Package 'GE'

June 11, 2020

Type Package

Title General Equilibrium Modeling

Version 0.1.4

Author LI Wu <liwu@staff.shu.edu.cn>

Maintainer LI Wu <liwu@staff.shu.edu.cn>

Depends CGE, data.tree

Imports DiagrammeR

Description Some tools for developing general equilibrium models and some general equilibrium models. These models can be used for teaching economic theory and are built by the methods of new structural economics (see <https://www.nse.pku.edu.cn/> and LI Wu, 2019, ISBN: 9787521804225, General Equilibrium and Structural Dynamics: Perspectives of New Structural Economics. Beijing: Economic Science Press). The model form and mathematical methods can be traced back to von Neumann, J. (1945, A Model of General Economic Equilibrium. The Review of Economic Studies, 13. pp. 1-9) and Kemeny, J. G., O. Morgenstern and G. L. Thompson (1956, A Generalization of the von Neumann Model of an Expanding Economy, Econometrica, 24, pp. 115-135) et al. By the way, J. G. Kemeny is a co-inventor of the computer language BASIC.

License GPL-2 | GPL-3

Encoding UTF-8

LazyData true

NeedsCompilation no

BugReports https://github.com/LiWuR/CGE/issues

RoxygenNote 7.1.0

Repository CRAN

Date/Publication 2020-06-11 19:00:14 UTC

R topics documented:

`apply_expand.grid` *Applying a Function to All Combinations of the Supplied Vectors*

Description

A wrapper of the functions apply and expand.grid. Returns a data frame of values obtained by applying a function to all combinations of the supplied vectors. Firstly, the function expand.grid will be used for the supplied vectors in ... and we will get a data frame containing one row for each combination of the supplied vectors. Then the function will be applied to each row of the data frame. The values of the data frame will also be included in the returned data frame.

Usage

```
apply_expand.grid(FUN, ...)
```

Arguments

`FUN`	the function to be applied. The argument is a numeric vector.
`...`	numeric vectors.

Value

A data frame.

Examples

```
apply_expand.grid(prod, a = 1:9, b = 1:9)

####
f <- function(x) c(r1 = sum(x), r2 = unname(x["b"] - x["a"]))
apply_expand.grid(f, a = c(1, 2), b = c(3, 4))

####
f <- function(x) list(list(sum(x)), prod(x))
apply_expand.grid(f, a = c(1, 2), b = c(3, 4))

####
f <- function(x){
  result <- SCES_A(alpha = 1, Beta = c(0.5, 0.5), p = c(x["p1"], 1), es = x["es"])
  names(result) <- c("dc1","dc2")
  result
}

apply_expand.grid(f, p1 = seq(0.1, 10, 0.1), es = c(0.3, 0.5, 1))

####
f <- function(x) {
  IT17 <- matrix(c(
    1.47, 6.47, 0.57, 2.51,
    2.18, 76.32, 12.83, 44.20,
    0.82, 19.47, 23.33, 35.61,
    6.53, 13.92, 21.88, 0,
    0.23, 4.05, 6.76, 0,
    -0.34, 6.43, 3.40, 0,
    0.13, 8.87, 10.46, 0
  ), 7, 4, TRUE)

  product.output <- c(11.02, 135.53, 79.23)

  rownames(IT17) <- c("agri", "manu", "serv", "lab", "cap", "tax", "dividend")
  colnames(IT17) <- c("sector.agri", "sector.manu", "sector.serv", "sector.hh")

  ge <- gemInputOutputTable_7_4(
    IT = IT17,
    product.output = product.output,
    supply.labor = x[1],
    supply.capital = x[2]
  )

  ge$p
}

apply_expand.grid(f, supply.labor = seq(10, 20, 10), supply.capital = seq(8, 16, 4))
```

CARA *Constant Absolute Risk Aversion (CARA) Utility Function*

Description

Compute the value and the certainty equivalent of the CARA utility function, i.e. -exp(-gamma*x). In general equilibrium analysis, the CARA utility function has an interval scale like temperature.

Usage

```
CARA(x, gamma, p = rep(1/length(x), length(x)))
```

Arguments

x	a vector of all possible states (e.g. returns).
gamma	the Arrow-Pratt measure of absolute risk aversion.
p	the probability vector. By default, the states are assumed to occur with equal probability.

Value

A list containing the following components:

- u: the utility level.
- CE: the certainty equivalent.

Examples

```
mu <- 5 # mu <- 8
a <- 1
x <- c(mu - a, mu + a)
gamma <- 0.8
mu - CARA(x, gamma)$CE

####
gamma <- 0.8
mu <- 2
sigma <- 2
x <- seq(mu - 5 * sigma, mu + 5 * sigma, length.out = 10000)
# two CE calculation methods for random variables of normal distribution
CARA(x, gamma, dnorm(x, mean = mu, sd = sigma))
mu - gamma * sigma^2 / 2
```

CES *CES Function*

Description

CES function, e.g. alpha * (beta1 * (x1 / theta1)^sigma + beta2 * (x2 / theta2)^sigma)^(1 / sigma).

Usage

```
CES(sigma, alpha, beta, x, theta = NULL)
```

Arguments

`sigma`	the sigma coefficient.
`alpha`	the alpha coefficient.
`beta`	a vector consisting of the beta coefficients.
`x`	a vector consisting of the inputs.
`theta`	a vector consisting of the theta coefficients.

Value

The output or utility level.

Examples

```
CES(1, 1, c(0.4, 0.6), c(1, 1), c(0.4, 0.6))
```

CRRA *Constant Relative Risk Aversion (CRRA) Utility Function*

Description

Compute the value and the certainty equivalent of the CRRA utility function.

Usage

```
CRRA(x, gamma, p = rep(1/length(x), length(x)))
```

Arguments

x	a vector of all possible states (e.g. returns).
gamma	the relative risk aversion coefficient.
p	the probability vector. By default, the states are assumed to occur with equal probability.

Value

A list containing the following components:

- u: the utility level.
- CE: the certainty equivalent.

Examples

```
csv <- 0.05 # coefficient of standard deviation
mu <- 90 # mu <- 100
sigma <- mu * csv
x <- seq(mu - 5 * sigma, mu + 5 * sigma, length.out = 10000)
pd <- dnorm(x, mean = mu, sd = sigma)
gamma <- 0.8
# the ratio of risk premium to expected return (i.e. the relative risk premium).
(mu - CRRA(x, gamma, pd)$CE) / mu

####
df <- apply_expand.grid(
  function(arg) {
    CRRA(arg["x"], arg["gamma"])$u
  },
  x = seq(0.5, 3, 0.1),
  gamma = c(0.5, 1, 2, 3)
)
coplot(result ~ x | as.factor(gamma), data = df)
```

`demand_coefficient` *Compute Demand Coefficients of an Agent with a Demand Structural Tree*

Description

Given a price vector, this function computes the demand coefficients of an agent with a demand structural tree. The class of a demand structural tree is Node defined by the package data.tree.

Usage

```
demand_coefficient(node, p)
```

Arguments

`node`	a demand structural tree.
`p`	a price vector with names of commodities.

Details

Demand coefficients often indicate the quantity of various commodities needed by an economic agent in order to obtain a unit of output or utility, and these commodities can include both real commodities and financial instruments such as tax receipts, stocks, bonds and currency.
The demand for various commodities by an economic agent can be expressed by a demand structure tree. Each non-leaf node can be regarded as the output of all its child nodes. Each node can be regarded as an input of its parent node. In other words, the commodity represented by each non-leaf node is a composite commodity composed of the commodities represented by its child nodes. Each non-leaf node usually has an attribute named type. This attribute describes the input-output relationship between the child nodes and the parent node. This relationship can sometimes be represented by a production function or a utility function. The type attribute of each non-leaf node can take the following values.

- SCES. SCES can also be written as CES. In this case, this node also has parameters alpha, beta and es (or sigma = 1 - 1 / es). alpha and es are scalars. beta is a vector. These parameters are parameters of a standard CES function (see `SCES` and `SCES_A`).
- Leontief. In this case, this node also has the parameter a, which is a vector and is the parameter of a Leontief function.
- CD. CD is Cobb-Douglas. In this case, this node also has parameters alpha and beta. These parameters are parameters of a Cobb-Douglas function.
- FIN. That is the financial type. In this case, this node also has the parameter rate or beta. If the parameter beta is not NULL, then the parameter rate will be ignored. The parameter rate applies to all situations, while the parameter beta only applies for some special cases. For FIN nodes, the first child node should represent for a physical commodity or a composite commodity containing a physical commodity, and other child nodes represent for financial instruments. The parameter beta indicates the proportion of each child node's expenditure. The parameter rate indicates the expenditure ratios between financial-instrument-type child

nodes and the first child node. The first element of the parameter rate indicates the amount of the first child node needed to get a unit of output.

- FUNC. That is the function type. In this case, this node also has an attribute named func. The value of that attribute is a function which calculates the demand coefficient for the child nodes. The argument of that function is a price vector. The length of that price vector is equal to the number of the child nodes.

Value

A vector consisting of demand coefficients.

Examples

```
#### a Leontief-type node
dst <- node_new("firm",
  type = "Leontief", a = c(0.5, 0.1),
  "wheat", "iron"
)
print(dst, "type")
node_print(dst)
plot(dst)
node_plot(dst)

demand_coefficient(dst, p = c(wheat = 1, iron = 2)) # the same as a = c(0.5, 0.1)

#### a CD-type node
dst <- node_new("firm",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "wheat", "iron"
)

demand_coefficient(dst, p = c(wheat = 1, iron = 2))
# the same as the following
CD_A(1, c(0.5, 0.5), c(1, 2))

#### a SCES-type node
dst <- node_new("firm",
  type = "SCES",
  alpha = 2, beta = c(0.8, 0.2), es = 0.5,
  "wheat", "iron"
)

demand_coefficient(dst, p = c(wheat = 1, iron = 2))

# the same as the following
SCES_A(alpha = 2, Beta = c(0.8, 0.2), p = c(1, 2), es = 0.5)
CES_A(sigma = 1 - 1 / 0.5, alpha = 2, Beta = c(0.8, 0.2), p = c(1, 2), Theta = c(0.8, 0.2))

#### a FUNC-type node
dst <- node_new("firm",
  type = "FUNC",
  func = function(p) {
    CES_A(
      sigma = -1, alpha = 2,
      Beta = c(0.8, 0.2), p,
      Theta = c(0.8, 0.2)
    )
  },
  "wheat", "iron"
)

demand_coefficient(dst, p = c(wheat = 1, iron = 2))

# the same as the following
CES_A(sigma = -1, alpha = 2, Beta = c(0.8, 0.2), p = c(1, 2), Theta = c(0.8, 0.2))

####
p <- c(wheat = 1, iron = 3, labor = 2, capital = 4)
dst <- node_new("firm 1",
  type = "SCES", sigma = -1, alpha = 1, beta = c(1, 1),
  "cc1", "cc2"
)
node_set(dst, "cc1",
  type = "Leontief", a = c(0.6, 0.4),
  "wheat", "iron"
)
node_set(dst, "cc2",
  type = "SCES", sigma = -1, alpha = 1, beta = c(1, 1),
  "labor", "capital"
)

node_plot(dst)
demand_coefficient(dst, p)

####
p <- c(product = 1, labor = 1, money = 1)
dst <- node_new("firm",
  type = "FIN", rate = c(0.75, 1 / 3),
  "cc1", "money"
) # a financial-type node
node_set(dst, "cc1",
  type = "Leontief", a = c(0.8, 0.2),
  "product", "labor"
)

node_plot(dst)
demand_coefficient(dst, p)

#### the same as above
p <- c(product = 1, labor = 1, money = 1)
dst <- node_new("firm",
  type = "Leontief", a = c(0.8, 0.2),
  "cc1", "cc2"
)
node_set(dst, "cc1",
  type = "FIN", rate = c(0.75, 1 / 3),
  "product", "money"
)

node_set(dst, "cc2",
  type = "FIN", rate = c(0.75, 1 / 3),
  "labor", "money"
)
node_plot(dst)
demand_coefficient(dst, p)

#### the same as above
p <- c(product = 1, labor = 1, money = 1)
dst <- node_new("firm",
  type = "FIN", rate = c(1, 1 / 3),
  "cc1", "money"
) # Financial-type Demand Structure
node_set(dst, "cc1",
  type = "Leontief", a = c(0.6, 0.15),
  "product", "labor"
)

node_plot(dst)
demand_coefficient(dst, p)
```

---

gemCanonicalDynamicMacroeconomic_3_2 *A Canonical Dynamic Macroeconomic General Equilibrium Model (see Torres, 2016)*

Description

A canonical dynamic macroeconomic general equilibrium model (see Torres, 2016, Table 2.1 and 2.2).

Usage

```
gemCanonicalDynamicMacroeconomic_3_2(
  discount.factor = 0.97,
  depreciation.rate = 0.06,
  beta1.firm = 0.35,
  beta1.consumer = 0.4,
  policy.supply = NULL,
  policy.technology = NULL,
  policy.price = NULL,
  ...
)
```

Arguments

`discount.factor`	the intertemporal discount factor.
`depreciation.rate`	the physical depreciation rate of capital stock.
`beta1.firm`	the first beta parameter of the Cobb-Douglas production function.
`beta1.consumer`	the first beta parameter of the Cobb-Douglas utility function. This parameter represents the individual's preferences regarding consumption - leisure decisions.
`policy.supply`	a policy function or a policy function list which adjusts the supplies.
`policy.technology`	a policy function or a policy function list which adjusts the technology.
`policy.price`	a policy function or a policy function list which adjusts the prices.
`...`	arguments to be to be passed to the function sdm2.

Details

A general equilibrium model with 3 commodities (i.e. product, labor, and stock) and 2 agents (i.e. a firm and a consumer). Labor is the numeraire.

Value

A general equilibrium (see `sdm2`)

References

Torres, Jose L. (2016, ISBN: 9781622730452) Introduction to Dynamic Macroeconomic General Equilibrium Models (Second Edition). Vernon Press.

Li Xiangyang (2018, ISBN: 9787302497745) Dynamic Stochastic General Equilibrium (DSGE) Model: Theory, Methodology, and Dynare Practice. Tsinghua University Press. (In Chinese)

Examples

```
gemCanonicalDynamicMacroeconomic_3_2()

#### a market-clearing path (alias instantaneous equilibrium path)
ge <- gemCanonicalDynamicMacroeconomic_3_2(
  policy.price = policyMarketClearingPrice,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 100,
  z0 = c(0.5, 1)
)

par(mfrow = c(1, 2))
matplot(ge$ts.z, type = "b", pch = 20)
matplot(ge$ts.p, type = "b", pch = 20)
```

```
#### technology change in a market-clearing path
policyTechnologyChange <- function(time, dstl) {
  alpha <- 1.2 # The original value is 1.
  time.win <- c(50, 50)
  discount.factor <- 0.97
  depreciation.rate <- 0.06
  beta1.firm <- 0.35
  return.rate <- 1 / discount.factor - 1

  if (time >= time.win[1] && time <= time.win[2]) {
    dstl[[1]]$func <- function(p) {
      result <- CD_A(
        alpha, rbind(beta1.firm, 1 - beta1.firm, 0),
        c(p[1] * (return.rate + depreciation.rate), p[2:3])
      )
      result[3] <- p[1] * result[1] * return.rate / p[3]
      result
    }
  }
}

ge <- gemCanonicalDynamicMacroeconomic_3_2(
  policy.technology = policyTechnologyChange,
  policy.price = policyMarketClearingPrice,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 100,
  z0 = c(0.5, 1)
)

par(mfrow = c(1, 2))
matplot(ge$ts.z, type = "b", pch = 20)
matplot(ge$ts.p, type = "b", pch = 20)

#### an example on page 46 of Li Xiangyang (2018)
ge <- gemCanonicalDynamicMacroeconomic_3_2(
  discount.factor = 0.99,
  depreciation.rate = 0.025,
  beta1.firm = 0.36,
  beta1.consumer = 1
)
```

gemDualLinearProgramming

General Equilibrium Models and Linear Programming Problems (see Winston, 2003)

Description

Some examples illustrating the relationship between general equilibrium problems and (dual) linear programming problems. Some linear programming problems can be transformed into general equilibrium problems and vice versa.

Usage

```
gemDualLinearProgramming(...)
```

Arguments

`...`	arguments to be passed to the function CGE::sdm.

Details

These examples are similar and let us explain briefly the first example (Winston, 2003).
The Dakota Furniture Company manufactures desks, tables, and chairs. The manufacture of each type of furniture requires lumber and two types of skilled labor: finishing and carpentry. The amount of each resource needed to make each type of furniture is as follows:
desk: c(8, 4, 2)
table: c(6, 2, 1.5)
chair: c(1, 1.5, 0.5)
Currently, 48 board feet of lumber, 20 finishing hours, and 8 carpentry hours are available. A desk sells for \$60, a table for \$30, and a chair for \$20. Because the available resources have already been purchased, Dakota wants to maximize total revenue. This problem can be solved by the linear programming method.
Now let us regard the problem above as a general equilibrium problem. The Dakota Furniture Company can be regarded as a consumer who obtains 1 unit of utility from 1 dollar and owns lumber and two types of skilled labor. There are four commodities (i.e. dollar, desk, table and chair) and four agents (i.e. a desk producer, a table producer, a chair producer and the consumer Dakota) in this problem. We need to compute the equilibrium activity levels and the equilibrium prices, which are also the solutions of the (dual) linear programming problems (i.e. the utility-maximizing problem of the consumer and the cost-minimizing problem of the producers).

