$RNV_proxy #RNV size mean
RNV_elements$RNV_flux #flux at neutral zone


#With saved simulation
data(data_sim_RegPos)
RNV_elements <- RNV.for.simul(data_sim_RegPos$tabR,data_sim_RegPos$param$n,
data_sim_RegPos$param$N,data_sim_RegPos$param$correl,data_sim_RegPos$param$beta)
```

RNV.graph.double.at.eq

Graph of e and RNV in relation to A at equilibrium

Description

Plot of RNV size and relative concentrations in relation to activities at equilibrium

Usage

```
RNV.graph.double.at.eq(list_eq,posi.legend="topleft",add.title=TRUE,
cex.lab=1,mar.lab=2.5,enz.lab=FALSE,...)
```

Arguments

list_eq	Output of function RNV.size.at.equilibr: • list_eq$A: Numeric vector of activities (length n) • list_eq$RNV_size: Numeric matrix of n column and 2 rows. Only first row will be used, corresponding to near RNV (small mutations). • list_eq$e_eq: Numeric vector of equilibrium (length n)
posi.legend	Character string. Where would you put legend? See parameter x in function legend. If NULL, legend will not appear.
add.title	Logical. Add a title to the plot?
cex.lab	Numeric. Size of axis label.
mar.lab	Numeric. Distance of label from axis.
enz.lab	Logical. Add enzyme name on graph?
...	Arguments to be passed to plot function, such cex or cex.lab

Details

list_eq$A, list_eq$RNV_size[1,] and list_eq$e_eq are binded in a data.frame of n rows (one by enzyme) and three columns (A, RNV and e), then ordered according to A.

Title (option add.title=TRUE) is "Comparing e and RNV at equilibrium".

Value

Invisible data.frame of n rows (one by enzyme) and three columns ($A, $RNV and $e).

See Also

See RNV.size.at.equilibr to compute RNV.

Examples

```
list_eq <- RNV.size.at.equilibr(20,"Comp",1000,Etot_0=100,show.plot=FALSE)
RNV.graph.double.at.eq(list_eq)
```

RNV.mean.simul *Mean of RNV size in simulation*

Description

Computes mean of RNV size from simulation results and gives a graph of this RNV mean in relation to the RNV-ranking-order factor. Computes also a linear model of RNV mean in relation to RNV-ranking-order factor.

Usage

```
RNV.mean.simul(all_res_sim,end.mean=TRUE,which.sim=NULL,add.lm=TRUE,
add.mean=TRUE,add.pred.e=FALSE,
show.plot=TRUE,new.window=FALSE,cex.lab=1,mar.lab=2.5,...)
```

Arguments

all_res_sim	List, the output of function simul.evol.enz.multiple (results of evolution simulation).
end.mean	Logical. If FALSE, compute RNV size mean for all rows of res_sim. If TRUE, compute RNV size mean for last half of res_sim rows.
which.sim	Numeric vector containing integer numbers between 1 and nsim. Which simulations would you represent? If NULL (default), all simulations would be represented.
add.lm	Logical. Add line of linear model in graphics?
add.mean	Logical. Add mean of RNV size between all selected simulation for each enzyme?
add.pred.e	Logical. Add predicted relative concentrations mean between selected simulations? *See concerned graphs in details*
show.plot	Logical. Are plots visible?
new.window	Logical. Do graphics appear in a new window?
cex.lab	Numeric. Size of axis label.
mar.lab	Numeric. Distance of label from axis.
...	Arguments to be passed in plot function, such as lwd or cex.

Details

RNV.mean.simul works in three parts:

1. computing mean of RNV size
2. plotting RNV mean in relation to various variables
3. computing RNV mean against RNV-ranking-order factor

About RNV mean computing

Function `RNV.mean.simul` is designed to compute mean of RNV size in simulations launched by `simul.evol.enz.multiple`. Input `all_res_sim` is the output of `simul.evol.enz.multiple`.

RNV mean is computed by enzyme and by simulation, and a general mean for each enzyme (between selected simulations) is also computed.

RNV mean is made on all rows in simulation results (`end.mean=FALSE`) or only on last half rows of each simulation (`end.mean=TRUE`), i.e. when equilibrium is reached.

About graphics

Function `RNV.mean.simul` gives three graphics:

1. RNV mean in relation to activities
2. RNV mean in relation to the RNV-ranking-order factor (see `RNV.ranking.order.factor`)
3. RNV mean in relation to a soft value of the RNV-ranking-order factor (activities A for `"SC"` and `"Comp"`; global co-regulations coefficients B for `"RegPos"` and `"RegNeg"`; the hard value of the RNV-ranking-order factor for `"CRPos"` and `"CRNeg"`). Squares correspond to simulation.

Each simulation corresponds to one color. Colors for simulations are taken in palette `rainbow`.

About linear model

Function `RNV.mean.simul` computes also a linear model of RNV mean in relation to the RNV-ranking-order factor between **all** simulations (and not only selected ones by `which.sim`). If wanted, linear model can be put on graphics.

About logical parameters

Last graphic is RNV mean in relation to an interest variable, which is activities **A** (`"SC"` or `"Comp"` cases), global co-regulation coefficients **B** (`"RegPos"` or `"RegNeg"` cases) or the RNV-ranking-order factor (`"CRPos"` or `"CRNeg"` cases, see above).

In this last graphic, `add.mean=TRUE` adds the mean of RNV size between selected simulations, with black squares and line.

Also in this last graphic, `add.pred.e=TRUE` adds mean (between selected simulations) of the predicted relative concentrations at equilibrium, with grey axis, grey dots and grey dashed line.

`add.lm=TRUE` adds the linear model (black line) in the second graph (RNV mean against the RNV-ranking-order-factor).

Value

Invisible list of 5 elements:

- `$RNV_all_sim`: list of `nsim` elements (which is the number of simulation). Each element `i` contains the output of function `RNV.for.simul` for corresponding simulation `i`. If simulation is not selected by `which.sim`, corresponding element is `NULL`.
- `$RNV_mean_size`: numeric matrix of `n+2` columns and number of rows is between `length(which.sim)` and `2*length(which.sim)` (depending if there is one or two RNVs). Each of the `n` first columns is the RNV mean size for corresponding enzyme. Column `n+1` indicates the simulation number and column `n+2` the RNV number (1 for near RNV and 2 for far RNV).

- `$rank_var_value`: numeric matrix of `n` columns and `length(which.sim)` rows. Each cell is the value of the RNV-ranking-order factor to which RNV mean size is compared, for each simulation (in row) (in column) and each enzyme.
- `$rank_var_name`: character string, indicating the name of the RNV-ranking-order factor.
- `$lm_RNV`: object of `class "lm"`. Linear model of RNV the mean (only for near RNV or RNV number 1) in relation to the RNV-ranking-order factor.

See Also

RNV is computed with function `RNV.for.simul`.

Use function `graph.simul.by.time.RNV` to have other representations of RNV.

See function `RNV.ranking.order.factor` for details about RNV-ranking-order factor.

Examples