Value

A general equilibrium.

References

LI Wu (2019, ISBN: 9787521804225) General Equilibrium and Structural Dynamics: Perspectives of New Structural Economics. Beijing: Economic Science Press. (In Chinese)

Winston, Wayne L. (2003, ISBN: 9780534380588) Operations Research: Applications and Algorithms. Cengage Learning.

http://web.mit.edu/15.053/www/AMP-Chapter-04.pdf

https://web.stanford.edu/~ashishg/msande111/notes/chapter4.pdf

https://www.me.utexas.edu/~jensen/ORMM/supplements/methods/lpmethod/S3_dual.pdf

Stapel, Elizabeth. Linear Programming: Introduction. Purplemath. Available from https://www.purplemath.com/modules/lin

Examples

```
#### The Dakota example of Winston (2003, section 6.3, 6.6 and 6.8)
A <- matrix(c(
  0, 0, 0, 1,
  8, 6, 1, 0,
  4, 2, 1.5, 0,
  2, 1.5, 0.5, 0
), 4, 4, TRUE)
B <- matrix(c(
  60, 30, 20, 0,
  0, 0, 0, 0,
  0, 0, 0, 0,
  0, 0, 0, 0
), 4, 4, TRUE)
S0Exg <- {
  S0Exg <- matrix(NA, 4, 4)
  S0Exg[2:4, 4] <- c(48, 20, 8)
  S0Exg
}

## Compute the equilibrium by the function CGE::sdm.
gemDualLinearProgramming(A = A, B = B, S0Exg = S0Exg)

## Compute the equilibrium by the function sdm2.
## The function policyMeanValue is used to accelerate convergence.
ge <- sdm2(
  A = A, B = B, S0Exg = S0Exg,
  policy = policyMeanValue
)

ge$z
ge$p / ge$p[1]

## Compute the general equilibrium above and
## the market-clearing path by the function sdm2.
## Warning: time consuming.
ge2 <- sdm2(
  A = matrix_to_dstl(A), B = B, S0Exg = S0Exg,
  policy = makePolicyStickyPrice(stickiness = 0, tolCond = 1e-4),
  maxIteration = 1,
  numberOfPeriods = 60,
  ts = TRUE
)

matplot(ge2$ts.p, type = "l")
ge2$z
ge2$p / ge2$p[1]

#### An example in the mit reference.
A <- matrix(c(
  0, 0, 0, 1,
  0.5, 2, 1, 0,
  1, 2, 4, 0
), 3, 4, TRUE)
B <- matrix(c(
  6, 14, 13, 0,
  0, 0, 0, 0,
  0, 0, 0, 0
), 3, 4, TRUE)
S0Exg <- {
  S0Exg <- matrix(NA, 3, 4)
  S0Exg[2:3, 4] <- c(24, 60)
  S0Exg
}

ge <- gemDualLinearProgramming(
  A = A, B = B, S0Exg = S0Exg
)

ge$z
ge$p / ge$p[1]

#### An example in the stanford reference.
A <- matrix(c(
  0, 0, 1,
  4.44, 0, 0,
  0, 6.67, 0,
  4, 2.86, 0,
  3, 6, 0
), 5, 3, TRUE)
B <- matrix(c(
  3, 2.5, 0,
  0, 0, 0,
  0, 0, 0,
  0, 0, 0,
  0, 0, 0
), 5, 3, TRUE)
S0Exg <- {
  S0Exg <- matrix(NA, 5, 3)
  S0Exg[2:5, 3] <- 100
  S0Exg
}

ge <- gemDualLinearProgramming(
  A = A, B = B, S0Exg = S0Exg
)

ge$z
ge$p / ge$p[1]

#### An example in the utexas reference.
A <- matrix(c(
  0, 0, 1,
  0, 1, 0,
  1, 3, 0,
  1, 0, 0
), 4, 3, TRUE)
B <- matrix(c(
  2, 3, 0,
  1, 0, 0,
  0, 0, 0,
  0, 0, 0
), 4, 3, TRUE)
S0Exg <- {
  S0Exg <- matrix(NA, 4, 3)
  S0Exg[2:4, 3] <- c(5, 35, 20)
  S0Exg
}

ge <- gemDualLinearProgramming(
  A = A, B = B, S0Exg = S0Exg
)

ge$z
ge$p / ge$p[1]

#### The Giapetto example of Winston (2003, section 3.1)
A <- matrix(c(
  0, 0, 1,
  2, 1, 0,
  1, 1, 0,
  1, 0, 0
), 4, 3, TRUE)
B <- matrix(c(
  27 - 10 - 14, 21 - 9 - 10, 0,
  0, 0, 0,
  0, 0, 0,
  0, 0, 0
), 4, 3, TRUE)
S0Exg <- {
  S0Exg <- matrix(NA, 4, 3)
  S0Exg[2:4, 3] <- c(100, 80, 40)
  S0Exg
}

ge <- sdm2(
  A = A, B = B, S0Exg = S0Exg,
  policy = policyMeanValue,
  numeraire = 1
)

ge$z
ge$p

#### The Dorian example (a minimization problem) of Winston (2003, section 3.2)
A <- matrix(c(
  0, 0, 1,
  7, 2, 0,
  2, 12, 0
), 3, 3, TRUE)
B <- matrix(c(
  28, 24, 0,
  0, 0, 0,
  0, 0, 0
), 3, 3, TRUE)
S0Exg <- {
  S0Exg <- matrix(NA, 3, 3)
  S0Exg[2:3, 3] <- c(50, 100)
  S0Exg
}

ge <- sdm2(
  A = A, B = B, S0Exg = S0Exg,
  policy = policyMeanValue,
  numeraire = 1
)

ge$p
ge$z

#### The diet example (a minimization problem) of Winston (2003, section 3.4)
A <- matrix(c(
  0, 0, 0, 0, 1,
  400, 3, 2, 2, 0,
  200, 2, 2, 4, 0,
  150, 0, 4, 1, 0,
  500, 0, 4, 5, 0
), 5, 5, TRUE)
B <- matrix(c(
  500, 6, 10, 8, 0,
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0
), 5, 5, TRUE)
S0Exg <- {
  S0Exg <- matrix(NA, 5, 5)
  S0Exg[2:5, 5] <- c(50, 20, 30, 80)
  S0Exg
}

ge <- sdm2(
  A = A, B = B, S0Exg = S0Exg,
  policy = policyMeanValue,
  numeraire = 1
)

ge$p
ge$z
```

```
#### An example of Stapel (see the reference):
## Find the maximal value of 3x + 4y subject to the following constraints:
## x + 2y <= 14, 3x - y >= 0, x - y <= 2, x >= 0, y >= 0

A <- matrix(c(
  0, 0, 1,
  1, 2, 0,
  0, 1, 0,
  1, 0, 0
), 4, 3, TRUE)
B <- matrix(c(
  3, 4, 0,
  0, 0, 0,
  3, 0, 0,
  0, 1, 0
), 4, 3, TRUE)
S0Exg <- {
  S0Exg <- matrix(NA, 4, 3)
  S0Exg[2:4, 3] <- c(14, 0, 2)
  S0Exg
}

ge <- sdm2(
  A = A, B = B, S0Exg = S0Exg,
  policy = policyMeanValue,
  priceAdjustmentVelocity = 0.03,
  numeraire = 1
)

ge$z
ge$p
```

gemInputOutputTable_2_8_4
A Two-Country General Equilibrium Model with Money

Description

A two-country general equilibrium model with money. This general equilibrium model is based on a two-country (i.e. CHN and ROW) input-output table. Each country contains four sectors and eight commodities (or subjects). The four sectors are production, consumption, investment and foreign trade. The eight commodities (or subjects) are product, labor, capital goods, bond, tax, dividend, imported product and money interest. Hence the input-output table has 16 rows and 8 columns.

Usage

```
gemInputOutputTable_2_8_4(
  IT,
  product.output.CHN = sum(IT[, "production.CHN"]),
  product.output.ROW = sum(IT[, "production.ROW"]),
  labor.supply.CHN = sum(IT["labor.CHN", ]),
  labor.supply.ROW = sum(IT["labor.ROW", ]),
  capital.supply.CHN = sum(IT["capital.CHN", ]),
  capital.supply.ROW = sum(IT["capital.ROW", ]),
  money.interest.supply.CHN = 5,
  money.interest.supply.ROW = 30,
  es.DIProduct.production.CHN = 0.5,
  es.DIProduct.production.ROW = 0.5,
  es.laborCapital.production.CHN = 0.75,
  es.laborCapital.production.ROW = 0.75,
  es.consumption.CHN = 0.5,
  es.consumption.ROW = 0.5,
  es.investment.CHN = 0.9,
  es.investment.ROW = 0.9,
  interest.rate.CHN = NA,
  interest.rate.ROW = NA,
  return.dstl = FALSE,
  ...
)
```

Arguments

`IT`	the input part of the input-output table (unit: trillion yuan).
`product.output.CHN`	the product output of the production sector of CHN.
`product.output.ROW`	the product output of the production sector of ROW.
`labor.supply.CHN`	the labor supply of CHN.
`labor.supply.ROW`	the labor supply of ROW.
`capital.supply.CHN`	the capital supply of CHN.
`capital.supply.ROW`	the capital supply of ROW.
`money.interest.supply.CHN`	the money interest supply of CHN, that is, the exogenous money supply multiplied by the exogenous interest rate.
`money.interest.supply.ROW`	the money interest supply of ROW.
`es.DIProduct.production.CHN`	the elasticity of substitution between domestic product and imported product used by the production sector of CHN.
`es.DIProduct.production.ROW`	the elasticity of substitution between domestic product and imported product used by the production sector of ROW.

`es.laborCapital.production.CHN`	the elasticity of substitution between labor and capital goods used by the production sector of CHN.
`es.laborCapital.production.ROW`	the elasticity of substitution between labor and capital goods used by the production sector of ROW.
`es.consumption.CHN`	the elasticity of substitution between domestic product and imported product used by the consumption sector of CHN.
`es.consumption.ROW`	the elasticity of substitution between domestic product and imported product used by the consumption sector of ROW.
`es.investment.CHN`	the elasticity of substitution between domestic product and imported product used by the investment sector of CHN.
`es.investment.ROW`	the elasticity of substitution between domestic product and imported product used by the investment sector of ROW.
`interest.rate.CHN`	the interest rate of CHN.
`interest.rate.ROW`	the interest rate of ROW.
`return.dstl`	If TRUE, the demand structure tree will be returned.
`...`	arguments to be transferred to the function `sdm2`.

Details

If interest.rate.CHN is NA or interest.rate.CHN is NA, they are assumed to be equal. And in this case, the exchange rate is determined by the ratio of the interest of unit currency of the two countries. In this model, the ratio of a sector's monetary interest expenditure to its transaction value may not be equal to the interest rate because the ratio is not only affected by the interest rate, but also by the sector's currency circulation velocity and other factors.

Value

A general equilibrium, which usually is a list with the following elements:

- p - the price vector with CHN labor as numeraire, wherein the price of a currency is the interest per unit of currency.
- D - the demand matrix, also called the input table. Wherein the benchmark prices are used.
- DV - the demand value matrix, also called the value input table. Wherein the current price is used.
- SV - the supply value matrix, also called the value output table. Wherein the current price is used.
- eri.CHN - the exchange rate index of CHN currency.
- eri.ROW - the exchange rate index of ROW currency.

- p.money - the price vector with CHN money as numeraire if both interest.rate.CHN and interest.rate.CHN are not NA.
- dstl - the demand structure tree list of sectors if return.dstl == TRUE.
- ... - some elements returned by the function sdm2.

Examples

```
ITExample <- matrix(0, 16, 8, dimnames = list(
  c(
    "product.CHN", "labor.CHN", "capital.CHN", "bond.CHN",
    "tax.CHN", "dividend.CHN", "imported.product.CHN", "money.interest.CHN",
    "product.ROW", "labor.ROW", "capital.ROW", "bond.ROW",
    "tax.ROW", "dividend.ROW", "imported.product.ROW", "money.interest.ROW"
  ),
  c(
    "production.CHN", "consumption.CHN", "investment.CHN", "foreign.trade.CHN",
    "production.ROW", "consumption.ROW", "investment.ROW", "foreign.trade.ROW"
  )
))

production.CHN <- c(
  product.CHN = 140, labor.CHN = 40, capital.CHN = 10,
  tax.CHN = 10, dividend.CHN = 20, imported.product.CHN = 5, money.interest.CHN = 5
)
production.ROW <- c(
  product.ROW = 840, labor.ROW = 240, capital.ROW = 60,
  tax.ROW = 60, dividend.ROW = 120, imported.product.ROW = 6, money.interest.ROW = 30
)

consumption.CHN <- c(
  product.CHN = 40, bond.CHN = 30, imported.product.CHN = 5, money.interest.CHN = 2
)

consumption.ROW <- c(
  product.ROW = 240, bond.ROW = 180, imported.product.ROW = 6, money.interest.ROW = 12
)

investment.CHN <- c(
  product.CHN = 30,
  imported.product.CHN = 4, money.interest.CHN = 1,
  bond.ROW = 1,
  money.interest.ROW = 0.02
)

investment.ROW <- c(
  bond.CHN = 1,
  money.interest.CHN = 0.02,
  product.ROW = 180,
  imported.product.ROW = 4, money.interest.ROW = 6
)
```

```
foreign.trade.CHN <- c(
  product.ROW = 13,
  tax.CHN = 0.65,
  money.interest.ROW = 0.26
)

foreign.trade.ROW <- c(
  product.CHN = 15,
  tax.ROW = 0.75,
  money.interest.CHN = 0.3
)

ITExample <- matrix_add_by_name(
  ITExample, production.CHN, consumption.CHN, investment.CHN, foreign.trade.CHN,
  production.ROW, consumption.ROW, investment.ROW, foreign.trade.ROW
)

ge <- gemInputOutputTable_2_8_4(
  IT = ITExample,
  return.dstl = TRUE
)
ge$eri.CHN
ge$p
node_plot(ge$dstl[[4]])

ge2 <- gemInputOutputTable_2_8_4(
  IT = ge$DV,
  money.interest.supply.CHN = sum(ge$DV["money.interest.CHN", ]),
  money.interest.supply.ROW = sum(ge$DV["money.interest.ROW", ]),
  return.dstl = TRUE
)
ge2$eri.CHN
ge2$p


#### technology progress in CHN
ITTmp <- ITExample
ITTmp["labor.CHN", "production.CHN"] <- ITTmp["labor.CHN", "production.CHN"] * 0.8
geTmp <- gemInputOutputTable_2_8_4(
  IT = ITTmp,
  product.output.CHN = sum(ITExample[, "production.CHN"]),
  return.dstl = TRUE
)
geTmp$eri.CHN


#### increased demand for imported product in CHN
ITTmp <- ITExample
ITTmp["imported.product.CHN", "production.CHN"] <-
  ITTmp["imported.product.CHN", "production.CHN"] * 1.2
geTmp <- gemInputOutputTable_2_8_4(
  IT = ITTmp,
  return.dstl = TRUE
)
geTmp$eri.CHN


#### capital accumulation in CHN
geTmp <- gemInputOutputTable_2_8_4(
  IT = ITExample,
  capital.supply.CHN = sum(ITExample["capital.CHN", ]) * 1.2,
  return.dstl = TRUE
)
geTmp$eri.CHN

##
geTmp <- gemInputOutputTable_2_8_4(
  IT = ITExample,
  capital.supply.CHN = sum(ITExample["capital.CHN", ]) * 1.2,
  es.DIProduct.production.CHN = 0.3,
  return.dstl = TRUE
)
geTmp$eri.CHN
```

gemInputOutputTable_5_4

A General Equilibrium Model based on a 5x4 Input-Output Table (see Zhang Xin, 2017)

Description

This is a general equilibrium model based on a 5x4 input-output table (see Zhang Xin, 2017, Table 8.6.1).

Usage

```
gemInputOutputTable_5_4(
  dstl,
  supply.labor = 850,
  supply.capital = 770,
  names.commodity = c("agri", "manu", "serv", "lab", "cap"),
  names.agent = c("agri", "manu", "serv", "hh")
)
```

Arguments

`dstl`	a demand structure tree list.
`supply.labor`	the supply of labor.

`supply.capital`	the supply of capital.
`names.commodity`	names of commodities.
`names.agent`	names of agents.

Details

Given a 5x4 input-output table (e.g., see Zhang Xin, 2017, Table 8.6.1), this model calculates the corresponding general equilibrium. This input-output table contains 3 production sectors and one household. The household consumes products and supplies labor and capital.

Value

A general equilibrium which is a list with the following elements:

- D - the demand matrix, also called the input table. Wherein the benchmark prices are used.
- DV - the demand value matrix, also called the value input table. Wherein the current price is used.
- SV - the supply value matrix, also called the value output table. Wherein the current price is used.
- ... - some elements returned by the CGE::sdm function

References

Zhang Xin (2017, ISBN: 9787543227637) Principles of Computable General Equilibrium Modeling and Programming (Second Edition). Shanghai: Gezhi Press. (In Chinese)

Examples

```
es.agri <- 0.2 # the elasticity of substitution
es.manu <- 0.3
es.serv <- 0.1

es.VA.agri <- 0.25
es.VA.manu <- 0.5
es.VA.serv <- 0.8

d.agri <- c(260, 345, 400, 200, 160)
d.manu <- c(320, 390, 365, 250, 400)
d.serv <- c(150, 390, 320, 400, 210)
d.hh <- c(635, 600, 385, 0, 0)
# d.hh <- c(635, 600, 100, 0, 0)

D <- cbind(d.agri, d.manu, d.serv, d.hh)

dst.agri <- Node$new("sector.agri",
  type = "SCES", sigma = 1 - 1 / es.agri,
  alpha = 1,
  beta = prop.table(c(
    sum(d.agri[1:3]),
    sum(d.agri[4:5])
  ))
)
dst.agri$AddChild("cc1.agri",
  type = "Leontief",
  a = prop.table(d.agri[1:3])
)$
  AddChild("agri")$AddSibling("manu")$AddSibling("serv")$
  parent$
  AddSibling("cc2.agri",
  type = "SCES", sigma = 1 - 1 / es.VA.agri,
  alpha = 1,
  beta = prop.table(d.agri[4:5])
)$
  AddChild("lab")$AddSibling("cap")

##
dst.manu <- Node$new("sector.manu",
  type = "SCES", sigma = 1 - 1 / es.manu,
  alpha = 1,
  beta = prop.table(c(
    sum(d.manu[1:3]),
    sum(d.manu[4:5])
  ))
)
dst.manu$AddChild("cc1.manu",
  type = "Leontief",
  a = prop.table(d.manu[1:3])
)$
  AddChild("agri")$AddSibling("manu")$AddSibling("serv")$
  parent$
  AddSibling("cc2.manu",
  type = "SCES", sigma = 1 - 1 / es.VA.manu,
  alpha = 1,
  beta = prop.table(d.manu[4:5])
)$
  AddChild("lab")$AddSibling("cap")

##
dst.serv <- Node$new("sector.serv",
  type = "SCES", sigma = 1 - 1 / es.serv,
  alpha = 1,
  beta = prop.table(c(
    sum(d.serv[1:3]),
    sum(d.serv[4:5])
  ))
)
dst.serv$AddChild("cc1.serv",
  type = "Leontief",
  a = prop.table(d.serv[1:3])
)$
  AddChild("agri")$AddSibling("manu")$AddSibling("serv")$
  parent$
```

```
    AddSibling("cc2.serv",
    type = "SCES", sigma = 1 - 1 / es.VA.serv,
    alpha = 1,
    beta = prop.table(d.serv[4:5])
  )$
    AddChild("lab")$AddSibling("cap")

  ##
  dst.hh <- Node$new("sector.hh",
    type = "SCES", sigma = -1,
    alpha = 1,
    beta = prop.table(d.hh[1:3])
  )
  dst.hh$AddChild("agri")$AddSibling("manu")$AddSibling("serv")

  dstl <- list(
    dst.agri,
    dst.manu,
    dst.serv,
    dst.hh
  )

  ge <- gemInputOutputTable_5_4(dstl)

  #### labor supply increase
  geLSI <- gemInputOutputTable_5_4(dstl, supply.labor = 850 * 1.08)
  geLSI$p
  geLSI$z / ge$z
```

gemInputOutputTable_7_4 *A General Equilibrium Model based on a 7x4 Input-Output Table*

Description

This is a general equilibrium model based on a 7x4 input-output table.

Usage

```
gemInputOutputTable_7_4(
  IT,
  product.output,
  supply.labor,
  supply.capital,
  es.agri = 0,
  es.manu = 0,
  es.serv = 0,
  es.hh = 0,
  es.VA.agri = 0.25,
  es.VA.manu = 0.5,
  es.VA.serv = 0.8,
  ...
)
```

Arguments

`IT`	the input part of the input-output table in the base period (unit: trillion yuan).
`product.output`	the outputs of products in the base period.
`supply.labor`	the supply of labor.
`supply.capital`	the supply of capital.
`es.agri, es.manu, es.serv`	the elasticity of substitution between the intermediate input and the value-added input of the agriculture sector, manufacturing sector and service sector.
`es.hh`	the elasticity of substitution among products consumed by the household sector.
`es.VA.agri, es.VA.manu, es.VA.serv`	the elasticity of substitution between labor input and capital input of the agriculture sector, manufacturing sector and service sector.
`...`	arguments to be transferred to the function sdm of the package CGE.

Details

Given a 7x4 input-output table, this model calculates the corresponding general equilibrium. This input-output table contains 3 production sectors and 1 household. The household consumes products and supplies labor, capital, stock and tax receipt. Generally speaking, the value of the elasticity of substitution in this model should be between 0 and 1.

Value

A general equilibrium, which is a list with the following elements:

- p - the price vector with labor as numeraire.
- D - the demand matrix, also called the input table. Wherein the benchmark prices are used.
- DV - the demand value matrix, also called the value input table. Wherein the current price is used.
- SV - the supply value matrix, also called the value output table. Wherein the current price is used.
- IT - the nonstandard input table, wherein the agricultural tax can be negative.
- ITV - the nonstandard value input table, wherein the agricultural tax can be negative.
- value.added - the value-added of the three production sectors.
- dstl - the demand structure tree list of sectors.
- ... - some elements returned by the CGE::sdm function

Examples

```
IT17 <- matrix(c(
  1.47, 6.47, 0.57, 2.51,
  2.18, 76.32, 12.83, 44.20,
  0.82, 19.47, 23.33, 35.61,
  6.53, 13.92, 21.88, 0,
  0.23, 4.05, 6.76, 0,
  -0.34, 6.43, 3.40, 0,
  0.13, 8.87, 10.46, 0
), 7, 4, TRUE)

product.output <- c(11.02, 135.53, 79.23)

rownames(IT17) <- c("agri", "manu", "serv", "lab", "cap", "tax", "dividend")
colnames(IT17) <- c("sector.agri", "sector.manu", "sector.serv", "sector.hh")

ge <- gemInputOutputTable_7_4(
  IT = IT17,
  product.output = product.output,
  supply.labor = 42.33,
  supply.capital = 11.04
)

#### labor supply reduction
geLSR <- gemInputOutputTable_7_4(
  IT = IT17,
  product.output = product.output,
  supply.labor = 42.33 * 0.9,
  supply.capital = 11.04
)

geLSR$z / ge$z
geLSR$p / ge$p

#### capital accumulation
geCA <- gemInputOutputTable_7_4(
  IT = IT17,
  product.output = product.output,
  supply.labor = 42.33,
  supply.capital = 11.04 * 1.1
)

geCA$z / ge$z
geCA$p / ge$p

#### technology progress
IT.TP <- IT17
IT.TP ["lab", "sector.manu"] <-
  IT.TP ["lab", "sector.manu"] * 0.9

geTP <- gemInputOutputTable_7_4(
  IT = IT.TP,
  product.output = product.output,
  supply.labor = 42.33,
  supply.capital = 11.04
)

geTP$z / ge$z
geTP$p / ge$p

##
IT.TP2 <- IT.TP
IT.TP2 ["cap", "sector.manu"] <-
  IT.TP2["cap", "sector.manu"] * 1.02
geTP2 <- gemInputOutputTable_7_4(
  IT = IT.TP2,
  product.output = product.output,
  supply.labor = 42.33,
  supply.capital = 11.04
)

geTP2$z / ge$z
geTP2$p / ge$p

##
IT.TP3 <- IT17
IT.TP3 ["lab", "sector.manu"] <-
  IT.TP3 ["lab", "sector.manu"] * 0.9
IT.TP3 ["lab", "sector.agri"] <-
  IT.TP3 ["lab", "sector.agri"] * 0.8

geTP3 <- gemInputOutputTable_7_4(
  IT = IT.TP3,
  product.output = product.output,
  supply.labor = 42.33,
  supply.capital = 11.04
)

geTP3$value.added / ge$value.added
prop.table(geTP3$value.added) - prop.table(ge$value.added)

#### demand structure change
IT.DSC <- IT17
IT.DSC["serv", "sector.hh"] <- IT.DSC ["serv", "sector.hh"] * 1.2

geDSC <- gemInputOutputTable_7_4(
  IT = IT.DSC,
  product.output = product.output,
  supply.labor = 42.33,
  supply.capital = 11.04
)

geDSC$z[1:3] / ge$z[1:3]
geDSC$p / ge$p
```

```
#### tax change
IT.TC <- IT17
IT.TC["tax", "sector.agri"] <- IT.TC["tax", "sector.agri"] * 2

geTC <- gemInputOutputTable_7_4(
  IT = IT.TC,
  product.output = product.output,
  supply.labor = 42.33,
  supply.capital = 11.04
)

geTC$z / ge$z
geTC$p / ge$p

##
IT.TC2 <- IT17
IT.TC2["tax", "sector.manu"] <- IT.TC2["tax", "sector.manu"] * 0.8

geTC2 <- gemInputOutputTable_7_4(
  IT = IT.TC2,
  product.output = product.output,
  supply.labor = 42.33,
  supply.capital = 11.04
)

geTC2$z / ge$z
geTC2$p / ge$p
```

gemInputOutputTable_8_8

A General Equilibrium Model based on an 8x8 Input-Output Table

Description

This is a general equilibrium model based on a 8x8 input-output table.

Usage

```
gemInputOutputTable_8_8(
  IT,
  OT,
  es.agri = 0,
  es.manu = 0,
  es.serv = 0,
  es.hh = 0,
  es.FT = 0,
  es.VA.agri = 0.25,
  es.VA.manu = 0.5,
  es.VA.serv = 0.8,
  es.prodcctDI = 0.5,
  ...
)
```

Arguments

`IT`	the input part of the input-output table in the base period (unit: trillion yuan).
`OT`	the output part of the input-output table in the base period (unit: trillion yuan).
`es.agri, es.manu, es.serv`	the elasticity of substitution between the intermediate input and the value-added input of the agriculture sector, manufacturing sector and service sector.
`es.hh`	the elasticity of substitution among products consumed by the household sector.
`es.FT`	the elasticity of substitution among exported products.
`es.VA.agri, es.VA.manu, es.VA.serv`	the elasticity of substitution between labor input and capital input of the agriculture sector, manufacturing sector and service sector.
`es.prodcctDI`	the elasticity of substitution between domestic product and imported product.
`...`	arguments to be transferred to the function sdm of the package CGE.

Details

Given an 8x8 input-output table, this model calculates the corresponding general equilibrium. This input-output table contains 3 production sectors, 1 household, 1 foreign trade sector importing agriculture goods, 1 foreign trade sector importing manufacturing goods, 1 foreign trade sector importing service, 1 foreign trade sector importing bond. There are 8 kinds of commodities (or subjects) in the table, i.e. agriculture product, manufacturing product, service, labor, capital goods, tax, dividend and bond of ROW (i.e. the rest of the world). The household consumes products and supplies labor, capital, stock and tax receipt. Generally speaking, the value of the elasticity of substitution in this model should be between 0 and 1.

Value

A general equilibrium, which is a list with the following elements:

- p - the price vector with labor as numeraire.
- D - the demand matrix, also called the input table. Wherein the benchmark prices are used.
- DV - the demand value matrix, also called the value input table. Wherein the current price is used.
- SV - the supply value matrix, also called the value output table. Wherein the current price is used.
- value.added - the value-added of the three production sectors.
- dstl - the demand structure tree list of sectors.
- ... - some elements returned by the CGE::sdm function

Examples

```
IT17 <- matrix(c(
  1.47, 6.47, 0.57, 2.99, 0.12 * 0.60 / (0.60 + 12.10 + 2.23 + 1.45),
  0.12 * 12.10 / (0.60 + 12.10 + 2.23 + 1.45),
  0.12 * 2.23 / (0.60 + 12.10 + 2.23 + 1.45),
  0.12 * 1.45 / (0.60 + 12.10 + 2.23 + 1.45),

  2.18, 76.32, 12.83, 43, 13.30 * 0.60 / (0.60 + 12.10 + 2.23 + 1.45),
  13.30 * 12.10 / (0.60 + 12.10 + 2.23 + 1.45),
  13.30 * 2.23 / (0.60 + 12.10 + 2.23 + 1.45),
  13.30 * 1.45 / (0.60 + 12.10 + 2.23 + 1.45),


  0.82, 19.47, 23.33, 34.88, 2.96 * 0.60 / (0.60 + 12.10 + 2.23 + 1.45),
  2.96 * 12.10 / (0.60 + 12.10 + 2.23 + 1.45),
  2.96 * 2.23 / (0.60 + 12.10 + 2.23 + 1.45),
  2.96 * 1.45 / (0.60 + 12.10 + 2.23 + 1.45),

  6.53, 13.92, 21.88, 0, 0, 0, 0, 0,
  0.23, 4.05, 6.76, 0, 0, 0, 0, 0,
  0, 6.43, 3.40, 0, 0, 0, 0, 0,
  0.13, 8.87, 10.46, 0, 0, 0, 0, 0,
  0, 0, 0, 1.45, 0, 0, 0, 0
), 8, 8, TRUE)

OT17 <- matrix(c(
  11.02, 0, 0, 0, 0.60, 0, 0, 0,
  0, 135.53, 0, 0, 0, 12.10, 0, 0,
  0, 0, 79.23, 0, 0, 0, 2.23, 0,
  0, 0, 0, 42.33, 0, 0, 0, 0,
  0, 0, 0, 11.04, 0, 0, 0, 0,
  0.34, 0, 0, 9.49, 0, 0, 0, 0,
  0, 0, 0, 19.46, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 1.45
), 8, 8, TRUE)

rownames(IT17) <- rownames(OT17) <-
  c("agri", "manu", "serv", "lab", "cap", "tax", "dividend", "bond.ROW")
colnames(IT17) <- colnames(OT17) <- c(
  "sector.agri", "sector.manu", "sector.serv", "sector.hh",
  "sector.FT.agri", "sector.FT.manu", "sector.FT.serv", "sector.FT.bond.ROW"
)


ge <- gemInputOutputTable_8_8(
  IT = IT17,
  OT = OT17
)

#### technology progress
IT.TP <- IT17
IT.TP ["lab", "sector.manu"] <-
  IT.TP ["lab", "sector.manu"] * 0.9

geTP <- gemInputOutputTable_8_8(
  IT = IT.TP,
  OT = OT17
)

geTP$z / ge$z
geTP$p / ge$p
geTP$value.added
prop.table(geTP$value.added) - prop.table(ge$value.added)

#### capital accumulation
OT.CA <- OT17
OT.CA["cap", "sector.hh"] <- OT.CA["cap", "sector.hh"] * 1.1
geCA <- gemInputOutputTable_8_8(
  IT = IT17,
  OT = OT.CA
)

geCA$z / ge$z
geCA$p / ge$p
geCA$p
geCA$value.added
prop.table(geCA$value.added) - prop.table(ge$value.added)

#### tax change
OT.TC <- OT17
OT.TC["tax", "sector.agri"] <- OT.TC["tax", "sector.agri"] * 2

geTC <- gemInputOutputTable_8_8(
  IT = IT17,
  OT = OT.TC
)

geTC$z / ge$z
geTC$p / ge$p

##
IT.TC2 <- IT17
IT.TC2["tax", "sector.manu"] <- IT.TC2["tax", "sector.manu"] * 0.8

geTC2 <- gemInputOutputTable_8_8(
  IT = IT.TC2,
  OT = OT17
)

geTC2$z / ge$z
geTC2$p / ge$p
```

gemInputOutputTable_CES_3_3

A CES-type General Equilibrium Model based on an Input-Output Table.

Description

Given a 3x3 input-output table (e.g., see Zhang Xin, 2017, Table 2.2.2), this model can be used to calculate the corresponding equilibrium. This input-output table contains two firms and one household. The household consumes products and supplies labor.

Usage

```
gemInputOutputTable_CES_3_3(
  input = matrix(c(200, 300, 100, 150, 320, 530, 250, 380, 0), 3, 3, TRUE),
  output = c(600, 1000, 630),
  es = 0
)
```

Arguments

`input`	the input matrix in the base period.
`output`	a vector consisting of the product outputs and labor supply in the base period.
`es`	a scalar, which is the elasticity of substitution between the inputs.

Value

A general equilibrium, which is a list with the following elements:

- p - the price vector with labor as numeraire.
- D - the demand matrix, also called the input table. Wherein the benchmark prices are used.
- DV - the demand value matrix, also called the value input table. Wherein the current price is used.
- SV - the supply value matrix, also called the value output table. Wherein the current price is used.
- ... - some elements returned by the CGE::sdm function

References

Zhang Xin. (2017, ISBN: 9787543227637). Principles of Computable General Equilibrium Modeling and Programming (Second Edition). Shanghai: Gezhi Press. (In Chinese)

Examples

```
x <- 75
gemInputOutputTable_CES_3_3(
  input = matrix(c(
    200, 300, 100,
    x, 320, 530,
    250, 380, 0
  ), 3, 3, TRUE),
  output = c(600, 1000, 630),
  es = 0.5
)
```

---

gemInputOutputTable_easy_5_4

An Easy General Equilibrium Model based on a 5x4 Input-Output Table (see Zhang Xin, 2017)

Description

This is a general equilibrium model based on a 5x4 input-output table (see Zhang Xin, 2017, Table 8.6.1).

Usage

```
gemInputOutputTable_easy_5_4(
  IT = cbind(sector.agri = c(agri = 260, manu = 345, serv = 400, lab = 200, cap = 160),
     sector.manu = c(agri = 320, manu = 390, serv = 365, lab = 250, cap = 400),
   sector.serv = c(agri = 150, manu = 390, serv = 320, lab = 400, cap = 210), sector.hh
     = c(agri = 635, manu = 600, serv = 385, lab = 0, cap = 0)),
  supply.labor = 850,
  supply.capital = 770,
  es.agri = 0.2,
  es.manu = 0.3,
  es.serv = 0.1,
  es.VA.agri = 0.25,
  es.VA.manu = 0.5,
  es.VA.serv = 0.8
)
```

Arguments

`IT`	the input and consumption part of the input-output table.
`supply.labor`	the supply of labor.
`supply.capital`	the supply of capital.
`es.agri, es.manu, es.serv`	the elasticity of substitution between the intermediate input and the value-added input of the agriculture sector, manufacturing sector and service sector.