```
# With saved simulation
data(data_sim_SC)
RNV.mean.simul(data_sim_SC,new.window=TRUE,which.sim=c(1,5,10))



# case for 3 enzymes
n <- 3
E0 <- c(30,30,30)
kin <- c(1,10,30)
Keq <- c(1,1,1)
nsim <- 2 # 2 simulations
N <- 1000
correl <- "SC"

evol_sim <- simul.evol.enz.multiple(E0,kin,Keq,nsim,N,correl,npt=250)

RNV.mean.simul(evol_sim,new.window=TRUE)
```

RNV.mean.suitability *Compare RNV mean in simulation with predicted RNV at equilibrium*

Description

Compare RNV mean in simulation with predicted RNV at equilibrium to measure suitability of RNV mean as a proxy of RNV at equilibrium With a graph and a linear model.

Usage

```
RNV.mean.suitability(all_res_sim,end.mean=TRUE,which.sim=NULL,
new.window=FALSE,posi.legend="topleft",...)
```

Arguments

all_res_sim	List, the output of function simul.evol.enz.multiple (results of evolution simulation).
end.mean	Logical. If FALSE, compute RNV size mean for all rows of res_sim. If TRUE, compute RNV size mean for last half of res_sim rows.
which.sim	Numeric vector containing integer numbers between 1 and nsim. Which simulations would you represent? If NULL (default), all simulations would be represented.
new.window	Logical. Do graphics appear in a new window?
posi.legend	Character string. Where would you put the legend? See parameter x in function legend. If NULL, legend will not appear.
...	Arguments to be passed in plot function, such as lwd or cex.

Details

Function RNV.mean.suitability computes mean of RNV size by enzyme, using function RNV.mean.simul. It computes also the RNV size around predicted equilibrium, using function RNV.size.at.equilibr. Then RNV.mean.suitability plots the RNV means in relation to predicted RNVs for every enzyme and every simulation.

RNV.mean.suitability computes also a linear model of RNV means in relation to RNV size at equilibrium, for selected simulations only (by setting which.sim).

Each simulation corresponds to one color. Colors for simulations are taken in palette rainbow. Displayed number is the enzyme number. Black line is the linear model. Dashed line is the symmetry line.

Function RNV.mean.suitability is designed to compute mean RNV from simulations launched by simul.evol.enz.multiple. Input all_res_sim is the output of simul.evol.enz.multiple.

Value

Invisible list of 6 elements:

- $RNV_mean_simul: numeric matrix of n+2 columns and number of rows is between nsim and 2*nsim (depending on RNV number). n first columns contain RNV mean for corresponding enzyme. Column n+1 indicates simulation number and column n+2 the RNV number (1 for near RNV and 2 for far RNV).
- $list_eq_all: list of nsim elements. Each element s is the output of function RNV.size.at.equilibr for simulation s
- $RNV_at_eq: numeric matrix of n+1 columns and nsim rows. n first columns correspond to RNV size at predicted equilibrium for corresponding enzyme. Column n+1 indicates simulation number.

- $lm_compar_RNV: object of class "lm". Linear model of RNV mean for near RNV (RNV number 1) in relation to RNV at predicted equilibrium.
- $A_mean: numeric matrix of n columns (enzyme) and nsim rows (simulation). Each cell is the activity mean for each enzyme (in column) and each simulation (in row).
- $Etot_mean: numeric vector of length nsim. Each value is the total concentration mean for each simulation.

See Also

Function RNV.for.simul is used to compute RNV.

Function RNV.mean.simul is used to compute RNV mean for each simulation.

Function RNV.size.at.equilibr is used to compute RNV at equilibrium.

Examples

```
# With saved simulation
data(data_sim_SC)
RNV.mean.suitability(data_sim_SC,new.window=TRUE,which.sim=c(1,4,6))


data(data_sim_RegNeg)
RNV.mean.suitability(data_sim_RegNeg,new.window=TRUE)
```

RNV.ranking.order.factor

Name and value of RNV-ranking-order factor

Description

Gives the name and values of the RNV-ranking-order factor

Usage

```
RNV.ranking.order.factor(A_fun,correl_fun,E_ini_fun,B_fun=NULL)
```

Arguments

A_fun	Numeric vector of activities
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
E_ini_fun	Numeric vector of initial concentrations.
B_fun	Numeric vector of global co-regulation coefficients

Details

Factors governing ranking order of RNV

RNV-ranking-order factor depends on constraint. It would be:

- in case of independence (`correl_fun="SC"`): activities `A` power 1/3;
- in case of competition only (`correl_fun="Comp"`): activities `A` power 1/4;
- in case of regulation only (`correl_fun="RegPos"` or `"RegNeg"`): absolute value of inverse of global co-regulation coefficients `|1/B|`;
- in case of competition and regulation (`correl_fun="CRPos"` or `"CRNeg"`): absolute value of inverse of global co-regulation coefficients minus initial relative concentrations `|1/B -e0|`.

Value

Invisible list of 2 elements:

- `$value`: numeric vector (length `n`) of the values of the RNV-ranking-order factor for each enzymes
- `$name`: character sting indicating the name of the RNV-ranking-order variable

Examples

```
A <- c(1,20,30)
E0 <- c(30,30,30)

#Independence
rank_var <- RNV.ranking.order.factor(A,"SC",E0)
all.equal(A^(1/3),rank_var$value) #TRUE

#Positive regulation
B <- 1/c(0.2,0.5,0.3)
rank_var <- RNV.ranking.order.factor(A,"RegPos",E0,B)
all.equal(1/B,rank_var$value) #TRUE
```

RNV.size.at.equilibr *Plot and linear model of RNV size at equilibrium*

Description

Gives a plot and computes a linear model of the RNV size at equilibrium against the activities and the RNV-ranking-order factor

Usage