`es.VA.agri, es.VA.manu, es.VA.serv`
the elasticity of substitution between labor input and capital input of the agriculture sector, manufacturing sector and service sector.

Details

Given a 5x4 input-output table (e.g., see Zhang Xin, 2017, Table 8.6.1), this model calculates the corresponding general equilibrium. This input-output table contains 3 production sectors and one household. The household consumes products and supplies labor and capital.

Value

A general equilibrium, which is a list with the following elements:

- p - the price vector with labor as numeraire.
- D - the demand matrix, also called the input table. Wherein the benchmark prices are used.
- DV - the demand value matrix, also called the value input table. Wherein the current price is used.
- SV - the supply value matrix, also called the value output table. Wherein the current price is used.
- ... - some elements returned by the CGE::sdm function

References

Zhang Xin (2017, ISBN: 9787543227637) Principles of Computable General Equilibrium Modeling and Programming (Second Edition). Shanghai: Gezhi Press. (In Chinese)

Examples

```
sector.agri <- c(260, 345, 400, 200, 160)
sector.manu <- c(320, 390, 365, 250, 400)
sector.serv <- c(150, 390, 320, 400, 210)
sector.hh <- c(635, 600, 100, 0, 0)

IT <- cbind(sector.agri, sector.manu, sector.serv, sector.hh)
rownames(IT) <-  c("agri", "manu", "serv", "lab", "cap")

ge <- gemInputOutputTable_easy_5_4(IT)

####
ge <- gemInputOutputTable_easy_5_4(supply.capital = 1870)
prop.table(ge$z[1:3])
```

`gemInputOutputTable_Leontief_3_3`
A Leontief-type General Equilibrium Model based on a 3x3 Input-Output Table

---

Description

Given a 3x3 input-output table (e.g., see Zhang Xin, 2017, Table 2.2.2), this model can be used to calculate the corresponding equilibrium. This input-output table contains two firms and one household. The household consumes products and supplies labor.

Usage

```
gemInputOutputTable_Leontief_3_3(
  input = matrix(c(200, 300, 100, 150, 320, 530, 250, 380, 0), 3, 3, TRUE),
  output = c(600, 1000, 630)
)
```

Arguments

`input`	the input matrix in the base period.
`output`	a vector consisting of the product outputs and labor supply in the base period.

Value

A general equilibrium, which is a list with the following elements:

- p - the price vector with labor as numeraire.
- D - the demand matrix, also called the input table. Wherein the benchmark prices are used.
- DV - the demand value matrix, also called the value input table. Wherein the current price is used.
- SV - the supply value matrix, also called the value output table. Wherein the current price is used.
- ... - some elements returned by the CGE::sdm function

References

Zhang Xin. (2017, ISBN: 9787543227637). Principles of Computable General Equilibrium Modeling and Programming (Second Edition). Shanghai: Gezhi Press. (In Chinese)

Examples

```
x <- 75
gemInputOutputTable_Leontief_3_3(
  input = matrix(c(
    200, 300, 100,
    x, 320, 530,
    250, 380, 0
  ), 3, 3, TRUE),
  output = c(600, 1000, 630)
)
```

---

gemMoney_3_2 *A General Equilibrium Model with Money*

Description

A general equilibrium model with money.

Usage

```
gemMoney_3_2(
  dstl,
  supply.labor = 100,
  supply.money = 300,
  names.commodity = c("product", "labor", "money"),
  names.agent = c("firm", "household"),
  ...
)
```

Arguments

`dstl`	the demand structure tree list.
`supply.labor`	the supply of labor.
`supply.money`	the supply of money.
`names.commodity`	names of commodities.
`names.agent`	names of agents.
`...`	arguments to be to be passed to the function sdm2.

Details

A general equilibrium model with 3 commodities (i.e. product, labor, and money) and 2 agents (i.e. a firm and a household). To produce, the firm needs product, labor and money. The household only consumes the product. But money is also needed to buy the product. The household supplies labor and money.

In the calculation results, the price of the currency is the interest per unit of currency (i.e. the rent price, interest price). It should be noted that the unit of currency can be arbitrarily selected. For example, a unit of currency may be two dollars or ten dollars. The interest price divided by the interest rate is the asset price of 1 unit of the currency.

Value

A general equilibrium (see sdm2)

Examples

```
#### Leontief-type firm
interest.rate <- 0.25

dst.Leontief.firm <- node_new("output",
  type = "FIN", rate = c(1, interest.rate),
  "cc1", "money"
)
node_set(dst.Leontief.firm, "cc1",
  type = "Leontief", a = c(0.6, 0.2),
  "product", "labor"
)

dst.household <- node_new("utility",
  type = "FIN", rate = c(1, interest.rate),
  "product", "money"
)

dstl.Leontief <- list(dst.Leontief.firm, dst.household)

ge.Leontief <- gemMoney_3_2(dstl.Leontief)
ge.Leontief$p

## CES-type firm
dst.CES.firm <- Clone(dst.Leontief.firm)
node_set(dst.CES.firm, "cc1",
  type = "SCES", a = NULL, alpha = 1, beta = c(0.6, 0.2),
  es = 0 # es is the elasticity of substitution.
)

node_plot(dst.CES.firm)

dstl.CES <- list(dst.CES.firm, dst.household)

ge.CES <- gemMoney_3_2(dstl.CES)
ge.CES$p
p.money <- ge.CES$p
p.money["money"] <- p.money["money"] / interest.rate
p.money <- p.money / p.money["money"] # prices in terms of the asset price of the currency
p.money

## The price of money is the interest rate.
## The other prices are in terms of the asset price of the currency.
gemMoney_3_2(dstl.CES,
             numeraire = c("money" = interest.rate)
)
```

gemQuasilinearPureExchange_2_2 *A Pure Exchange Economy with a Quasilinear Utility Function*

Description

An example of a pure exchange economy with a quasilinear utility function (Karaivanov, see the reference).

Usage

```
gemQuasilinearPureExchange_2_2(
  A,
  Endowment = matrix(c(3, 4, 7, 0), 2, 2, TRUE),
  policy = NULL
)
```

Arguments

`A`	a demand structure tree list, a demand coefficient 2-by-2 matrix (alias demand structure matrix) or a function A(state) which returns a 2-by-2 matrix (see `sdm2`).
`Endowment`	a 2-by-2 matrix.
`policy`	a policy function (see `sdm2`).

Details

Suppose there are only two goods (bananas and fish) and 2 consumers (Annie and Ben) in an exchange economy. Annie has a utility function x_1^(1/3) * x_2^(2/3) where x_1 is the amount of fish she eats and x_2 is the amount of bananas she eats. Annie has an endowment of 3 kilos of fish and 7 bananas. Ben has a utility function x_1 + 1.5 * log(x_2) and endowments of 4 kilos of fish and 0 bananas. Assume the price of bananas is 1. See the reference for more details.

Value

A general equilibrium.

References

http://www.sfu.ca/~akaraiva/CE_example.pdf

Examples

```
demand_consumer2 <- function(p, w) {
  a <- 1.5
  d <- rbind(0, 0)
  w <- w / p[1]
  p <- p / p[1] # normalize the wealth and prices
```

```
    d[2] <- a / p[2]
    if (d[2] * p[2] > w) {
      d[2] <- w / p[2]
      d[1] <- 0
    } else {
      d[1] <- w - d[2] * p[2]
    }

    d
  }

  A <- function(state) {
    a1 <- CD_A(1, rbind(1 / 3, 2 / 3), state$p)
    a2 <- demand_consumer2(state$p, state$w[2])
    cbind(a1, a2)
  }

  ge.mat <- gemQuasilinearPureExchange_2_2(A = A)
  ge.mat

  ## Use a dstl and a policy function to compute the general equilibrium above.
  dst.consumer1 <- node_new("util",
                            type = "CD", alpha = 1, beta = c(1 / 3, 2 / 3),
                            "fish", "banana"
  )
  dst.consumer2 <- node_new("util",
                            type = "Leontief", a = c(1, 1),
                            "fish", "banana"
  )

  dstl <- list(dst.consumer1, dst.consumer2)

  policy.quasilinear <- function(dstl, state) {
    wealth <- t(state$p) %*% state$S
    dstl[[2]]$a <- demand_consumer2(state$p, wealth[2])
  }

  ge.dstl <- gemQuasilinearPureExchange_2_2(
    A = dstl,
    policy = policy.quasilinear
  )
  ge.dstl

  #### Another example. Now Ben has a utility function x_1 + sqrt(x_2).
  demand_consumer2 <- function(p, w) {
    d <- rbind(0, 0)
    w <- w / p[1]
    p <- p / p[1] # normalize the wealth and prices
    d[2] <- (2 * p[2])^-2
    if (d[2] * p[2] > w) {
      d[2] <- w / p[2]
      d[1] <- 0
    } else {
```

```
    d[1] <- w - d[2] * p[2]
  }

  d
}

A <- function(state) {
  a1 <- CD_A(1, rbind(1 / 3, 2 / 3), state$p)
  a2 <- demand_consumer2(state$p, state$w[2])
  cbind(a1, a2)
}

ge.2_2 <- gemQuasilinearPureExchange_2_2(A = A)
ge.2_2

## another computation method for the economy above
A <- function(state) {
  a1 <- CD_A(1, rbind(1 / 3, 2 / 3, 0, 0), state$p)
  a2 <- c(0, 0, 1, 0)
  a3 <- c(1, 0, 0, 0) # firm 1
  a4 <- CD_A(1, rbind(0, 1 / 2, 0, 1 / 2), state$p) # firm 2
  cbind(a1, a2, a3, a4)
}

ge.4_4 <- sdm2(
  A = A,
  B = {
    B <- matrix(0, 4, 4)
    B[3, 3] <- 1
    B[3, 4] <- 1
    B
  },
  S0Exg = {
    S0Exg <- matrix(NA, 4, 4)
    S0Exg[1:2, 1] <- c(3, 7)
    S0Exg[1:2, 2] <- c(4, 0)
    S0Exg[4, 1:2] <- c(0, 1)
    S0Exg
  },
  names.commodity = c("fish", "banana", "util2", "land"),
  names.agent = c("Annie", "Ben", "firm1", "firm2"),
  numeraire = "banana"
)
ge.4_4
```

gemRobinson_3_2 *A Robinson Crusoe Economy*

Description

Compute the general equilibrium of a Robinson Crusoe economy.

Usage

```
gemRobinson_3_2(dstl, endowment)
```

Arguments

`dstl`	the demand structure tree list.
`endowment`	the endowment 3-vector. The endowment of the product is a non-negative number. The endowments of labor and land are positive numbers.

Details

A general equilibrium model with 3 commodities (i.e. product, labor, and land) and 2 agents (i.e. a firm and a consumer). The numeraire is labor.

Value

A general equilibrium.

References

http://essentialmicroeconomics.com/ChapterY5/SlideChapter5-1.pdf

http://homepage.ntu.edu.tw/~josephw/MicroTheory_Lecture_11a_RobinsonCrusoeEconomy.pdf

Examples

```
#### a general equilibrium model with 2 basic commodities (i.e. labor and land) and 1 agent
dst.Robinson <- node_new("util",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "prod", "lab"
)
node_set(dst.Robinson, "prod",
  type = "CD", alpha = 8, beta = c(0.5, 0.5),
  "lab", "land"
)

node_plot(dst.Robinson)

ge <- sdm2(
  A = list(dst.Robinson),
  names.commodity = c("lab", "land"),
  names.agent = c("Robinson"),
  B = matrix(0, 2, 1),
  S0Exg = matrix(c(
    12,
    1
  ), 2, 1, TRUE),
  numeraire = "lab"
)
ge
```

```
## the same economy as above
dst.Robinson <- node_new("util",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "prod", "lab"
)
dst.firm <- node_new("output",
  type = "CD", alpha = 8, beta = c(0.5, 0.5),
  "lab", "land"
)

dstl <- list(dst.firm, dst.Robinson)

ge <- gemRobinson_3_2(dstl, endowment = c(0, 12, 1))
ge

## another example
dst.firm$alpha <- 1

ge <- gemRobinson_3_2(dstl, endowment = c(3, 144, 1))
ge


#### a Robinson Crusoe economy with labor and two types of land
dst.Robinson <- node_new("util",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "prod1", "prod2"
)
node_set(dst.Robinson, "prod1",
  type = "CD", alpha = 1, beta = c(0.2, 0.8),
  "lab", "land1"
)
node_set(dst.Robinson, "prod2",
  type = "CD", alpha = 1, beta = c(0.8, 0.2),
  "lab", "land2"
)
node_plot(dst.Robinson)

dstl <- list(dst.Robinson)

ge.3_1 <- sdm2(dstl,
  names.commodity = c("lab", "land1", "land2"),
  names.agent = c("Robinson"),
  B = matrix(0, 3, 1),
  S0Exg = matrix(c(
    100,
    100,
    100
  ), 3, 1, TRUE),
  numeraire = "lab"
)
ge.3_1
```

```
#### the same economy as above
ge.5_3 <- sdm2(
  A = list(
    dst.firm1 = node_new("output",
                         type = "CD", alpha = 1, beta = c(0.2, 0.8),
                         "lab", "land1"
    ),
    dst.firm2 = node_new("output",
                         type = "CD", alpha = 1, beta = c(0.8, 0.2),
                         "lab", "land2"
    ),
    dst.Robinson = node_new("util",
                            type = "CD", alpha = 1, beta = c(0.5, 0.5),
                            "prod1", "prod2"
    )
  ),
  names.commodity = c("prod1", "prod2", "lab", "land1", "land2"),
  names.agent = c("firm1", "firm2", "Robinson"),
  B = {
    B <- matrix(0, 5, 3)
    B[1, 1] <- B[2, 2] <- 1
    B
  },
  S0Exg = {
    S0Exg <- matrix(NA, 5, 3)
    S0Exg[3:5, 3] <- 100
    S0Exg
  },
  numeraire = "lab"
)
ge.5_3
```

`gemStandardInputOutputTable_7_4` *A General Equilibrium Model based on a 7x4 Standard Input-Output Table*

Description

This is a general equilibrium model based on a 7x4 standard input-output table. There is no negative number in this standard input-output table, and both the input and output parts are 7x4 matrices. The standard input-output table consists of input and output parts with the same dimensions.

Usage

```
gemStandardInputOutputTable_7_4(
  IT,
  OT,
  es.agri = 0,
  es.manu = 0,
  es.serv = 0,
  es.hh = 0,
  es.VA.agri = 0.25,
  es.VA.manu = 0.5,
  es.VA.serv = 0.8,
  ...
)
```

Arguments

`IT`	the input part of the input-output table in the base period (unit: trillion yuan).
`OT`	the output part of the input-output table in the base period (unit: trillion yuan).
`es.agri, es.manu, es.serv`	the elasticity of substitution between the intermediate input and the value-added input of the agriculture sector, manufacturing sector and service sector.
`es.hh`	the elasticity of substitution among products consumed by the household sector.
`es.VA.agri, es.VA.manu, es.VA.serv`	the elasticity of substitution between labor input and capital input of the agriculture sector, manufacturing sector and service sector.
`...`	arguments to be transferred to the function sdm of the package CGE.

Details

Given a 7x4 input-output table, this model calculates the corresponding general equilibrium. This input-output table contains 3 production sectors and 1 household. The household consumes products and supplies labor, capital, stock and tax receipt. Generally speaking, the value of the elasticity of substitution in this model should be between 0 and 1.

Value

A general equilibrium, which is a list with the following elements:

- p - the price vector with labor as numeraire.
- D - the demand matrix, also called the input table. Wherein the benchmark prices are used.
- DV - the demand value matrix, also called the value input table. Wherein the current price is used.
- SV - the supply value matrix, also called the value output table. Wherein the current price is used.
- value.added - the value-added of the three production sectors.
- dstl - the demand structure tree list of sectors.
- ... - some elements returned by the CGE::sdm function

See Also

gemInputOutputTable_7_4

Examples

```
IT2017 <- matrix(c(
  1.47, 6.47, 0.57, 2.51,
  2.18, 76.32, 12.83, 44.20,
  0.82, 19.47, 23.33, 35.61,
  6.53, 13.92, 21.88, 0,
  0.23, 4.05, 6.76, 0,
  0, 6.43, 3.40, 0,
  0.13, 8.87, 10.46, 0
), 7, 4, TRUE)

OT2017 <- matrix(c(
  11.02, 0, 0, 0,
  0, 135.53, 0, 0,
  0, 0, 79.23, 0,
  0, 0, 0, 42.33,
  0, 0, 0, 11.04,
  0.34, 0, 0, 9.49,
  0, 0, 0, 19.46
), 7, 4, TRUE)

rownames(IT2017) <- rownames(OT2017) <-
  c("agri", "manu", "serv", "lab", "cap", "tax", "dividend")
colnames(IT2017) <- colnames(OT2017) <-
  c("sector.agri", "sector.manu", "sector.serv", "sector.hh")

ge <- gemStandardInputOutputTable_7_4(
  IT = IT2017,
  OT = OT2017
)



#### labor supply reduction
OTLSR <- OT2017
OTLSR["lab", "sector.hh"] <- OTLSR["lab", "sector.hh"] * 0.9
geLSR <- gemStandardInputOutputTable_7_4(
  IT = IT2017,
  OT = OTLSR
)

geLSR$z / ge$z
geLSR$p / ge$p

#### capital accumulation
OTCA <- OT2017
OTCA["cap", "sector.hh"] <- OTCA["cap", "sector.hh"] * 1.1
geCA <- gemStandardInputOutputTable_7_4(
  IT = IT2017,
  OT = OTCA
)
```

```
geCA$z / ge$z
geCA$p / ge$p

#### technology progress
IT.TP <- IT2017
IT.TP ["lab", "sector.manu"] <-
  IT.TP ["lab", "sector.manu"] * 0.9

geTP <- gemStandardInputOutputTable_7_4(
  IT = IT.TP,
  OT = OT2017
)

geTP$z / ge$z
geTP$p / ge$p

##
IT.TP2 <- IT.TP
IT.TP2 ["cap", "sector.manu"] <-
  IT.TP2["cap", "sector.manu"] * 1.02
geTP2 <- gemStandardInputOutputTable_7_4(
  IT = IT.TP2,
  OT = OT2017
)

geTP2$z / ge$z
geTP2$p / ge$p

##
IT.TP3 <- IT2017
IT.TP3 ["lab", "sector.manu"] <-
  IT.TP3 ["lab", "sector.manu"] * 0.9
IT.TP3 ["lab", "sector.agri"] <-
  IT.TP3 ["lab", "sector.agri"] * 0.8

geTP3 <- gemStandardInputOutputTable_7_4(
  IT = IT.TP3,
  OT = OT2017
)

geTP3$value.added / ge$value.added
prop.table(geTP3$value.added) - prop.table(ge$value.added)

#### demand structure change
IT.DSC <- IT2017
IT.DSC["serv", "sector.hh"] <- IT.DSC ["serv", "sector.hh"] * 1.2

geDSC <- gemStandardInputOutputTable_7_4(
  IT = IT.DSC,
  OT = OT2017
)
```

```
geDSC$z[1:3] / ge$z[1:3]
geDSC$p / ge$p

#### tax change
OT.TC <- OT2017
OT.TC["tax", "sector.agri"] <- OT.TC["tax", "sector.agri"] * 2

geTC <- gemStandardInputOutputTable_7_4(
  IT = IT2017,
  OT = OT.TC
)

geTC$z / ge$z
geTC$p / ge$p

##
IT.TC2 <- IT2017
IT.TC2["tax", "sector.manu"] <- IT.TC2["tax", "sector.manu"] * 0.8

geTC2 <- gemStandardInputOutputTable_7_4(
  IT = IT.TC2,
  OT = OT2017
)

geTC2$z / ge$z
geTC2$p / ge$p
```

`gemTax_5_4` *A General Equilibrium Model with Tax (see Cardenete et al., 2012).*

Description

A general equilibrium model with tax (see chapter 4, Cardenete et al., 2012), wherein there are 5 commodities (i.e. product 1, product 2, labor, capital goods, and tax receipt) and 4 agents (i.e. 2 firms and 2 consumers).

Usage

```
gemTax_5_4(
  dstl,
  names.commodity = c("prod1", "prod2", "lab", "cap", "tax"),
  names.agent = c("taxed.firm1", "taxed.firm2", "consumer1", "consumer2"),
  delta = 1,
  supply.lab.consumer1 = 30,
  supply.cap.consumer1 = 20,
  supply.lab.consumer2 = 20,
  supply.cap.consumer2 = 5,
  policy.tax = NULL
)
```

Arguments

`dstl`	the demand structure tree list.
`names.commodity`	names of commodities.
`names.agent`	names of agents.
`delta`	the proportion of tax revenue allocated to consumer 1. 1-delta is the proportion of tax revenue allocated to consumer 2.
`supply.lab.consumer1`	the labor supply of consumer 1.
`supply.cap.consumer1`	the capital supply of consumer 1.
`supply.lab.consumer2`	the labor supply of consumer 2.
`supply.cap.consumer2`	the capital supply of consumer 2.
`policy.tax`	a tax policy function (see sdm2).

Value

A general equilibrium (see sdm2), wherein labor is the numeraire.

References

Manuel Alejandro Cardenete, Ana-Isabel Guerra, Ferran Sancho (2012, ISBN: 9783642247453) Applied General Equilibrium: An Introduction. Springer-Verlag Berlin Heidelberg.

Examples

```
dst.consumer1 <- node_new("utility",
  type = "CD",
  alpha = 1,
  beta = c(0.3, 0.7),
  "prod1", "prod2"
)

dst.consumer2 <- Clone(dst.consumer1)
dst.consumer2$beta <- c(0.6, 0.4)

dst.firm1 <- node_new("output",
  type = "Leontief",
  a = c(0.5, 0.2, 0.3),
  "VA", "prod1", "prod2"
)
node_set(dst.firm1, "VA",
  type = "CD",
  alpha = 0.8^-0.8 * 0.2^-0.2,
  beta = c(0.8, 0.2),
  "lab", "cap"
```

```
)

dst.firm2 <- Clone(dst.firm1)
node_set(dst.firm2, "output",
  a = c(0.25, 0.5, 0.25)
)
node_set(dst.firm2, "VA",
  alpha = 0.4^-0.4 * 0.6^-0.6,
  beta = c(0.4, 0.6)
)

## no taxation
dstl <- list(dst.firm1, dst.firm2, dst.consumer1, dst.consumer2)
ge <- gemTax_5_4(dstl, delta = 1)

## ad valorem output tax (see Table 4.1)
output.tax.rate <- 0.1
dst.taxed.firm1 <- node_new("taxed.output",
  type = "FIN",
  rate = c(1, output.tax.rate),
  dst.firm1, "tax"
)
node_plot(dst.taxed.firm1)

dst.taxed.firm2 <- node_new("taxed.output",
  type = "FIN",
  rate = c(1, output.tax.rate),
  dst.firm2, "tax"
)
node_plot(dst.taxed.firm2)

dstl <- list(dst.taxed.firm1, dst.taxed.firm2, dst.consumer1, dst.consumer2)

ge.output.tax1 <- gemTax_5_4(dstl, delta = 1)
ge.output.tax2 <- gemTax_5_4(dstl, delta = 0.5)
ge.output.tax3 <- gemTax_5_4(dstl, delta = 0)

## labor tax (see Table 4.3)
lab.tax.rate <- 0.1

dst.taxed.lab <- node_new("taxed.lab",
  type = "FIN",
  rate = c(1, lab.tax.rate),
  "lab",
  "tax"
)

dst.labor.taxed.firm1 <- Clone(dst.firm1)
node_prune(dst.labor.taxed.firm1, "lab", "cap")
node_set(
  dst.labor.taxed.firm1, "VA",
  dst.taxed.lab,
  "cap"
```

```
)

dst.labor.taxed.firm2 <- Clone(dst.labor.taxed.firm1)
node_set(dst.labor.taxed.firm2, "output",
  a = c(0.25, 0.5, 0.25)
)
node_set(dst.labor.taxed.firm2, "VA",
  alpha = 0.4^-0.4 * 0.6^-0.6,
  beta = c(0.4, 0.6)
)

dstl.labor.tax <- list(dst.labor.taxed.firm1, dst.labor.taxed.firm2, dst.consumer1, dst.consumer2)

ge.lab.tax <- gemTax_5_4(dstl.labor.tax, delta = 0.5)

ge.lab.tax$p
ge.lab.tax$z / ge$z - 1

## income tax (see Table 4.3)
income.tax.rate <- 0.2
consumption.tax.rate <- income.tax.rate / (1 - income.tax.rate)
dst.taxed.consumer1 <- node_new("taxed.utility",
  type = "FIN",
  rate = c(1, consumption.tax.rate),
  dst.consumer1,
  "tax"
)

dst.taxed.consumer2 <- node_new("taxed.utility",
  type = "FIN",
  rate = c(1, consumption.tax.rate),
  dst.consumer2,
  "tax"
)

dstl <- list(dst.firm1, dst.firm2, dst.taxed.consumer1, dst.taxed.consumer2)

ge.income.tax <- gemTax_5_4(dstl, delta = 0.5)
ge.income.tax$z / ge$z - 1

## labor tax (see Table 4.3)
lab.tax.rate <- 0.3742
node_set(dst.labor.taxed.firm1, "taxed.lab",
  rate = c(1, lab.tax.rate)
)
node_set(dst.labor.taxed.firm2, "taxed.lab",
  rate = c(1, lab.tax.rate)
)

ge.lab.tax <- gemTax_5_4(list(
  dst.labor.taxed.firm1,
  dst.labor.taxed.firm2,
  dst.consumer1,
  dst.consumer2
), delta = 0.5)
ge.lab.tax$z / ge$z - 1

## variable labor tax rate
policy.var.tax.rate <- function(time, dstl, state) {
  current.tax.rate <- NA
  if (time >= 200) {
    tax.amount <- (state$p / state$p[3])[5]
    adjustment.ratio <- ratio_adjust(tax.amount / 18.7132504, coef = 0.1)
    last.tax.rate <- node_set(dstl[[1]], "taxed.lab")$rate[2]
    current.tax.rate <- last.tax.rate / adjustment.ratio
  } else {
    current.tax.rate <- 0.1
  }
  node_set(dstl[[1]], "taxed.lab", rate = c(1, current.tax.rate))
  node_set(dstl[[2]], "taxed.lab", rate = c(1, current.tax.rate))

  state$current.policy.data <- c(time, current.tax.rate)
  state
}

ge.var.lab.tax <- gemTax_5_4(dstl.labor.tax, policy = policy.var.tax.rate)
matplot(ge.var.lab.tax$ts.z, type = "l")
matplot(ge.var.lab.tax$ts.p / ge.var.lab.tax$p[3], type = "l")
plot(ge.var.lab.tax$policy.data[, 1], ge.var.lab.tax$policy.data[, 2],
  ylab = "labor tax rate"
)
ge.var.lab.tax$p / ge.var.lab.tax$p[3]
```

gemTwoCountryPureExchange *Some Examples of Two-Country Pure Exchange Economy*

Description

Some general equilibrium examples of two-country pure exchange economy.

Usage

```
gemTwoCountryPureExchange(...)
```

Arguments

`...`	arguments to be passed to the function sdm2.

Value

A general equilibrium.

Examples

```
es.DFProd <- 0.8 # substitution elasticity between domestic and foreign products
technology.level.CHN <- 0.9

dst.CHN <- node_new("util",
  type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = es.DFProd,
  "prod.CHN", "prod.USA"
)
node_set(dst.CHN, "prod.CHN",
  type = "Leontief", a = 1 / technology.level.CHN,
  "lab.CHN"
)
node_set(dst.CHN, "prod.USA",
  type = "Leontief", a = 1,
  "lab.USA"
)
node_plot(dst.CHN)

dst.USA <- Clone(dst.CHN)

dstl <- list(dst.CHN, dst.USA)

ge <- gemTwoCountryPureExchange(dstl,
  names.commodity = c("lab.CHN", "lab.USA"),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 2, 2, TRUE),
  S0Exg = matrix(c(
    100, 0,
    0, 100
  ), 2, 2, TRUE),
  numeraire = "lab.CHN"
)
ge$p[2]
# the same as above
technology.level.CHN^(1 / es.DFProd - 1)

## supply change
geSC <- gemTwoCountryPureExchange(dstl,
  names.commodity = c("lab.CHN", "lab.USA"),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 2, 2, TRUE),
  S0Exg = matrix(c(
    200, 0,
    0, 100
  ), 2, 2, TRUE),
  numeraire = "lab.CHN"
)
geSC$p[2]

## preference change
dst.CHN$beta <- c(0.6, 0.4)
```

```
gePC <- gemTwoCountryPureExchange(dstl,
  names.commodity = c("lab.CHN", "lab.USA"),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 2, 2, TRUE),
  S0Exg = matrix(c(
    100, 0,
    0, 100
  ), 2, 2, TRUE),
  numeraire = "lab.CHN"
)

gePC$p[2]

#### Add currencies to the example above.
interest.rate <- 1e-4
es.DFProd <- 0.8
technology.level.CHN <- 0.9

prod_money.CHN <- node_new("prod_money.CHN",
  type = "FIN", rate = c(1, interest.rate),
  "prod.CHN", "money.CHN"
)
node_set(prod_money.CHN, "prod.CHN",
  type = "Leontief", a = 1 / technology.level.CHN,
  "lab.CHN"
)

prod_money.USA <- node_new("prod_money.USA",
  type = "FIN", rate = c(1, interest.rate),
  "prod.USA", "money.USA"
)
node_set(prod_money.USA, "prod.USA",
  type = "Leontief", a = 1,
  "lab.USA"
)

dst.CHN <- node_new("util",
  type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = es.DFProd,
  prod_money.CHN, prod_money.USA
)

dst.USA <- Clone(dst.CHN)

dstl <- list(dst.CHN, dst.USA)

ge <- gemTwoCountryPureExchange(dstl,
  names.commodity = c(
    "lab.CHN", "money.CHN",
    "lab.USA", "money.USA"
  ),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 4, 2, TRUE),
  S0Exg = matrix(c(
    100, 0,
    100, 0,
    0, 100,
    0, 100
  ), 4, 2, TRUE),
  numeraire = c("money.CHN" = interest.rate)
)

ge$p["money.USA"] / ge$p["money.CHN"] # the exchange rate

#### supply change
geSC <- gemTwoCountryPureExchange(dstl,
  names.commodity = c(
    "lab.CHN", "money.CHN",
    "lab.USA", "money.USA"
  ),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 4, 2, TRUE),
  S0Exg = matrix(c(
    200, 0,
    100, 0,
    0, 100,
    0, 100
  ), 4, 2, TRUE),
  numeraire = c("money.CHN" = interest.rate)
)
geSC$p["money.USA"] / geSC$p["money.CHN"]

## preference change
dst.CHN$beta <- c(0.6, 0.4)
gePC <- gemTwoCountryPureExchange(dstl,
  names.commodity = c(
    "lab.CHN", "money.CHN",
    "lab.USA", "money.USA"
  ),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 4, 2, TRUE),
  S0Exg = matrix(c(
    100, 0,
    100, 0,
    0, 100,
    0, 100
  ), 4, 2, TRUE),
  numeraire = c("money.CHN" = interest.rate)
)

gePC$p["money.USA"] / gePC$p["money.CHN"]

#### the exchange rate under a high substitution elasticity
#### between domestic and foreign products.
interest.rate <- 1e-4
es.DFProd <- 3
technology.level.CHN <- 0.9
```

```
prod_money.CHN <- node_new("prod_money.CHN",
  type = "FIN", rate = c(1, interest.rate),
  "prod.CHN", "money.CHN"
)
node_set(prod_money.CHN, "prod.CHN",
  type = "Leontief", a = 1 / technology.level.CHN,
  "lab.CHN"
)

prod_money.USA <- node_new("prod_money.USA",
  type = "FIN", rate = c(1, interest.rate),
  "prod.USA", "money.USA"
)
node_set(prod_money.USA, "prod.USA",
  type = "Leontief", a = 1,
  "lab.USA"
)

dst.CHN <- node_new("util",
  type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = es.DFProd,
  prod_money.CHN, prod_money.USA
)

dst.USA <- Clone(dst.CHN)

dstl <- list(dst.CHN, dst.USA)

ge <- gemTwoCountryPureExchange(dstl,
  names.commodity = c(
    "lab.CHN", "money.CHN",
    "lab.USA", "money.USA"
  ),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 4, 2, TRUE),
  S0Exg = matrix(c(
    100, 0,
    100, 0,
    0, 100,
    0, 100
  ), 4, 2, TRUE),
  numeraire = c("money.CHN" = interest.rate)
)

ge$p["money.USA"] / ge$p["money.CHN"] # the exchange rate

## supply change and high substitution elasticity
geSC <- gemTwoCountryPureExchange(dstl,
  names.commodity = c(
    "lab.CHN", "money.CHN",
    "lab.USA", "money.USA"
  ),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 4, 2, TRUE),
  S0Exg = matrix(c(
    200, 0,
    100, 0,
    0, 100,
    0, 100
  ), 4, 2, TRUE),
  numeraire = c("money.CHN" = interest.rate)
)
geSC$p["money.USA"] / geSC$p["money.CHN"]
```

---

ge_tidy *Tidy a General Equilibrium*

Description

Add names to the matrices and vectors of a general equilibrium, and add demand matrix, demand value matrix and supply value matrix to it.

Usage

```
ge_tidy(ge, names.commodity, names.agent)
```

Arguments

`ge`	a general equilibrium.
`names.commodity`	a character vector consisting of names of commodities.
`names.agent`	a character vector consisting of names of agents.

Value

A tidied general equilibrium.

---

growth_rate *Compute the Growth Rate*

Description

Compute the growth rates for a vector or each column of a matrix.

Usage

```
growth_rate(x, log = FALSE, first.na = FALSE)
```

Arguments

`x`	a vector or a matrix.
`log`	If log == TRUE, the logarithmic growth rate will be computed.
`first.na`	If first.na==FALSE, the result doesn't contain the first NA.

Value

a vector or a matrix consisting of growth rates.

Examples

```
x <- matrix(1:8, 4, 2)
growth_rate(x)
```

`intertemporal_utility` *Intertemporal Utility Function*

Description

An intertemporal utility function with constant relative risk aversion.

Usage

```
intertemporal_utility(x, beta = 1, gamma = 1)
```

Arguments

`x`	a vector consists of the instantaneous utility levels of some periods.
`beta`	the discount factor.
`gamma`	the coefficient (>=0) in CRRA instantaneous utility function.