```
RNV.size.at.equilibr(n_fun,correl_fun,N_fun,A_fun=NA,B_fun=NA,E0_fun=NA,
Etot_0=sum(E0_fun),Etot_eq=100,A_lim=c(0.001,10000),inv_B_lim=c(0,100),E0_lim=c(1,100),
RNV.by.E=FALSE,show.plot=TRUE,enz.lab=FALSE,...)
```

Arguments

n_fun	Integer number indicating the number of enzymes.
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
N_fun	Numeric. Population size
A_fun, B_fun, E0_fun	Numeric vectors (length n_fun) of activities, global co-regulation coefficients and initial concentrations respectively. If NA (default), values are randomly chosen.
Etot_0	Numeric value of initial total concentration. E0_fun is rescaled to have sum(E0_fun)=Etot_0.
Etot_eq	Numeric value of total concentration at equilibrium for cases "SC" and "RegPos". Default is 100.
A_lim	Numeric vector of the two limits between which activities are chosen if A_fun=NA.
inv_B_lim	Numeric vector of the two limits between which inverse of global co-regulations B are chosen if B_fun=NA.
E0_lim	Numeric vector of the two limits between which initial concentrations E0 are chosen if E0_fun=NA.
RNV.by.E	Logical. Show RNV divided by Ei rather than RNV size?
show.plot	Logical. Show plot of RNV in relation to RNV-ranking-order factor?
enz.lab	Logical. Add enzyme name above points in graphics?
...	Arguments to be passed to plot function, such cex or cex.lab

Details

Function RNV.size.at.equilibr gives a plot of RNV size in relation to activities, and then a plot of RNV size in relation to RNV-ranking-order-factor.

RNV size is computed at equilibrium with function RNV.for.simul. Only "near" RNV (small mutations) is used.

WARNING! This function is not adapted or regulation groups (1<sum(1/B)<n).

Factors governing ranking order of RNV

RNV-ranking-order factor depends on constraint. See function RNV.ranking.order.factor.

Random variables

Input parameters A_fun,B_fun,E0_fun are chosen randomly if not specified (default NA).

Activities A_fun are taken in a uniform law between limits given by A_lim.

Initial concentrations E0_fun are taken in a uniform law between limits given by E0_lim, then leveled to have sum of E0_fun equal to Etot_fun.

Global co-regulation coefficients B_fun are chosen differently. The inverse of B are taken between limits given by inv_B_lim. If correl_fun is equal to "RegNeg" or "CRNeg" (negative co-regulations), sign are randomly chosen. Then these inverse values are leveled to have sum of 1/B equal to 1. Thus B are computed by the reverse operation, and therefore the matrix of beta by function compute.beta.from.B.

Equilibrium

RNV is computed at equilibrium, theoretical one for cases "SC", "Comp" and "RegPos", and effective one for cases "RegNeg", "CRPos" and "CRNeg".

Graphics

If RNV.by.E=TRUE, RNV size divided by its corresponding enzyme concentration is plotted, rather than RNV size.

Value

Invisible list of 12 elements:

- $RNV_size: matrix of one or two rows and n columns indicating the RNV size of every enzymes (in columns) and current RNV (near or far, in rows). See output $RNV_size of function RNV.for.simul;
- $lm_RNV: linear model of RNV size in relation to the ranking-order variable;
- $E_eq: numeric vector (length n) of concentrations at equilibrium;
- $e_eq: numeric vector (length n) of relative concentrations at equilibrium;
- $A: numeric vector (length n) of activities;
- $B: numeric vector (length n) of global co-regulation coefficients;
- $beta: numeric matrix of n rows and n columns indicating the co-regulation coefficients;
- $E0: numeric vector (length n) of initial concentrations;
- $rank_var: list of 2 elements:
  - $value: numeric vector (length n) of the RNV-ranking-order factor values for each enzyme *(see details)*;
  - $name: character string indicating the name of the RNV-ranking-order factor.
- $N, $n, $correl: numeric value of population size, number enzymes and applied constraints respectively (respectively input value of N_fun, n_fun and correl_fun)

See Also

RNV is computed with the function RNV.for.simul.

See function RNV.ranking.order.factor to have further details on RNV-ranking-order.

Examples

```
RNV.size.at.equilibr(20,"Comp",1000)
RNV.size.at.equilibr(100,"CRNeg",1000)
RNV.size.at.equilibr(3,"SC",1000,c(1,10,30),NA,c(30,30,30),100,200)
 #in this case, sum(E0)=100 and sum(E*)=200
```

search_group *Search the regulation group for an enzyme*

Description

Give the number of the regulation group Phi_q where is the interest enzyme giving the list of regulation group

Usage

```
search_group(i,Lv)
```

Arguments

i	Integer which is the number of the interest enzyme
Lv	List of regulation group, preferably the output of function class_group

Details

Enzymes are classified in regulation groups depending on the co-regulation matrix (see function link{class_group}).

Function search_group also allows to find the group type from the list of group types. In that case, i is the interest group, and Lv is the list of group type, which the output of function group_types.

More largely, the function gives the number of a list element that contains a particular number.

Value

Return an integer which is the number q of the group Phi_q that contains the interest enzyme

See Also

Function class_group to compute the list of regulation groups

Examples

```
## One group
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
L_Phi <- class_group(beta)

#enzyme 2 is in group 1
search_group(2,L_Phi)

## Two groups
n <- 3
beta <- diag(1,n)
beta[1,2] <- -0.32
beta[2,1] <- 1/beta[1,2]
```

```
L_Phi <- class_group(beta)