Value

A list containing the following components:

- u.intertemporal: the intertemporal utility level.
- CE: the certainty-equivalent instantaneous utility level (i.e. steady-state-equivalent instantaneous utility level).

Examples

```
intertemporal_utility(c(1, 2, 3), beta = 0.99, gamma = 0.5)
intertemporal_utility(c(1, 2, 3) / 10, beta = 0.99, gamma = 0.5)
intertemporal_utility(c(1, 2, 3) / 10, gamma = 0)
```

`iterate` *Iteration Function*

Description

Iteration function

Usage

```
iterate(x, f, times = 100, tol = NA, ...)
```

Arguments

`x`	the initial state vector. If x has a name attribute, the names will be used to label the output matrix.
`f`	a user-supplied function that computes the values of the next time.
`times`	the iteration times.
`tol`	the tolerance for stopping calculation. If the canberra distance of the last two state vectors is less than tol the calculation will stop.
`...`	optional arguments passed to the f function.

Value

A matrix consisting of state vectors.

Examples

```
x <- c(1, 2)
f <- function(x, a) prop.table(c(sum(x), a * prod(x)^(1 / 2)))
iterate(x, f, 100, a = 3)
iterate(x, f, 100, tol = 1e-5, a = 3)

x <- c(1, 2, 3)
f <- function(x) {
  n <- length(x)
  sigma <- seq(-1, 1, length.out = n)
  result <- rep(NA, n)
  for (k in 1:n) result[k] <- CES(sigma[k], 1, rep(1 / n, n), x, rep(1 / n, n))
  prop.table(result)
}
iterate(x, f, 100)
```

---

makePolicyIncomeTax *Make a Policy of Income Tax*

Description

This function returns a policy function that redistributes the supplies of economic agents, and the effect is equivalent to the collection of income tax.

Usage

```
makePolicyIncomeTax(agent, tax.rate, redistribution, time.win = c(1, Inf))
```

Arguments

`agent`	a vector specifying the indices or names of taxed agents.
`tax.rate`	a vector specifying the income tax rates for agents, which has the same length with the argument agent.
`redistribution`	a vector specifying the proportions of tax revenue received by agents, which has the same length with the argument agent.
`time.win`	the time window vector, i.e. a 2-vector specifying the start time and end time of policy implementation.

Value

A policy function, which is often used as an argument of the function sdm2.

References

Manuel Alejandro Cardenete, Ana-Isabel Guerra, Ferran Sancho (2012, ISBN: 9783642247453) Applied General Equilibrium: An Introduction. Springer-Verlag Berlin Heidelberg.

See Also

`gemTax_5_4`

Examples

```
## an exmaple of income tax (see Cardenete et al., 2012, Table 4.3)
dst.consumer1 <- node_new("utility",
                          type = "CD",
                          alpha = 1,
                          beta = c(0.3, 0.7),
                          "prod1", "prod2"
)

dst.consumer2 <- Clone(dst.consumer1)
dst.consumer2$beta <- c(0.6, 0.4)

dst.firm1 <- node_new("output",
                      type = "Leontief",
                      a = c(0.5, 0.2, 0.3),
                      "VA", "prod1", "prod2"
)
node_set(dst.firm1, "VA",
         type = "CD",
         alpha = 0.8^-0.8 * 0.2^-0.2,
         beta = c(0.8, 0.2),
         "lab", "cap"
)

dst.firm2 <- Clone(dst.firm1)
node_set(dst.firm2, "output",
         a = c(0.25, 0.5, 0.25)
)
node_set(dst.firm2, "VA",
         alpha = 0.4^-0.4 * 0.6^-0.6,
         beta = c(0.4, 0.6)
)
dstl <- list(dst.firm1, dst.firm2, dst.consumer1, dst.consumer2)
ge <- sdm2(dstl,
  names.commodity = c("prod1", "prod2", "lab", "cap"),
  names.agent = c("firm1", "firm2", "consumer1", "consumer2"),
  numeraire = "lab",
  B = {
    tmp <- matrix(0, 4, 4)
    tmp[1, 1] <- 1
    tmp[2, 2] <- 1
    tmp
  },
  S0Exg = {
    tmp <- matrix(NA, 4, 4)
    tmp[3:4, 3] <- c(30, 20)
    tmp[3:4, 4] <- c(20, 5)
    tmp
  },
  maxIteration = 1,
  policy = makePolicyIncomeTax(
    agent = c(3, 4),
    tax.rate = c(0.2, 0.2),
    redistribution = c(0.5, 0.5)
  )
)
```

makePolicyStickyPrice *Make a Policy of Sticky Price*

Description

Given a stickiness value and a time window vector, this function returns a policy function that sets the current prices equal to the weighted mean of the market-clearing prices and the current prices during this time window. When the stickiness value is 0, the prices will be set to the market-clearing prices. When the stickiness value is 1, the current prices will keep unchanged.

Usage

```
makePolicyStickyPrice(stickiness = 0.5, time.win = c(1, Inf), tolCond = 1e-06)
```

Arguments

`stickiness`	a stickiness value between 0 and 1.
`time.win`	the time window vector, i.e. a 2-vector specifying the start time and end time of policy implementation.
`tolCond`	the tolerance condition for computing the market-clearing price vector.

Value

A policy function, which is often used as an argument of the function sdm2.

Note

Three major price adjustment methods can be used in the structural dynamic model. The corresponding three kinds of prices are exploratory prices (the default case), market clearing prices, and sticky prices. The exploratory prices are computed based on the prices and sales rates of the previous period. In economic reality, the market clearing prices are unknown, so exploratory prices are more realistic.

When the stickiness value is positive and the parameter of priceAdjustmentFunction of sdm2 is set to {function(p, q) p} (that is, the current prices are the prices in the previous period), after the implementation of this policy the current prices will be the weighted mean of the market-clearing prices and the the prices in the previous period. In general, this function should be used this way.

See Also

`sdm2`

Examples

```
InitialEndowments <- {
  tmp <- matrix(0, 3, 2)
  tmp[1, 1] <- 0.01
  tmp[2, 2] <- tmp[3, 2] <- 1
  tmp
}

ge <- gemCanonicalDynamicMacroeconomic_3_2(
  priceAdjustmentFunction = function(p, q) p,
  policy.supply = makePolicySupply(InitialEndowments),
  policy.price = makePolicyStickyPrice(stickiness = 0.5),
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 50
)

par(mfrow = c(1, 2))
matplot(ge$ts.z, type = "b", pch = 20)
matplot(ge$ts.p, type = "b", pch = 20)
```

`makePolicySupply` *Make a Policy of Supply*

Description

Given a supply matrix and a time window vector, this function returns a policy function that sets the supply during this time window. By default, the time window of this function is c(1, 1), which means that this function will set the initial supply.

Usage

```
makePolicySupply(S, time.win = c(1, 1))
```

Arguments

`S`	a supply matrix.
`time.win`	the time window vector, i.e. a 2-vector specifying the start time and end time of policy implementation.

Value

A policy function, which is often used as an argument of the function sdm2.

See Also

sdm2

Examples

```
InitialEndowments <- {
  tmp <- matrix(0, 3, 2)
  tmp[1, 1] <- 0.01
  tmp[2, 2] <- tmp[3, 2] <- 1
  tmp
}

ge <- gemCanonicalDynamicMacroeconomic_3_2(
  policy.supply = makePolicySupply(InitialEndowments),
  policy.price = policyMarketClearingPrice,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 50
)

par(mfrow = c(1, 2))
matplot(ge$ts.z, type = "b", pch = 20)
matplot(ge$ts.p, type = "b", pch = 20)
```

---

makePolicyTechnologyChange

Make a Policy of Technology Change

Description

This function returns a policy function that changes the attributes alpha and a of the demand structure trees of agents specified. An attribute alpha is usually a parameter of a CES or CD function. An attribute a is usually a parameter of a Leontief function. For demand structure trees that do not contain these two attributes, this function has no effect.

Usage

```
makePolicyTechnologyChange(
  adjumentment.ratio = 1.1,
  agent = 1,
  time.win = c(20, 20)
)
```

Arguments

`adjumentment.ratio`	a scalar. The attributes alpha will be multiplied by adjumentment.ratio. The attributes a will be divided by adjumentment.ratio.
`agent`	a vector specifying the indices or names of agents.
`time.win`	the time window vector, i.e. a 2-vector specifying the start time and end time of policy implementation.

Value

A policy function, which is often used as an argument of the function sdm2.

See Also

`sdm2`

Examples

```
dst.firm <- node_new("output",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "prod", "lab"
)

dst.consumer <- node_new("utility",
  type = "Leontief", a = 1, "prod"
)

B <- matrix(c(
  1, 0,
  0, 0
), 2, 2, TRUE)
S0Exg <- matrix(c(
  NA, NA,
  NA, 100
), 2, 2, TRUE)

ge <- sdm2(
  A = list(dst.firm, dst.consumer), B = B, S0Exg = S0Exg,
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  priceAdjustmentFunction = function(p, q) p,
  policy = list(
    makePolicyTechnologyChange(agent = "firm"),
    makePolicyStickyPrice(stickiness = 0, time.win = c(1, 20)),
    makePolicyStickyPrice(stickiness = 0.9, time.win = c(20, Inf))
  ),
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 40
)
```

```
par(mfrow = c(1, 2))
matplot(ge$ts.z, type = "b", pch = 20)
matplot(ge$ts.p, type = "b", pch = 20)
```

---

matrix_add_by_name *Add Matrices by Names of Columns and Rows*

Description

Add together some matrices by names of columns and rows. Those matrices may have distinct sizes. All matrices should not have column names and row names other than those of the first matrix.

Usage

```
matrix_add_by_name(M, ...)
```

Arguments

M	a matrix with column names and row names.
...	some matrices with column names and row names which constitute subsets of those of M. If there is a vector, it will be converted to a matrix of one column and the column will be named after the vector.

Value

A matirx.

Examples

```
M <- matrix(0, 5, 5)
colnames(M) <- paste("c", 1:5, sep = "")
rownames(M) <- paste("r", 1:5, sep = "")

M2 <- matrix(1:9, 3, 3)
colnames(M2) <- c("c2", "c3", "c5")
rownames(M2) <- c("r1", "r2", "r4")

matrix_add_by_name(M, M2)

c1 <- c(r1 = 1, r3 = 2)
matrix_add_by_name(M, c1)
matrix_add_by_name(M, c1, M2)
```

matrix_to_dstl *Convert a Matrix into a Demand Structural Tree List*

Description

Convert a demand coefficient matrix into a demand structural tree list.

Usage

```
matrix_to_dstl(
  x,
  names.commodity = paste("comm", 1:nrow(x), sep = ""),
  names.agent = paste("agt", 1:ncol(x), sep = "")
)
```

Arguments

`x`	a matrix.
`names.commodity`	names of commodities. They will be the names of leaf nodes of each demand structural tree.
`names.agent`	names of agents. They will be the names of root nodes of those demand structural trees.

Value

A demand structural tree list.

Examples

```
A <- matrix(c(
  0, 0, 0, 1,
  8, 6, 1, 0,
  4, 2, 1.5, 0,
  2, 1.5, 0.5, 0
), 4, 4, TRUE)

dstl <- matrix_to_dstl(A)
node_print(dstl[[1]])
```

node_insert *Insert Nodes into a Tree*

Description

Scan the tree and insert nodes before the first non-root node having the name specified.
This function is based on the package data.tree and has side-effects. It modifies the tree given by the argument (see the package data.tree).

Usage

```
node_insert(tree, node.name, ...)
```

Arguments

`tree`	a tree (i.e. a Node object).
`node.name`	a character string specifying the name of a node. Some nodes will be inserted before it.
`...`	some Node objects or character strings. A character string will be treated as the name of a new node to be created. Those nodes will be inserted into the tree.

Value

Invisibly returns the parent node of those new nodes.

Examples

```
dst.firm <- node_new(
  "output",
  "prod1",  "prod2"
)
plot(dst.firm)

dst.VA <- node_new(
  "VA",
  "lab",  "cap"
)

node_insert(
  dst.firm, "prod1",
  dst.VA, "prod3"
)
node_set(
  dst.firm, "output",
  "prod4"
)
plot(dst.firm)
```

node_new *Create a Tree*

Description

Create a tree by the node_set function.

Usage

```
node_new(root.name, ...)
```

Arguments

root.name	a character string specifying the name of the root node.
...	attribute names and values (e.g. alpha=1). The parameter name of a value will be treated as the name of an attribute. A value without a parameter name will be treated as a child node or the name of a child node. If the class of the value is Node, it will be added as a child. If the class of the value is character, a child node will be created with the value as the name.

Value

A tree (i.e. a Node object).

Examples

```
#### create a tree
dst1 <- node_new("firm1")
print(dst1)

## create a tree with children
dst <- node_new(
  "firm",
  "lab", "cap", dst1
)
print(dst)

#### create a tree with attributes
dst <- node_new("firm",
  type = "CD", alpha = 1, beta = c(0.5, 0.5)
)
node_print(dst)

#### create a tree with attributes and children
dst <- node_new("firm",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "lab", "cap"
)
node_print(dst)
```

node_plot *Plot a Tree and Show the Type Attribute*

Description

A wrapper of the function plot.Node of the packages data.tree. If a non-leaf node has a type attribute, then the attribute will be shown. And if a type attribute is CES, it will be shown as SCES.

Usage

```
node_plot(node, ...)
```

Arguments

`node`	a tree (i.e. a Node object).
`...`	arguments to be to be passed to the function plot.Node.

See Also

`demand_coefficient`

node_print *Print a Tree and Its Fields*

Description

A wrapper of the function print.Node of the package data.tree. Print a tree and its fields except the func field.

Usage

```
node_print(node, ...)
```

Arguments

`node`	a Node object.
`...`	arguments passed to print.Node.

Examples

```
dst <- node_new("firm",
                type = "SCES",
                alpha = 2, beta = c(0.8, 0.2),
                es = 0.5,
                "wheat", "iron"
)

node_print(dst)

####
dst <- node_new("firm",
                type = "FUNC",
                func = min,
                "wheat", "iron"
)

node_print(dst)
```

`node_prune` *Prune Nodes off a Tree by Names*

Description

A wrapper of data.tree::Prunes. Prune nodes off a tree by names. This function has side-effects, it modifies the tree given by the argument (see the package data.tree).

Usage

```
node_prune(tree, ...)
```

Arguments

`tree`	a tree (i.e. a Node object).
`...`	some character strings specifies the names of nodes to be pruned.

Value

Invisibly returns the tree.

Examples

```
dst <- node_new(
  "firm",
  "lab", "cap", "land"
)
node_prune(
  dst,
  "cap", "land"
)
plot(dst)
```

`node_replace` *Replace a Node of a Tree*

Description

Scan the tree and replace the first non-root node having the name specified.
This function is based on the package data.tree and has side-effects. It modifies the tree given by the argument (see the package data.tree).

Usage

```
node_replace(tree, node.name, ...)
```

Arguments

`tree`	a tree (i.e. a Node object).
`node.name`	a character string specifying the name of the node to be pruned off.
`...`	some Node objects or character strings. A character string will be treated as the name of a new node to be created. Those nodes will be added to the tree.

Value

Invisibly returns the parent node of those new nodes.

Examples

```
dst.firm <- node_new(
  "output",
  "prod1", "prod2"
)
plot(dst.firm)

dst.VA <- node_new(
  "VA",
  "lab", "cap"
)

node_replace(
  dst.firm, "prod2",
  dst.VA, "prod3"
)
plot(dst.firm)

node_replace(
  dst.firm, "lab",
  "labor"
)
plot(dst.firm)

node_replace(
  dst.firm, "VA",
  "prod2"
)
plot(dst.firm)
```

---

node_set *Create a Tree or Set Attributes for a Node*

Description

Create a tree or set attributes for a node by the package data.tree. This function can also be used to add child nodes to a node. This function has side-effects, it modifies the tree given by the argument (see the package data.tree).

Usage

```
node_set(tree, node.name = NA, ...)
```

Arguments

`tree`	a tree (i.e. a Node object) or a character string. If it is a character string, a tree will be created and the string will be the name of the root. And in this case, if you need to use the following parameters to set the attributes of the root node, then the second parameter node.name should be set to NA.
`node.name`	a character string, the name of a node. If the first parameter is a tree, the value of this parameter should be the name of a node in the tree.
`...`	attribute names and values (e.g. alpha=1). The parameter name of a value will be treated as the name of an attribute. If a value is NULL, the corresponding attribute should exist and will be deleted.

A value without a parameter name will be treated as a child node or the name of a child node. If the class of the value is Node, it will be added as a child. If the class of the value is character, a child node will be created with the value as the name.

Value

Invisibly returns the node.

See Also

node_new

Examples

```
#### create a tree
dst1 <- node_set("firm1")
print(dst1)

## create a tree with children
dst <- node_set(
  "firm", NA,
  "lab", "cap", dst1
)
print(dst)

#### create a tree with attributes
dst <- node_set("firm", NA,
  type = "CD", alpha = 1, beta = c(0.5, 0.5)
)
print(dst, "type", "alpha", "beta")

#### create a tree with attributes and children
dst <- node_set("firm", NA,
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "lab", "cap"
)
print(dst, "type", "alpha", "beta")

#### set attributes for a node
dst.firm <- node_set("firm", NA, "VA")
node_set(dst.firm, "VA",
  type = "CD",
  alpha = 0.8^-0.8 * 0.2^-0.2,
  beta = c(0.8, 0.2),
  "lab",
  "cap"
)
print(dst.firm, "alpha", "beta")

## set attributes and add a child for a node
```

```
node_set(dst.firm, "VA",
  type = "SCES",
  alpha = 1,
  beta = c(0.1, 0.8, 0.1),
  es = 0,
  "land"
)
print(dst.firm, "type", "alpha", "beta", "es")

## find a node
x <- node_set(dst.firm, "VA")
node_print(x)
```

`policyMarketClearingPrice` *Market-Clearing-Price Policy Function*

Description

This policy is to make the market clear every period. In this case, the path of the economy is the market clearing path (alias instantaneous equilibrium path, iep). Generally, this function is passed to the function sdm2 as an argument to compute the market clearing path. And in this case, the argument A of the function sdm2 must be a demand structure tree list.

Usage

```
policyMarketClearingPrice(time, dstl, state)
```

Arguments

`time`	the current time.
`dstl`	the demand structure tree list in the model.
`state`	the current state.

Value

A list consisting of p, S and B which specify the prices, supplies and supply coefficient matrix after adjustment.

References

LI Wu (2019, ISBN: 9787521804225) General Equilibrium and Structural Dynamics: Perspectives of New Structural Economics. Beijing: Economic Science Press. (In Chinese)

See Also

CGE::iep and sdm2. The market clearing prices are the prices with a stickiness value equal to zero. Therefore, this function can actually be replaced by makePolicyStickyPrice in the calculation.

Examples

```
#### an iep of the example (see Table 2.1 and 2.2) of the canonical dynamic
#### macroeconomic general equilibrium model in Torres (2016).
ge <- gemCanonicalDynamicMacroeconomic_3_2(
  policy.price = policyMarketClearingPrice,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 50,
  z0 = c(0.5, 1)
)

par(mfrow = c(1, 2))
matplot(ge$ts.z, type = "b", pch = 20)
matplot(ge$ts.p, type = "b", pch = 20)

#### the same as above
ge <- gemCanonicalDynamicMacroeconomic_3_2(
  policy.price = makePolicyStickyPrice(stickiness = 0),
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 50,
  z0 = c(0.5, 1)
)

par(mfrow = c(1, 2))
matplot(ge$ts.z, type = "b", pch = 20)
matplot(ge$ts.p, type = "b", pch = 20)

#### TFP shock in the economy above (see Torres, 2016, section 2.8).
numberOfPeriods <- 200

discount.factor <- 0.97
depreciation.rate <- 0.06
beta1.firm <- 0.35
return.rate <- 1 / discount.factor - 1

set.seed(1)
alpha.shock <- rep(1, 100)
alpha.shock[101] <- exp(0.01)
for (t in 102:numberOfPeriods) {
  alpha.shock[t] <- exp(0.95 * log(alpha.shock[t - 1]))
}

policyTechnologyChange <- function(time, dstl) {
  dstl[[1]]$func <- function(p) {
    result <- CD_A(
      alpha.shock[time], rbind(beta1.firm, 1 - beta1.firm, 0),
      c(p[1] * (return.rate + depreciation.rate), p[2:3])
    )
    result[3] <- p[1] * result[1] * return.rate / p[3]
    result
  }
}

InitialEndowments <- {
  tmp <- matrix(0, 3, 2)
  tmp[1, 1] <- tmp[2, 2] <- tmp[3, 2] <- 1
  tmp
}

ge <- gemCanonicalDynamicMacroeconomic_3_2(
  policy.supply = makePolicySupply(InitialEndowments),
  policy.technology = policyTechnologyChange,
  policy.price = policyMarketClearingPrice,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 200
)

c <- ge$A[1, 2] * ge$ts.z[, 2] # consumption
par(mfrow = c(2, 2))
matplot(ge$ts.z, type = "l")
x <- 100:140
plot(x, ge$ts.z[x, 1] / ge$ts.z[x[1], 1], type = "b", pch = 20)
plot(x, ge$ts.z[x, 2] / ge$ts.z[x[1], 2], type = "b", pch = 20)
plot(x, c[x] / c[x[1]], type = "b", pch = 20)

#### an iep of example 7.2 (a monetary economy) in Li (2019). See CGE::Example7.2.
interest.rate <- 0.25
dst.firm <- node_new("cc", #composite commodity
                     type = "FIN",
                     rate = c(1, interest.rate),
                     "cc1", "money"
)
node_set(dst.firm, "cc1",
         type = "CD", alpha = 1, beta = c(0.5, 0.5),
         "wheat", "labor"
)

dst.laborer <- Clone(dst.firm)
dst.money.lender <- Clone(dst.firm)

dstl <- list(dst.firm, dst.laborer, dst.money.lender)

B <- matrix(0, 3, 3)
B[1, 1] <- 1

S0Exg <- matrix(NA, 3, 3)
S0Exg[2, 2] <- 100
```

```
S0Exg[3, 3] <- 100

InitialEndowments <- {
  tmp <- matrix(0, 3, 3)
  tmp[1, 1] <- 10
  tmp[2, 2] <- tmp[3, 3] <- 100
  tmp
}

ge <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("wheat", "labor", "money"),
  names.agent = c("firm", "laborer", "money.lender"),
  numeraire = c(money = interest.rate),
  numberOfPeriods = 20,
  maxIteration = 1,
  ts = TRUE,
  policy = list(
    makePolicySupply(S = InitialEndowments),
    policyMarketClearingPrice
  )
)

par(mfrow = c(1, 2))
matplot(ge$ts.z, type = "b", pch = 20)
matplot(ge$ts.p, type = "b", pch = 20)
```

`policyMeanValue` *Mean Value Policy Function*

Description

When the time index is an integer multiple of 200, this policy sets the current prices and supplies to the average of the previous 199 periods. This policy function is mainly used as an argument of the function sdm2 in order to accelerate convergence when calculating general equilibrium.

Usage

```
policyMeanValue(time, state, state.history)
```

Arguments

`time`	the current time.
`state`	the current state.
`state.history`	the state history, which is a list consisting of the time series of p, S, q, and z.

Value

A list consisting of p, S and B which specify the prices, supplies and supply coefficient matrix after adjustment.

See Also

`sdm2`, `gemDualLinearProgramming`.

`rate_to_beta` *Conversion between a Rate Vector and a Beta Vector*

Description

Conversion between an expenditure rate vector and a beta vector (i.e. an expenditure proportion vector). For an economic agent, the rate vector indicates the ratios between expenditures on financial instruments and the physical commodity. The first element of the rate vector indicates the quantity of the physical commodity needed to obtain a unit of output. Other elements indicate the ratio of expenditures on various financial instruments to that of the physical commodity, which may be equal to the interest rate, the tax rate, the dividend rate, etc. The beta vector indicates the proportions of expenditures on various commodities.

Usage

```
rate_to_beta(x)

beta_to_rate(x)
```

Arguments

`x`	a vector.

Value

A vector.

Functions

- `rate_to_beta`: Convert a rate vector to a beta vector.
- `beta_to_rate`: Convert a beta vector to a rate vector. When converting the beta vector into a rate vector, it will be assumed that the first element of these two vectors is the same.

See Also

demand_coefficient

Examples

```
rate_to_beta(c(1, 1 / 3, 1 / 4))
rate_to_beta(c(0.5, 1 / 3, 1 / 4))

x <- beta_to_rate(c(0.7, 0.1, 0.2))
rate_to_beta(x)
```

`ratio_adjust` *Ratio Adjustment*

Description

Adjust ratios to new values.

Usage

```
ratio_adjust(
  ratio,
  coef = 0.8,
  method = c("log", "left.linear", "trunc.log", "linear")
)
```

Arguments

`ratio`	a positive numeric vector.
`coef`	a positive number.
`method`	a character string specifying the adjustment method.

Details

For a positive ratio and the following methods, the return values are as follows:

- log : coef * log(ratio) + 1, if ratio >= 1; 1 / (coef * log(1 / ratio) + 1), if ratio < 1.
- left.linear : 1 / (coef * (1 / ratio - 1) + 1), if ratio >= 1; 1 + coef * (ratio - 1), if ratio < 1.
- trunc.log : max(coef * log(ratio) + 1, 0).
- linear : coef * (ratio - 1) + 1.

Value

A vector.

Examples

```
ratio_adjust(10, 0.8)
ratio_adjust(0.1, 0.8)

x <- seq(0.01, 2, 0.01)
plot(x, x, type = "l")
lines(x, ratio_adjust(x, 0.8, method = "log"), col = "red")
lines(x, ratio_adjust(x, 0.8, method = "left.linear"), col = "blue")
lines(x, ratio_adjust(x, 0.8, method = "trunc.log"), col = "green")
```

`SCES` *Standard CES Function*

Description

Standard CES function, e.g. alpha * (beta1 * (x1 / beta1)^sigma + beta2 * (x2 / beta2)^sigma)^(1 / sigma) wherein beta1 + beta2 == 1.

Usage

```
SCES(sigma = 1 - 1/es, alpha, beta, x, es = NA)
```

Arguments

`sigma`	the sigma coefficient.
`alpha`	the alpha coefficient.
`beta`	a vector consisting of the beta coefficients.
`x`	a vector consisting ofthe inputs.
`es`	the elasticity of substitution. If es is not NA, the value of sigma will be ignored.

Value

The output or utility level.

Examples

```
SCES(alpha = 1, beta = c(0.6, 0.4), x = c(0.6, 0.4), es = 0.5)
```

SCES_A	*Standard CES Demand Coefficient Matrix*

Description

This function computes the standard CES demand coefficient matrix (i.e. Theta==Beta), which is a wrapper of CES_A of CGE package.

Usage

```
SCES_A(sigma = 1 - 1/es, alpha, Beta, p, es = NA)
```

Arguments

`sigma`	a numeric m-vector or m-by-1 matrix. 1/(1-sigma) is the elasticity of substitution.
`alpha`	a nonnegative numeric m-vector or m-by-1 matrix.
`Beta`	a nonnegative numeric n-by-m matrix, where the sum of each column is equal to 1. If a vector is provided, then it will be converted into a single-column matrix.
`p`	a nonnegative numeric n-vector or n-by-1 matrix.
`es`	a numeric m-vector or m-by-1 matrix of elasticity of substitution. If es is not NA, the value of sigma will be ignored.

Value

A demand coefficient n-by-m matrix.

Examples

```
SCES_A(-1, 1, c(0.9, 0.1), c(1, 1))
SCES_A(alpha = 1, Beta = c(0.9, 0.1), p = c(1, 1), es = 0.5)
SCES_A(0, 1, c(0.9, 0.1), c(1, 1))
beta <- c(0.9, 0.1)
CD_A(prod(beta^-beta), c(0.9, 0.1), c(1, 1))

####
SCES_A(0, 1, c(0.9, 0.1, 0), c(1, 1, 1))

####
input <- matrix(c(
  200, 300, 100,
  150, 320, 530,
  250, 380, 0
), 3, 3, TRUE)
Beta <- prop.table(input, 2)
SCES_A(sigma = rep(0, 3), alpha = c(1, 1, 1), Beta = Beta, p = c(1, 1, 1))
SCES_A(sigma = rep(-Inf, 3), alpha = c(1, 1, 1), Beta = Beta, p = c(1, 1, 1))
```

sdm2 *Structural Dynamic Model (alias Structural Growth Model) Version 2*

Description

A new version of the sdm function in the package CGE. Now the parameter A can be a demand structure tree list. Hence we actually no longer need the function sdm_dstl. Some rarely used parameters in the function sdm have been deleted. This function is the core of this package.

Usage

```
sdm2(
  A,
  B,
  S0Exg = matrix(NA, nrow(B), ncol(B)),
  names.commodity = paste("comm", 1:nrow(B), sep = ""),
  names.agent = paste("agt", 1:ncol(B), sep = ""),
  p0 = matrix(1, nrow = nrow(B), ncol = 1),
  z0 = matrix(100, nrow = ncol(B), ncol = 1),
  GRExg = NA,
  pExg = NULL,
  numeraire = NULL,
  tolCond = 1e-05,
  maxIteration = 200,
  numberOfPeriods = 300,
  depreciationCoef = 0.8,
  priceAdjustmentFunction = NULL,
  priceAdjustmentVelocity = 0.15,
  trace = TRUE,
  ts = FALSE,
  policy = NULL,
  exchangeFunction = F_Z
)
```

Arguments

A	a demand structure tree list (i.e. dstl, see demand_coefficient), a demand coefficient n-by-m matrix (alias demand structure matrix) or a function A(state) which returns an n-by-m matrix (see the sdm function). When the calculation process involves policy functions, dstl is strongly recommended. Some policy functions in this package only support dstl.
B	a supply coefficient n-by-m matrix (alias supply structure matrix). If the (i,j)-th element of S0Exg is not NA, the value of the (i,j)-th element of B will be useless and ignored.
S0Exg	an initial exogenous supply n-by-m matrix. If the (i,j)-th element of S0Exg is zero, it means there is no supply, and NA means the exogenous part of the supply

	is zero and there may be an endogenous supply part. In most cases, this matrix contains NA values but no zeros.
`names.commodity`	names of commodities. If the parameter A is a demand structure tree list, the values in names.commodity should be the names of those leaf nodes.
`names.agent`	names of agents.
`p0`	an initial price n-vector.
`z0`	an m-vector consisting of the initial exchange levels (i.e. activity levels, production levels or utility levels).
`GRExg`	an exogenous growth rate of the exogenous supplies in S0Exg. If GRExg is NA and some commodities have exogenous supply, then GRExg will be set to 0.
`pExg`	an n-vector indicating the exogenous prices (if any).
`numeraire`	the name, index or price of the numeraire commodity. If it is a character string, then it is assumed to be the name of the numeraire commodity. If it is a number without a name, then it is assumed to be the index of the numeraire commodity. If it is a number with a name, e.g. c("lab" = 0.5), then the name is assumed to be the name of the numeraire commodity and the number is assumed to be the price of the numeraire commodity, even though the price of the numeraire commodity usually is 1.
`tolCond`	the tolerance condition.
`maxIteration`	the maximum iteration count. If the main purpose of running this function is to do simulation instead of calculating equilibrium, then maxIteration should be set to 1.
`numberOfPeriods`	the period number in each iteration, which should not be less than 20.
`depreciationCoef`	the depreciation coefficient (i.e. 1 minus the depreciation rate) of the unsold products.
`priceAdjustmentFunction`	the price adjustment function. The arguments are a price n-vector p and a sales rate n-vector q. The return value is a price n-vector. The default price adjustment method is p * (1 - priceAdjustmentVelocity * (1 - q)).
`priceAdjustmentVelocity`	the price adjustment velocity.
`trace`	if TRUE, information is printed during the running of sdm.
`ts`	if TRUE, the time series of the last iteration are returned.
`policy`	a policy function or a policy function list. A policy function has the following optional parameters: • time - the current time. • dstl - the demand structure tree list in the model, which can be adjusted in the policy function and it need not be returned. • state - the current state, which is a list. state$p is the current price vector with names. state$S is the current supply matrix. state$last.z is the last output and utility vector. state$B is the current supply coefficient matrix. state$names.commodity contains the names of commodities. state$names.agent contains the names of agents.

- state.history - the state history, which is a list consisting of the time series of p, S, q, and z.

The return value of the policy function other than a list will be ignored. If the return value is a list, it should have elements p, S and B which specify the prices, supplies and supply coefficient matrix after adjustment. A vector with the name current.policy.data can be put into the state list as well, which will be put into the return value of the sdm2.

`exchangeFunction`
: the exchange function.

Details

In each period of the structural dynamic model, the economy runs as follows.
Firstly, the new price vector emerges on the basis of the price vector and sales rates of the previous period, which indicates the current market prices.
Secondly, outputs and depreciated inventories of the previous period constitute the current supplies.
Thirdly, policy functions (if any) are implemented.
Fourthly, the current input coefficient matrix is computed and the supplies are exchanged under market prices. The exchange vector and sales rate vector are obtained. Unsold goods constitute the inventories, which will undergo depreciation and become a portion of the supplies of the next period. The exchange vector determines the current outputs and utility levels.

Value

A list usually containing the following components:

- tolerance - the tolerance of the results.
- p - equilibrium prices.
- z- equilibrium exchange levels (i.e. activity levels, output levels or utility levels).
- S - the equilibrium supply matrix at the initial period.
- growthRate - the endogenous equilibrium growth rate in a pure production economy.
- A - the equilibrium demand coefficient matrix.
- B - the supply coefficient matrix.
- S0Exg - the initial exogenous supply n-by-m matrix.
- D - the demand matrix.
- DV - the demand value matrix.
- SV - the supply value matrix.
- ts.p - the time series of prices in the last iteration.
- ts.z - the time series of exchange levels (i.e. activity levels, production levels or utility levels) in the last iteration.
- ts.S - the time series of supply matrix in the last iteration.
- ts.q - the time series of sales rates in the last iteration.
- policy.data - the policy data.

Note

In the package CGE, the instantaneous equilibrium path (alias market clearing path) is computed by the function iep. In this package, the instantaneous equilibrium path can be computed by the function sdm2 with the parameter policy equal to `policyMarketClearingPrice`.

The order of implementation of various policies is critical. When a policy list contains a supply policy, a technology (i.e. dstl) policy, a price policy (e.g. a market-clearing-price policy) and a B policy (i.e. a policy adjusting the argument B), both the supply policy and the technology policy should be placed before the price policy, and the B policy should be placed after the price policy. The reason is that the calculation of the current prices may require the use of supply and technology, while the calculation of B may require the use of the current prices.