search_group(2,L_Phi) #gives 1
search_group(3,L_Phi) #gives 2
```

SimEvolEnzCons *SimEvolEnzCons: Simulation of Enzyme Evolution Under Constraints*

Description

Simulate the evolution of enzyme concentrations under constraints in a metabolic pathway, in accordance to the theoretical background developed in Coton *et al.* (2021).

The functions are divided in five sections. Succinct definitions are given below. There is also a section about the correct use of recurrent function parameters.

In version 2.0.0 and more, some functions have been added or modified to take account of regulation groups. Most important modifications are listed in section *Regulation groups*.

Informations about recurrent parameters

Integer numbers

`n_fun` (or `n`) is the number of enzymes in he pathway.

`nsim` is the number of simulations.

Allowed constraints

`correl_fun` is used to indicate the constraint applied on the system. Possible constraints and their corresponding abbreviation to pass in `correl_fun` are listed below:

- `"SC"`: independence between all enzymes
- `"Comp"`: competition for resources
- `"RegPos"`: positive regulation
- `"RegNeg"`: negative regulation
- `"CRPos"`: competition plus positive regulation
- `"CRNeg"`: competition plus negative regulation

`name.correl` helps to find the correct abbreviation, and `is.correl.authorized` verifies the abbreviation.

Co-regulation coefficients

For cases with co-regulations (i.e. `correl_fun` value is `"RegPos"`, `"RegNeg"`, `"CRPos"` or `"CRNeg"`), `beta_fun` or `B_fun` is obligatory. In other cases (i.e. `correl_fun` value is `"SC"` or `"Comp"`), `beta_fun` and `B_fun` are ignored, that is why default is `NULL`.

`beta_fun` is a square matrix of size `n*n`, indicating co-regulation coefficients. `B_fun` is vector of length `n`, indicating global co-regulation coefficients.

`compute.beta.from.B` and `compute.B.from.beta` help to compute `beta_fun` and `B_fun` from another.

Initial concentrations `E_ini_fun` is used in the same way as `B_fun`

Basic functions

Basic functions used for other functions.

- `flux`: computes flux depending on A, E and X
- `activities`: computes pseudo-activities depending on kinetic parameters and equilibrium constants
- `alpha_ij`: computes redistribution coefficients
- `coef_rep`: computes response coefficients
- `coef_sel.continue`: computes selection coefficient from expression $s_i = R_i\delta_i/E_i$
- `coef_sel.discrete`: computes selection coefficient from expression $s_i = (J^m - J^r)/J^r$
- `compute.delta`: computes the actual effect δ of a mutation
- `droite_e`, `droite_E.CR`, `droite_E.Reg` and `droite_tau`: gives respectively relative concentrations **e**, concentrations **E** (competition + regulation), concentrations **E** (regulation only) and driving variable *tau* when there is regulation
- `range_delta`: bounds of δ_i for mutant concentrations *E_j* between 0 and *Etot*
- `range_tau`: bounds of driving variable *tau* for relative concentrations between 0 and 1
- `name.correl`: gives abbreviation of constraint names

Equilibrium

Functions used to find equilibrium of relative concentrations.

- `predict_th`: computes theoretical equilibrium
- `predict_eff`: computes effective equilibrium
- `predict_eff_allE0`: computes effective equilibrium for various initial concentrations

Graphics

Functions for drawing figures.

- `flux.dome.graph3D`: gives dome of flux in a 3D-plot
- `flux.dome.projections`: gives projection on plane of relative concentrations of the flux dome in a triangular plot
- `graph.simul.by.time.by.enz` and `graph.simul.by.time.by.sim`: illustrations of simulation results. Give different plots with time in x-axis, and color pattern depends on enzymes and simulation numbers respectively
- `graph.simul.by.time.RNV`: various plots of RNVs, computing from simulation results
- `graph.simul.triangle.diagram.e`: gives a triangular diagram of relative concentrations from simulation results
- `RNV.graph.double.at.eq`: RNV size and relative concentrations depending on activities at equilibrium

Simulation

Functions for simulations of enzyme evolution.

- `simul.evol.enz.one`: evolution of enzyme concentrations (one simulation)
- `simul.evol.enz.multiple`: evolution of enzyme concentrations (multiple simulations)

RNV

Functions for computing *Range of Neutral Variation* of enzyme concentrations.

- `RNV.delta.all.enz`: computes δ_i at limits of neutral zone for each enzyme
- `RNV.for.simul`: computes RNV in the simulations
- `RNV.mean.simul`: computes mean of RNV size in the simulations
- `RNV.ranking.order.factor`: gives the name and the value of of the factor that influences the ranking order of RNV
- `RNV.size.at.equilibr`: computes RNV at equilibrium, then plots RNV size against the ranking-order factor
- `graph.simul.by.time.RNV`: various plots of RNVs, computing from simulation results

Informations about *Range of Neutral Variations* (RNV)

The *Range of Neutral Variations* (RNV) are mutant concentration values such as coefficient selection is between $1/(2N)$ and $-1/(2N)$.

Inferior (resp. superior) bound of RNV corresponds to selection coefficient equal to $-1/(2N)$ (resp. $1/(2N)$).

Depending on applied constraint `correl_fun`, it exists 1 or 2 RNV. In case of independence (`"SC"`) or positive regulation between all enzymes (`"RegPos"`), flux has no limit and there is only one RNV. In other cases (competition and/or negative regulation), because flux can reach a maximum, there is two RNV: a "near" one, for small mutations, and a "far" one for big mutations that put mutants on the other side of flux dome.

The RNV size is the absolute value of $\delta_i^s up$ minus $\delta_i^i nf$. If there is no superior bounds but there is two RNVs, RNV size is obtained by the difference of the two $\delta_i^i nf$.

Regulation groups

In version 2.0.0 and more, some functions have been added or modified to take account of regulation groups. Most important new functions are listed below, with function section in parenthesis:

- `class_group` (basic): classifies enzymes in regulation groups from the co-regulation matrix
- `group_types` (basic): gives types of regulation groups, aka negative group, positive group or singleton, from co-regulation matrix
- `predict_grp` (equilibrium): computes equilibrium for the different relative concentrations when there are regulation groups
- `apparent.activities.Aq` (equilibrium): computes apparent activities at equilibrium for regulation groups

- graph.simul.group (graphics): gives graphics of the different kind of relative enzyme concentrations through time when there are regulation groups
- extract.tabEtot (simulation): extracts table of enzyme concentration from simulation results and computes sum of concentrations in a group. Useful to compute the different kind of relative concentrations.

Other graphics function have been modified to take account of equilibrium for regulation groups.

Two new datasets have been added, to have examples when there are regulation groups.

However, RNV.size.at.equilibr and RNV.ranking.order.factor have not been modified, as we cannot determine the relative RNV at equilibrium where there are regulation groups. Moreover, simulations simul.evol.enz.one do not compute equilibrium when there are regulation groups.

References

Coton, C., Talbot, G., Le Louarn, M., Dillmann, C., de Vienne, D., 2021. Evolution of enzyme levels in metabolic pathways: A theoretical approach. bioRxiv 2021.05.04.442631. https://doi.org/10.1101/2021.05.04.442631

Version 2: Coton, C., Dillmann, C., de Vienne, D., 2021. Evolution of enzyme levels in metabolic pathways: A theoretical approach. Part 2.

---

simul.evol.enz.multiple

Multiple simulations of enzyme evolution

Description

This function gives multiple simulations of evolution of enzyme concentrations under constraints

Usage

```
simul.evol.enz.multiple(E_ini_fun, kin_fun, Keq_fun, nsim,
N_fun, correl_fun, beta_fun=NULL, X_fun=1, pasobs=250, npt=500,
same.E0=TRUE, is.random.E0=FALSE, same.kin0=TRUE, is.random.kin0=FALSE,
Etot_fun=100, kin_max=1000, max_mut_size_E=1, max_mut_size_A=1,
pmutA=0, typ_E=1, typ_A=1, use.old.mut=FALSE)
```

Arguments

E_ini_fun	Numeric vector of the initial concentrations
kin_fun	Numeric vector of the initial kinetic parameters
Keq_fun	Numeric vector of equilibrium constants
nsim	Numeric. Number of simulations
N_fun	Numeric. Population size
correl_fun	Character string indicating the abbreviation of the constraint applied on the system

beta_fun	Matrix of co-regulation coefficients
X_fun	Numeric. Numerator of function flux. Default is 1
pasobs	Numeric. Number of time steps between two successive observations of the system. Default is 250
npt	Numeric. Number of observations. Default is 500
same.E0, same.kin0	Logical. Uses same E_ini_fun / kin_fun for all simulations ? Default is TRUE.
is.random.E0, is.random.kin0	Logical. Is E_ini_fun / kin_fun chosen randomly ? Default is FALSE.
Etot_fun	Numeric. Total concentration if is.random.E0 is TRUE. Default is 100.
kin_max	Numeric. Maximal value for random kin_fun. Default is 1000.
max_mut_size_E	Numeric. Maximum absolute size of mutation for enzyme concentrations. Default is 1
max_mut_size_A	Numeric. Maximum absolute size of mutation for kinetic parameters. Default is 1
pmutA	Numeric. Mutation probability of kinetic parameters. Higher pmutA, higher the mutation probability of kinetic parameters compared to enzyme concentrations. Default is 0, i.e. no mutation of kinetic parameters
typ_E	Numeric for mutation method. Authorized values: 1 or 2. Default is 1.
typ_A	Numeric for mutation method. Default is 1. *See details in* mut.kin.
use.old.mut	Logical. If FALSE (default), use mut.E.direct mutation method, else use mut.E.old mutation method if TRUE

Details

Details about how does simulations work are in function simul.evol.enz.one documentation.

Apply would also work, but only for multiple values of E_ini_fun. simul.evol.enz.multiple *gives all results in a same table, and has the possibility to change kinetic parameters between simulation.*

Initial parameters

You can choose if initial concentrations and initial kinetic parameters are identical between all simulations by setting same.E0 and same.kin0.

You can choose randomly initial concentrations and initial kinetic parameters by setting is.random.E0 and is.random.kin0.

Four cases are possible. Examples are given for initial concentrations, but are the same for kinetic parameters, where same.E0, is.random.E0 and E_ini_fun are respectively same.kin0, is.random.kin0 and kin_fun.

- same.E0 is TRUE and is.random.E0 is FALSE is default parameters. In this case, input E_ini_fun of length n is used for all simulations.
- If same.E0 and is.random.E0 are TRUE, E_ini_fun is chosen randomly and used for all nsim simulations.
- If same.E0 is FALSE and is.random.E0 TRUE, each simulation has different random initial concentrations.

- If same.E0 and is.random.E0 are FALSE, a matrix of n columns and nsim rows is needed for E_ini_fun, each row corresponding to one simulation.

$list_init is the return for initial concentrations, kinetic parameters and activities.

Other details

Note that n is the number of enzyme and is determined by length of Keq_fun.

Value

Invisible list of 6 elements:

- $tabR: dataframe of nsim x npt rows and 3*n+4 columns. Each row corresponds to state at each observation step (i.e. after pasobs mutations), and columns are respectively concentrations (1:n), kinetic parameters (n+1:2n), total concentration (2n+1), total kinetic (2n+2), flux/fitness (2n+3), activities (2n+4:3n+3), and simulation number ($sim or 3n+4);
- $tabP_e: numeric matrix of npt rows and n+1 columns, corresponding to relative concentrations at equilibrium (column 1 to n) for each observation step (in rows), plus column $sim;
- $tabP_r: same as $tabP_e, but for response coefficients
- $list_init: list of 3 elements, containing initial values of concentrations in $E0, kinetic parameters in $kin0 and activities in $A0 for each simulation. Each element is a numeric matrix of nsim rows (one by simulation) and n columns (one by enzyme);
- $list_final: list of 3 elements, containing final values of concentrations in $E_f, kinetic parameters in $kin_f and activities in $A_f for each simulation. Each element is a numeric matrix of nsim rows and n columns;
- $param: list of input parameters.

Note that n is the number of enzymes, which is the length of E_ini_fun.

See Also

See function simul.evol.enz.one to see how works a simulation.

Examples

```
# case for 3 enzymes
n <- 3
E0 <- c(30,30,30)
kin <- c(1,10,30)
Keq <- c(1,1,1)
nsim <- 2 # 2 simulations
N <- 1000
beta <- diag(1,n)
beta[upper.tri(beta)] <- c(0.32,0.32*(-0.43),-0.43)
#beta_12 = 0.32, beta_13 = beta_12 x beta_23, beta_23 = -0.43
t_beta <- t(beta) #because R fills matrix column by column
beta[lower.tri(beta)] <- 1/t_beta[lower.tri(t_beta)] #beta_ji = 1/beta_ij
if (n==3) {beta[lower.tri(beta)] <- 1/beta[upper.tri(beta)]} #only available if n=3
```

```
correl <- "RegNeg"

evol_sim <- simul.evol.enz.multiple(E0,kin,Keq,nsim,N,correl,beta)
evol_sim$tabR[,1:n] #concentrations
evol_sim$tabR[,(n+1):(2*n)] #kinetic parameters
evol_sim$tabR[,(2*n+1)] #total concentration
evol_sim$tabR[,(2*n+2)] #total kinetic
evol_sim$tabR[,(2*n+3)] #flux
evol_sim$tabR[,(2*n+4):(3*n+3)] #activities
evol_sim$tabR$sim #simulation number
evol_sim$tabR[evol_sim$tabR$sim==2,] #results for 2nd simulation
```

`simul.evol.enz.one` *Simulation of enzyme evolution*

Description

This function simulates evolution of enzyme concentrations under constraints

Usage

```
simul.evol.enz.one(E_ini_fun, kin_fun, Keq_fun, N_fun, correl_fun,
beta_fun=NULL, X_fun=1, pasobs=250, npt=500, max_mut_size_E=1, max_mut_size_A=1,
pmutA=0, typ_E=1, typ_A=1, use.old.mut=FALSE)
```

Arguments

`E_ini_fun`	Numeric vector of the initial concentrations
`kin_fun`	Numeric vector of the initial kinetic parameters
`Keq_fun`	Numeric vector of equilibrium constants
`N_fun`	Numeric. Population size
`correl_fun`	Character string indicating the abbreviation of the constraint applied on the system
`beta_fun`	Matrix of co-regulation coefficients
`X_fun`	Numeric. Numerator of function `flux`. Default is 1
`pasobs`	Numeric. Number of time steps between two successive observations of the system. Default is `250`
`npt`	Numeric. Number of observations. Default is `500`
`max_mut_size_E`	Numeric. Maximum absolute size of mutation for enzyme concentrations. Default is 1
`max_mut_size_A`	Numeric. Maximum absolute size of mutation for kinetic parameters. Default is 1

pmutA	Numeric. Mutation probability of kinetic parameters. Higher pmutA, higher the mutation probability of kinetic parameters compared to enzyme concentrations. Default is 0, i.e. no mutation of kinetic parameters
typ_E	Numeric for mutation method. Authorized values: 1 or 2. Default is 1.
typ_A	Numeric for mutation method. Default is 1. *See details in* mut.kin.
use.old.mut	Logical. If FALSE (default), use mut.E.direct mutation method, else use mut.E.old mutation method if TRUE

Details

Time step is defined between appearance of two mutation.

Time step is simulated with function simul.next.resident, which gives values of resident after a time step.

To reduce size of result matrix, certain results only are conserved. State of simulation is observed every pasobs time steps. There is npt observations, so result matrix has npt rows. In total, a simulation includes pasobs*npt time steps. By default, there is 125 000 time steps.

Chosen equilibrium for pred_enzsim and pred_contsim are the theoretical equilibrium for constraints "SC", "Comp" and "RegPos", and the effective one for constraints "RegNeg", "CRPos" and "CRNeg". If there are regulation groups (1<sum(1/B)<n), the equilibrium is not computed, and pred_ returns NA.

Value

Invisible list of 4 elements:

- $res_sim: numeric matrix of npt rows and 3*n+3 columns. Each row corresponds to state at each observation step (i.e. after pasobs mutations), and columns are respectively concentrations (1:n), kinetic parameters (n+1:2n), total concentration (2n+1), total kinetic (2n+2), flux/fitness (2n+3), activities (2n+4:3n+3);
- $pred_enzsim: numeric matrix of npt rows and n columns, corresponding to relative concentrations at equilibrium, for each observation step (in rows);
- $pred_contsim: same as $pred_enzsim, but for response coefficients
- $param: list of input parameters:
  - n: number of enzymes,
  - E0: numeric vector of initial concentrations,
  - kin: numeric vector of initial kinetic parameters,
  - Keq: numeric vector of constant equilibrium,
  - beta: matrix of co-regulation coefficients,
  - B: numeric vector of global co-regulation coefficients,
  - correl: character string indicating the constraint abbreviation,
  - N: population size,
  - pasobs: number of steps between two system observations,
  - npt: number of system observations,
  - X: parameter for flux computation,

- pmutA: probability for activity mutation,
- other input parameters

Note that n is the number of enzymes, which is the length of E_ini_fun.

See Also

See function simul.next.resident to see how works each time steps.

Use function simul.evol.enz.multiple to compute several simulations.

---

simul.evol.graph.methods

Graphic methods for simulations of enzyme evolution

Description

Graphics illustrating enzyme evolution simulations obtained by function simul.evol.enz.multiple.

Function graph.simul.by.time.by.sim gives graphics depending on time, colored by simulations.

Function graph.simul.by.time.by.enz gives graphics depending on time, colored by enzymes, with series of graphics for each simulation.

Function graph.simul.others.by.sim gives different graphics depending on other variables than time in x-axis.

Function graph.simul.by.time.RNV gives graphics of *Range of Neutral Variation* (RNV) for each enzyme.

Function graph.simul.group gives graphics depending on time, colored by simulations, specifically for regulation groups.

Usage

```
graph.simul.by.time.by.sim(all_res_sim,new.window=FALSE,add.eq=TRUE,which.sim=NULL,
gr.J.time=FALSE,gr.e.time=TRUE,gr.E.time=FALSE,gr.Etot.time=FALSE,
gr.kin.time=FALSE,gr.A.time=FALSE,gr.tau.time=FALSE,
lwd.eq=1.5,...)

graph.simul.by.time.by.enz(all_res_sim,new.window=FALSE,add.eq=TRUE,which.sim=NULL,
gr.J.time=TRUE,gr.e.time=TRUE,gr.E.time=FALSE,gr.Etot.time=FALSE,
gr.kin.time=FALSE,gr.A.time=FALSE,gr.tau.time=FALSE,
gr.rep.time=FALSE,gr.sim.heading=FALSE,lwd.eq=1.5,...)

graph.simul.others.by.sim(all_res_sim,new.window=FALSE,add.eq=TRUE,which.sim=NULL,
gr.Ef.E0=FALSE, gr.Af.A0=FALSE, gr.Ef.Af=FALSE, gr.J.A.E=TRUE,
gr.J.e=FALSE, gr.J.E=FALSE, env.curve=FALSE, gr.J.A=FALSE, ...)

graph.simul.by.time.RNV(all_res_sim,new.window=FALSE,add.eq=TRUE,which.sim=NULL,
```

```
gr.RNV.E=TRUE,gr.RNV.size=FALSE,gr.RNV.delta=FALSE,
gr.RNV.J=FALSE,zoom.RNV.J=NULL,gr.sim.heading=FALSE,
col_RNV=c("grey60","grey80"),lty_RNV=c("dashed","longdash"),lwd.eq=1.5,...)

graph.simul.group(all_res_sim,new.window=FALSE,add.eq=TRUE,which.sim=NULL,which.grp=NULL,
gr.eiq.time=TRUE,gr.eq.time=TRUE,gr.ei.time=FALSE,gr.Eq.time=FALSE,gr.Ei.time=FALSE,
gr.tauq.time=FALSE,lwd.eq=1.5,...)
```

Arguments

`all_res_sim`	List, the output of function `simul.evol.enz.multiple` (results of evolution simulation).
`new.window`	Logical. Do graphics appear in a new window?
`add.eq`	Logical. Do equilibrium appear on graph?
`which.sim`	Numeric vector containing integer numbers between 1 and `nsim`. Which simulations would you represent? If `NULL` (default), all simulations would be represented.
`gr.J.time, gr.e.time, gr.E.time, gr.Etot.time, gr.kin.time, gr.A.time`	Logical. Add graph flux `J` / relative concentrations `e` / absolute concentrations `E` / total concentration `Etot` / kinetic parameters `kin` / activities `A` in relation to time?
`gr.tau.time`	Logical. Add graph depending on driving variable τ if exists?
`lwd.eq`	Numeric. Line width for equilibrium line only.
`...`	Arguments to be passed in `plot` function, such as `lwd` or `cex`.
`gr.rep.time`	Logical. Add graph response coefficients in relation to time?
`gr.sim.heading`	Logical. Add an heading before each series of graphics corresponding to current simulation?
`gr.Ef.E0`	Logical. Add graph of final concentrations (absolute E and relative e) depending its initial value?
`gr.Af.A0`	Logical. Add graph of final activities (resp. kinetic parameters) depending its initial value?
`gr.Ef.Af`	Logical. Add graph of final concentrations (absolute E and relative e) depending on final activities A?
`gr.J.A.E`	Logical. Add 3D-graph of flux J depending on concentrations E and activities A?
`gr.J.e`	Logical. Add graph of flux J depending on *relative* concentrations e?
`gr.J.E`	Logical. Add graph of flux J depending on *absolute* concentrations E?
`env.curve`	Logical. Add envelope curve of competition dome? Available only for `gr.J.E=T` or `gr.J.e=T`.
`gr.J.A`	Logical. Add graph of of flux J depending on activities A?
`gr.RNV.E, gr.RNV.size, gr.RNV.delta`	Logical. Add graph concentrations `E` with its RNV / RNV size and RNV size divided by Etot / apparent mutation effect `delta` corresponding to RNV in relation to time?

`gr.RNV.J` Logical. Add graph of flux depending on time and depending on concentrations with RNV and neutral zone?

`zoom.RNV.J` Numeric vector of length 2, corresponding to `ylim` of graphics flux with RNV. If `NULL` (by default), `ylim` is zoomed around maximal flux of current simulation. If it is nor `NULL` nor vector of length 2, there is no zoom on flux.

`col_RNV, lty_RNV`
Vector of length 2, for color (resp. lty, see plot function) of RNV lines. First element correspond to inferior bounds and second one to superior bounds of RNV.

`which.grp` Numeric vector containing integer numbers between 1 and p (number of regulation groups). Which regulation groups would you represent? If `NULL` (default), all groups would be represented.

`gr.eiq.time, gr.eq.time, gr.ei.time, gr.Eq.time, gr.Ei.time, gr.tauq.time`
Logical. Add graph intra-group relative concentrations e_i^q / inter-group relative concentrations e^q / total relative concentrations e_i / group absolute concentrations E^q / absolute concentrations E_i / group driving variable τ^q in relation to time?

Details

If only one simulation may be represented, use preferably function `graph.simul.by.time.by.enz`.

Colors for simulations are taken in palette `rainbow`. Colors for enzymes correspond to their number plus one.

Function `graph.simul.by.time.by.sim` gives graphs of flux, relative concentrations, absolute concentrations, total concentration, kinetic parameters and activities through time. In addition, if all enzymes are co-regulated, gives also driving variable τ in relation to time. *Lines are colored according to simulation.* There is one graph by enzyme if necessary. Dashed lines correspond to theoretical equilibrium, and dotted lines to effective equilibrium. Every graph follow same scheme:

1. empty graph with time in x-axis and interesting variable in y-axis
2. for each simulation *i*
3. add connected points for current variable for simulation *i*
4. add text for simulation number *i* at end of x-axis
5. eventually, add predicted values

Function `graph.simul.by.time.by.enz` gives graphs of flux, relative concentrations, absolute concentrations, total concentration, kinetic parameters, activities and response coefficients through time. In addition, if all enzymes are co-regulated, gives also driving variable τ in relation to time and flux in relation to τ. *Lines are colored according to enzyme.* There is one graph by simulation. An heading with parameters of current simulation can be added with `gr.sim.heading` Dashed lines correspond to theoretical equilibrium, and dotted lines to effective equilibrium.

1. for each simulation *i*
2. line graph with time in x-axis and interesting variable in y-axis
3. eventually, add predicted values
4. add legend

Function `graph.simul.others.by.sim` gives graphs of:

- final concentrations in relation to initial concentrations
- final relative concentrations in relation to initial relative concentrations
- final kinetic parameters in relation to initial kinetic parameters
- final activities in relation to initial activities
- final concentrations in relation to final activities
- final relative concentrations in relation to final activities
- flux in relation to concentrations and activities (3D-graph)
- flux in relation to absolute or relative concentrations (one graph by enzyme, colored by simulation)

One color by enzyme. The colored numbers correspond to the simulations.

Function `graph.simul.by.time.RNV` gives graphs of, for each simulations:

- concentrations with RNV bounds
- apparent mutation effects δ at RNV bounds, for each enzyme considering at mutant
- RNV size
- RNV size divided by total concentration
- flux with neutral zone bounds, in relation to time and in relation to enzyme concentrations of each enzyme (where data are ordered to facilitate view)

Lines are colored by enzymes. Bounds of RNV is colored depending on `col_RNV`.

#' **Function** `graph.simul.group` gives graphs of:

- intra-group relative concentrations e_i^q
- inter-group relative concentrations e^q
- total relative concentrations e_i (same as `gr.e.time` in `graoh.simul.by.time.by.sim`)
- absolute concentrations for a group E^q
- absolute concentrations E_i (same as `gr.E.time` in `graoh.simul.by.time.by.sim`)
- driving variable of group τ^q

Lines are colored by simulations.

Graphical parameters

To modify line width, input both `lwd` and `lwd.eq`. Input `lwd` without input `lwd.eq` modifies only equilibrium line width, and not all line width.

Envelope curve

The envelope curve is the projection of competition dome in graph of flux J depending on concentrations (`gr.J.E=TRUE` and `gr.J.e=TRUE`). This curve is available only if there is competition, if the total concentration and activities are identical between simulations, and activities are not subject yo mutations.

col_RNV and lty_RNV

Vector of length 2, for color (resp. lty, see `plot` function) of RNV lines. First element correspond to inferior bounds and second one to superior bounds of RNV.

These parameters are only available for plot `gr.RNV.E` and `gr.RNV.J`.

Value

Function `graph.simul.by.time.by.sim` returns invisible list of 5 elements:

- `$eq_th_e`: Numeric matrix of `n` columns and `nsim` rows. Every row corresponds to relative concentrations at theoretical equilibrium computed from initial values of current simulation;
- `$eq_th_r`: Same structure, for response coefficients at theoretical equilibrium;
- `$eq_eff_e`: Same structure, for relative concentrations at effective equilibrium (if exists), else `NA`;
- `$eq_eff_E`: Same structure, for absolute concentrations at effective equilibrium (if exists), else `NA`;
- `$eq_eff_tau`: Numeric matrix of one `column` and `nsim` rows, corresponding to driving variable τ at effective equilibrium in case of regulation, else `NULL`.

Function `graph.simul.by.time.by.enz` returns nothing.

Function `graph.simul.others.by.sim` returns nothing.

Function `graph.simul.by.time.RNV` returns invisible list of 2 elements:

- `$RNV_all_sim`: List of `nsim` elements (which is number of simulation). Every element is the output of function `RNV.for.simul` for corresponding simulation. If simulation i is not contained in `which.sim`, `$RNV_all_sim[[i]]` is `NULL`.
- `$RNV_mean_size`: Numeric matrix of `n+2` columns. Row number is between `nsim` and `2*nsim`, depending on applied constraint. `n` first columns correspond to RNV mean size for corresponding enzyme for last half simulation; column `n+1` indicates simulation number and column `n+2` RNV number (between 1 and 2).

Function `graph.simul.group` returns an invisible list of `nsim` elements. Each element contains the output of `predict_grp`, which computes the equilibria, for the corresponding simulation.

See Also

Use function `simul.evol.enz.multiple` to simulate enzyme evolution.

Function `scatterplot3d` is used to make the 3D-graph in function `graph.simul.others.by.sim`.

Examples

```
  # With saved simulation
data(data_sim_RegNeg)

graph.simul.by.time.by.sim(data_sim_RegNeg,new.window=TRUE)
graph.simul.by.time.by.enz(data_sim_RegNeg,new.window=TRUE,which.sim=c(1))
graph.simul.others.by.sim(data_sim_RegNeg,new.window=TRUE,env.curve=TRUE,gr.J.E)
graph.simul.by.time.RNV(data_sim_RegNeg,new.window=TRUE,which.sim=c(1))

data(data_sim_CRNeg_1grpNeg1sgl)
graph.simul.group(data_sim_CRNeg_1grpNeg1sgl,gr.Eq.time=TRUE,gr.tauq.time=TRUE)
```

```
 #New simulation
# case for 3 enzymes
n <- 3
E0 <- c(30,30,30)
kin <- c(1,10,30)
Keq <- c(1,1,1)
nsim <- 2 # 2 simulations
N <- 1000
beta <- diag(1,n)
beta[upper.tri(beta)] <- c(0.32,0.32*(-0.43),-0.43)
#put : beta_12 = 0.32, beta_13 = beta_12 x beta_23, beta_23 = -0.43
t_beta <- t(beta) #because R fills matrix column by column
beta[lower.tri(beta)] <- 1/t_beta[lower.tri(t_beta)] #beta_ji = 1/beta_ij
if (n==3) {beta[lower.tri(beta)] <- 1/beta[upper.tri(beta)]} #only available if n=3
correl <- "RegNeg"

evol_sim <- simul.evol.enz.multiple(E0,kin,Keq,nsim,N,correl,beta,npt=250)
graph.simul.by.time.by.sim(evol_sim,new.window=TRUE)
graph.simul.by.time.by.enz(evol_sim,new.window=TRUE,which.sim=c(1))
graph.simul.others.by.sim(evol_sim,new.window=TRUE)
graph.simul.by.time.RNV(evol_sim,new.window=TRUE,which.sim=c(1))
```

`simul.next.resident` *Time step simulation (next resident values)*

Description

`simul.next.resident` simulates a time step, i.e. computes effects of a new mutation and gives next resident values whatever this mutation is fixed or not

Usage

```
simul.next.resident(E_res_fun, kin_fun, Keq_fun, N_fun,
correl_fun, beta_fun=NULL, X_fun=1, max_mut_size_E=1, max_mut_size_A=1,
pmutA=0, typ_E=1, typ_A=1, use.old.mut=FALSE)
```

Arguments

`E_res_fun`	Numeric vector of enzyme concentrations (resident)

kin_fun	Numeric vector of kinetic parameters (catalytic constant kcat divided by Michaelis constant Km)
Keq_fun	Numeric vector of equilibrium constants
N_fun	Numeric. Population size
correl_fun	Character string indicating the abbreviation of the constraint applied on the system
beta_fun	Matrix of co-regulation coefficients
X_fun	Numeric. Numerator of function flux. Default is 1
max_mut_size_E	Numeric. Maximum absolute size of mutation for enzyme concentrations. Default is 1
max_mut_size_A	Numeric. Maximum absolute size of mutation for kinetic parameters. Default is 1
pmutA	Numeric. Mutation probability of kinetic parameters. Higher pmutA, higher the mutation probability of kinetic parameters compared to enzyme concentrations. Default is 0, i.e. no mutation of kinetic parameters
typ_E	Numeric for mutation method. Authorized values: 1 or 2. Default is 1.
typ_A	Numeric for mutation method. Default is 1. *See details in* mut.kin.
use.old.mut	Logical. If FALSE (default), use mut.E.direct mutation method, else use mut.E.old mutation method if TRUE

Details

This function gives the genotype (enzyme concentrations and activities) and phenotype (flux) values of the next resident in an haploid population after a time step. Here, a time step corresponds to the apparition and fixation (or disappearance) of a mutation targeting one enzyme. Therefore a time step is the interval between appearances of two successive mutations.

This function is used for simulation of enzyme concentration evolution.

Algorithm

1. target enzyme and mutation sign are chosen randomly with a uniform law
2. mutation targets randomly concentration or kinetic parameter depending on pmutA value
3. mutation size is chosen randomly between 0 and max_mut_size_E for concentrations (resp. max_mut_size_A for kinetic parameters), then multiplied by its sign
4. mutation effects on all enzymes are computed with function mut.E.direct (resp. mut.kin). The input mutation method is only available for mut.E.old, but multiplicative mutation method typ_E=2 is not accurate in case of regulation
5. if concentration or kinetic parameter become negative, which is biologically impossible, they are set to 0
6. activities, then flux are computed
7. selection coefficient is also computed, considering flux as fitness
8. fixation probability of this mutation is computed
9. fixation of this mutation is random, depending on this fixation probability: if mutation is fixed, mutant become resident for next step, else resident is unchanged for next step
10. returns value of the resident for next step

Algorithm is also detailed in Coton et al. (2021)

Value

Returns a list of 7 elements:

- `$E_next`: numeric vector (length `n`) of the enzyme concentrations of next resident
- `$kin_next`: numeric vector (length `n`) of the kinetic parameters of next resident
- `$Etot_next`: numeric, the total concentration of next resident
- `$kintot_next`: numeric, the total kinetic of next resident
- `$J_next`: numeric, flux of next resident
- `$size_mut`: numeric, mutation size, even if it is not fixed
- `$target_mut`: numeric, number of enzyme targeted by the mutation

Note that `n` is the number of enzymes, which is the length of `E_res_fun`.

References

Coton et al. (2021)

See Also

See function `mut.E.direct` and `mut.kin` to see how enzymes are mutated.

Fitness is computed with function `flux`.

Examples

```
E <- c(30,30,30)
kin <- c(53/0.29,50/0.78,29)
Keq <- c(1.1e+8,4.9e+3,1.1e+3)
beta <- matrix(c(1,10,5,0.1,1,0.5,0.2,2,1),nrow=3)
correl <- "RegPos"
N <- 1000

simul.next.resident(E, kin, Keq, N, correl, beta, pmutA=0.1)
```

`solv.2dg.polynom` *Real solutions of quadratic equation*

Description

Computes real solutions of quadratic equation

Usage

```
solv.2dg.polynom(a_fun,b_fun,c_fun)
```

Arguments

a_fun	Numeric. The quadratic coefficient applied to x^2
b_fun	Numeric. The linear coefficient applied to x
c_fun	Numeric. The free term

Details

Quadratic equation is a second-degree polynomial equation of type $ax^2 + bx + c = 0$, where a is the quadratic coefficient, b the linear coefficient and c the free term.

Value

Three possible vectors:

- Numeric vector of length 2 if there is two real roots
- Numeric value if there is a double-root
- NULL if there is no real solution

Examples

```
solv.2dg.polynom(3,2,1)
#result : NULL

solv.2dg.polynom(1,2,1)
#result : -1

solv.2dg.polynom(1,0,-1)
#result : c(1,-1)
```

---

sol_eqeff *Expression of null response coefficient*

Description

(Utilitary function). Gives result of a transformed expression of the response coefficient

Usage

```
sol_eqeff(tau_fun,E_ini_fun,A_fun,B_fun,correl_fun)
```

Arguments

tau_fun	Numeric. Driving variable of the system
E_ini_fun	Numeric vector of initial concentrations
A_fun	Numeric vector of activities
B_fun	Numeric vector of global co-regulation coefficients. Same length as E_ini_fun.
correl_fun	Character string indicating the constraint abbreviation. Three constraints are allowed here: "RegNeg", "CRPos" and "CRNeg".

Details

Null this expression corresponds to search the effective equilibrium in case of regulation, with or without competition.

Value

A numeric value

References

Coton et al. (2021)

See Also

See function `predict_eff` to compute the effective equilibrium.

`sumbis` *Sum of vector elements*

Description

`sumbis` computes the sum of a vector

Usage

```
sumbis(v)
```

Arguments

v	Numeric vector

Details

`sumbis` computes the sum of a vector, to avoid some problems in comparison test due to R `sum` function.

This problem can also be avoided by using `all.equal` rather than `all` for example.

`sumbis` is used for compute global co-regulation coefficients in `compute.B.from.beta`.

Value

Numeric corresponding to the sum of all values present in the input vector

Examples

```
  # Computation of co-regulation coefficients
#Number of enzymes
n <- 3
#Matrix of co-regulations (n column and n rows)
beta<-matrix(rep(0,n*n),nrow=n)
#Vector of global co-regulation coefficients (length n)
B <- rep(0,n)

#For each enzyme
for (j in 1:n){
  beta[j,j]<-1
}

#Set of co-regulation coefficients
beta[1,2]<- 0.1
beta[2,3]<- 2.0 #-0.43
beta[1,3]<- beta[1,2]*beta[2,3]
beta[2,1]<- 1/beta[1,2]
beta[3,1]<- 1/beta[1,3]
beta[3,2]<- 1/beta[2,3]

#Computation of global co-regulation coefficients
for (j in 1:n){
  B[j]<-sumbis(beta[j,])
}
#or
apply(beta,1,sumbis)

# result : B = c(1.3, 13.0,  6.5)
```

Index