References

LI Wu (2019, ISBN: 9787521804225) General Equilibrium and Structural Dynamics: Perspectives of New Structural Economics. Beijing: Economic Science Press. (In Chinese)

LI Wu (2010) A Structural Growth Model and its Applications to Sraffa's System. http://www.iioa.org/conferences/18th/pape

Examples

```
dst.firm <- node_new("output",
  type = "Leontief", a = c(0.5, 1),
  "prod", "lab"
)

dst.consumer <- node_new("utility",
  type = "Leontief", a = 1, "prod"
)

dstl <- list(dst.firm, dst.consumer)

B <- matrix(c(
  1, 0,
  0, 0
), 2, 2, TRUE)
S0Exg <- matrix(c(
  NA, NA,
  NA, 100
), 2, 2, TRUE)

## variable dst and technology progress
policy.TP <- function(time, state, dstl) {
  if (time >= 200) {
    dstl[[1]]$a <- c(0.5, 0.8)
  } else {
    dstl[[1]]$a <- c(0.5, 1)
  }
  state
}
```

```
ge.TP <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  policy = policy.TP,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 1000
)
matplot(ge.TP$ts.z, type = "l")
plot(ge.TP$ts.p[, 1] / ge.TP$ts.p[, 2], type = "l")

## variable supply coefficient matrix and technology progress
policy.TP <- function(time, state) {
  if (time >= 200) {
    state$B[1, 1] <- 2
  } else {
    state$B[1, 1] <- 1
  }
  state
}

ge.TP <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  policy = policy.TP,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 1000
)
matplot(ge.TP$ts.z, type = "l")
plot(ge.TP$ts.p[, 1] / ge.TP$ts.p[, 2], type = "l")

## variable dst and disequilibrium
policy.DE <- function(time, dstl) {
  if (time >= 200) {
    dstl[[1]]$a[2] <- dstl[[1]]$a[2] * 0.999
  } else {
    dstl[[1]]$a[2] <- 1
  }
}

ge.DE <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  policy = policy.DE,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 1000
)
```

```
matplot(ge.DE$ts.z, type = "l")
plot(ge.DE$ts.p[, 1] / ge.DE$ts.p[, 2], type = "l")


## structural equilibria and structural transition
policy.SE <- function(time, state, dstl) {
  dstl[[1]]$a[2] <- structural_function(state$last.z[1], c(105, 125), 1, 0.5)
}

ge.low.level <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  policy = policy.SE,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 1000,
  z0 = c(100, 100)
)
matplot(ge.low.level$ts.z, type = "l")

ge.high.level <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  policy = policy.SE,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 1000,
  z0 = c(150, 100)
)
matplot(ge.high.level$ts.z, type = "l")

policy.ST <- function(time, state, dstl) {
  dstl[[1]]$a[2] <- structural_function(state$last.z[1], c(105, 125), 1, 0.5)
  if (time >= 200 && time <= 210) state$S[2, 2] <- 125 # Introduce foreign labor.
  state
}

ge.ST <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  policy = policy.ST,
  ts = TRUE,
  maxIteration = 1,
  numberOfPeriods = 1000,
  z0 = c(100, 100)
)
matplot(ge.ST$ts.z, type = "l")

#### economic cycles and an interest rate policy for the firm
dst.firm <- node_new("cc", #composite commodity
  type = "FIN",
  rate = c(1, 0.25),
  "cc1", "money"
)
node_set(dst.firm, "cc1",
  type = "Leontief",
  a = c(0.5, 0.5),
  "wheat", "labor"
)

dst.laborer <- Clone(dst.firm)
dst.money.lender <- Clone(dst.firm)

dstl <- list(dst.firm, dst.laborer, dst.money.lender)

policy.interest.rate <- function(time, state, dstl, state.history) {
  upsilon <- NA
  if (time >= 600) {
   upsilon <- state.history$z[time - 1, 1] / mean(state.history$z[(time - 50):(time - 1), 1])
    dstl[[1]]$rate[2] <- max(0.25 + 0.5 * log(upsilon), 0)
  } else {
    dstl[[1]]$rate[2] <- 0.25
  }

  state$current.policy.data <- c(time, dstl[[1]]$rate[2], upsilon)
  state
}

B <- matrix(0, 3, 3)
B[1, 1] <- 1

S0Exg <- matrix(NA, 3, 3)
S0Exg[2, 2] <- 100
S0Exg[3, 3] <- 100

de <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("wheat", "labor", "money"),
  names.agent = c("firm", "laborer", "money.lender"),
  p0 = rbind(0.625, 0.375, 0.25),
  z0 = rbind(95, 100, 100),
  priceAdjustmentVelocity = 0.3,
  numberOfPeriods = 1000,
  maxIteration = 1,
  trace = FALSE,
  ts = TRUE
)
matplot(de$ts.z, type = "l")

ge.policy <- sdm2(
  A = dstl, B = B, S0Exg = S0Exg,
  names.commodity = c("wheat", "labor", "money"),
  names.agent = c("firm", "laborer", "money.lender"),
  p0 = rbind(0.625, 0.375, 0.25),
  z0 = rbind(95, 100, 100),
  priceAdjustmentVelocity = 0.3,
  numberOfPeriods = 1000,
  maxIteration = 1,
  trace = FALSE,
  ts = TRUE,
  policy = policy.interest.rate
)
matplot(ge.policy$ts.z, type = "l")

#### Example 9.3 in Li (2019): fixed-ratio price adjustment method
#### and disequilibrium (business cycles) in a pure production economy
fixedRatioPriceAdjustmentFunction <- function(p, q) {
  thresholdForPriceAdjustment <- 0.99
  priceAdjustmentVelocity <- 0.02
  result <- ifelse(q <= thresholdForPriceAdjustment,
    p * (1 - priceAdjustmentVelocity),
    p
  )
  return(prop.table(result))
}

de.Sraffa <- sdm2(
  A = matrix(c(
    56 / 115, 6,
    12 / 575, 2 / 5
  ), 2, 2, TRUE),
  B = diag(2),
  maxIteration = 1,
  numberOfPeriods = 100,
  p0 = rbind(1 / 15, 1),
  z0 = rbind(575, 20),
  priceAdjustmentFunction = fixedRatioPriceAdjustmentFunction,
  ts = TRUE
)
matplot(growth_rate(de.Sraffa$ts.z), type = "l")
```

`sdm_dstl` *Structural Dynamic Model (alias Structural Growth Model) with a Demand Structure Tree List*

Description

This is a wrapper of the function CGE::sdm. The parameter A of CGE::sdm is replaced with a demand structure tree list. This function can be replaced by the more comprehensive function `sdm2`, so it is not recommended.

Usage

```
sdm_dstl(dstl, names.commodity, names.agent, ...)
```

Arguments

`dstl`	a demand structure tree list.
`names.commodity`	names of commodities.
`names.agent`	names of agents.
`...`	arguments to be passed to the function CGE::sdm.

Value

A general equilibrium, which is a list with the following elements:

- D - the demand matrix.
- DV - the demand value matrix.
- SV - the supply value matrix.
- ... - some elements returned by the CGE::sdm function

References

LI Wu (2019, ISBN: 9787521804225) General Equilibrium and Structural Dynamics: Perspectives of New Structural Economics. Beijing: Economic Science Press. (In Chinese)

LI Wu (2010) A Structural Growth Model and its Applications to Sraffa's System. http://www.iioa.org/conferences/18th/pape

Manuel Alejandro Cardenete, Ana-Isabel Guerra, Ferran Sancho (2012, ISBN: 9783642247453) Applied General Equilibrium: An Introduction. Springer-Verlag Berlin Heidelberg.

Torres, Jose L. (2016, ISBN: 9781622730452) Introduction to Dynamic Macroeconomic General Equilibrium Models (Second Edition). Vernon Press.

See Also

`sdm2`

Examples

```
#### a pure exchange economy with two agents and two commodities
dst.CHN <- node_new("util.CHN",
                    type = "SCES", alpha = 1, beta = c(0.8, 0.2), es = 2,
                    "lab.CHN", "lab.ROW"
)
node_plot(dst.CHN)

dst.ROW <- node_new("util.ROW",
                    type = "SCES", alpha = 1, beta = c(0.05, 0.95), es = 2,
                    "lab.CHN", "lab.ROW"
)
```

```
dstl <- list(dst.CHN, dst.ROW)

ge <- sdm_dstl(dstl,
               names.commodity = c("lab.CHN", "lab.ROW"),
               names.agent = c("CHN", "ROW"),
               B = matrix(0, 2, 2, TRUE),
               S0Exg = matrix(c(
                 100, 0,
                 0, 600
               ), 2, 2, TRUE)
)

## supply change
geSC <- sdm_dstl(dstl,
                 names.commodity = c("lab.CHN", "lab.ROW"),
                 names.agent = c("CHN", "ROW"),
                 B = matrix(0, 2, 2, TRUE),
                 S0Exg = matrix(c(
                   200, 0,
                   0, 600
                 ), 2, 2, TRUE)
)

geSC$p / ge$p

## preference change
dst.CHN$beta <- c(0.9, 0.1)
gePC <- sdm_dstl(dstl,
                 names.commodity = c("lab.CHN", "lab.ROW"),
                 names.agent = c("CHN", "ROW"),
                 B = matrix(0, 2, 2, TRUE),
                 S0Exg = matrix(c(
                   100, 0,
                   0, 600
                 ), 2, 2, TRUE)
)

gePC$p / ge$p


#### a pure exchange economy with two agents and four basic commodities
prod.CHN <- node_new("prod.CHN",
                     type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = 0.75,
                     "lab.CHN", "cap.CHN"
)

node_plot(prod.CHN)

prod.ROW <- node_new("prod.ROW",
                     type = "SCES", alpha = 2, beta = c(0.4, 0.6), es = 0.75,
                     "lab.ROW", "cap.ROW"
)
```

```
dst.CHN <- node_new("CHN",
                    type = "SCES", alpha = 1, beta = c(0.8, 0.2), es = 2,
                    prod.CHN, prod.ROW
)

node_plot(dst.CHN)
node_print(dst.CHN)
p <- c("lab.CHN" = 1, "cap.CHN" = 1, "lab.ROW" = 1, "cap.ROW" = 1)
demand_coefficient(dst.CHN, p)


dst.ROW <- node_new("ROW",
                    type = "SCES", alpha = 1, beta = c(0.05, 0.95), es = 2,
                    prod.CHN, prod.ROW
)

node_plot(dst.ROW)
node_print(dst.ROW)

dstl <- list(dst.CHN, dst.ROW)

ge <- sdm_dstl(dstl,
               names.commodity = c("lab.CHN", "cap.CHN", "lab.ROW", "cap.ROW"),
               names.agent = c("CHN", "ROW"),
               B = matrix(0, 4, 2, TRUE),
               S0Exg = matrix(c(
                 100, 0,
                 100, 0,
                 0, 600,
                 0, 800
               ), 4, 2, TRUE)
)


## Add currencies to the example above.
prod_money.CHN <- node_new("prod_money.CHN",
                           type = "FIN", rate = c(1, 0.1), # 0.1 is the interest rate.
                           prod.CHN, "money.CHN"
)

prod_money.ROW <- node_new("prod_money.ROW",
                           type = "FIN", rate = c(1, 0.1),
                           prod.ROW, "money.ROW"
)

dst.CHN <- node_new("util.CHN",
                    type = "SCES", alpha = 1, beta = c(0.8, 0.2), es = 2,
                    prod_money.CHN, prod_money.ROW
)

dst.ROW <- node_new("util.ROW",
                    type = "SCES", alpha = 1, beta = c(0.05, 0.95), es = 2,
                    prod_money.CHN, prod_money.ROW
)

dstl <- list(dst.CHN, dst.ROW)

ge <- sdm_dstl(dstl,
               names.commodity = c(
                 "lab.CHN", "cap.CHN", "money.CHN",
                 "lab.ROW", "cap.ROW", "money.ROW"
               ),
               names.agent = c("CHN", "ROW"),
               B = matrix(0, 6, 2, TRUE),
               S0Exg = matrix(c(
                 100, 0,
                 100, 0,
                 100, 0,
                 0, 600,
                 0, 800,
                 0, 100
               ), 6, 2, TRUE)
)

ge$p["money.ROW"] / ge$p["money.CHN"] # the exchange rate


#### Example 7.6 in Li (2019), which illustrates foreign exchange rates.
interest.rate.CHN <- 0.1
interest.rate.ROW <- 0.1

firm.CHN <- node_new("output.CHN",
                     type = "FIN", rate = c(1, interest.rate.CHN),
                     "cc1.CHN", "money.CHN"
)
node_set(firm.CHN, "cc1.CHN",
         type = "CD", alpha = 1, beta = c(0.5, 0.5),
         "lab.CHN", "iron"
)

household.CHN <- node_new("util",
                          type = "FIN", rate = c(1, interest.rate.CHN),
                          "wheat", "money.CHN"
)

moneylender.CHN <- Clone(household.CHN)


firm.ROW <- node_new("output.ROW",
                     type = "FIN", rate = c(1, interest.rate.ROW),
                     "cc1.ROW", "money.ROW"
)
node_set(firm.ROW, "cc1.ROW",
         type = "CD", alpha = 1, beta = c(0.5, 0.5),
         "iron", "lab.ROW"
```

```
)

household.ROW <- node_new("util",
                          type = "FIN", rate = c(1, interest.rate.ROW),
                          "wheat", "money.ROW"
)

moneylender.ROW <- Clone(household.ROW)


dstl <- list(
  firm.CHN, household.CHN, moneylender.CHN,
  firm.ROW, household.ROW, moneylender.ROW
)

ge <- sdm_dstl(dstl,
               names.commodity = c(
                 "wheat", "lab.CHN", "money.CHN",
                 "iron", "lab.ROW", "money.ROW"
               ),
               names.agent = c(
                 "firm.CHN", "household.CHN", "moneylender.CHN",
                 "firm.ROW", "household.ROW", "moneylender.ROW"
               ),
               B = {
                 tmp <- matrix(0, 6, 6)
                 tmp[1, 1] <- 1
                 tmp[4, 4] <- 1
                 tmp
               },
               S0Exg = {
                 tmp <- matrix(NA, 6, 6)
                 tmp[2, 2] <- 100
                 tmp[3, 3] <- 600
                 tmp[5, 5] <- 100
                 tmp[6, 6] <- 100
                 tmp
               }
)

ge$p.money <- ge$p
ge$p.money["money.CHN"] <- ge$p["money.CHN"] / interest.rate.CHN
ge$p.money["money.ROW"] <- ge$p["money.ROW"] / interest.rate.ROW
ge$p.money <- ge$p.money / ge$p.money["money.CHN"]

ge$p.money["money.ROW"] / ge$p.money["money.CHN"] # the exchange rate


#### the example (see Table 2.1 and 2.2) of the canonical dynamic
#### macroeconomic general equilibrium model in Torres (2016).
discount.factor <- 0.97
return.rate <- 1 / discount.factor - 1
depreciation.rate <- 0.06
```

```
production.firm <- node_new("output",
                            type = "CD", alpha = 1, beta = c(0.65, 0.35),
                            "labor", "capital.goods"
)

household <- node_new("util",
                      type = "CD", alpha = 1, beta = c(0.4, 0.6),
                      "product", "labor"
)

leasing.firm <- node_new("output",
                         type = "FIN", rate = c(1, return.rate),
                         "product", "dividend"
)

dstl <- list(
  production.firm, household, leasing.firm
)

ge <- sdm_dstl(dstl,
               names.commodity = c("product", "labor", "capital.goods", "dividend"),
               names.agent = c("production.firm", "household", "leasing.firm"),
               B = matrix(c(
                 1, 0, 1 - depreciation.rate,
                 0, 1, 0,
                 0, 0, 1,
                 0, 1, 0
               ), 4, 3, TRUE),
               S0Exg = {
                 tmp <- matrix(NA, 4, 3)
                 tmp[2, 2] <- 1
                 tmp[4, 2] <- 1
                 tmp
               },
               priceAdjustmentVelocity = 0.03,
               maxIteration = 1,
               numberOfPeriods = 15000,
               ts = TRUE
)

ge$D # the demand matrix
ge$p / ge$p[1]

plot(ge$ts.z[, 1], type = "l")


#### an example of applied general equilibrium (see section 3.4, Cardenete et al., 2012).
dst.consumer1 <- node_new("util",
                          type = "CD", alpha = 1, beta = c(0.3, 0.7),
                          "prod1", "prod2"
)
```

```
dst.consumer2 <- node_new("util",
                          type = "CD", alpha = 1, beta = c(0.6, 0.4),
                          "prod1", "prod2"
)

dst.firm1 <- node_new("output",
                      type = "Leontief", a = c(0.5, 0.2, 0.3),
                      "VA", "prod1", "prod2"
)
node_set(dst.firm1, "VA",
         type = "CD",
         alpha = 0.8^-0.8 * 0.2^-0.2, beta = c(0.8, 0.2),
         "lab", "cap"
)

dst.firm2 <- Clone(dst.firm1)
dst.firm2$a <- c(0.25, 0.5, 0.25)
node_set(dst.firm2, "VA",
         alpha = 0.4^-0.4 * 0.6^-0.6, beta = c(0.4, 0.6)
)

node_print(dst.firm2)

dstl <- list(dst.firm1, dst.firm2, dst.consumer1, dst.consumer2)

ge <- sdm_dstl(dstl,
               names.commodity = c("prod1", "prod2", "lab", "cap"),
               names.agent = c("firm1", "firm2", "consumer1", "consumer2"),
               B = {
                 tmp <- matrix(0, 4, 4)
                 tmp[1, 1] <- 1
                 tmp[2, 2] <- 1
                 tmp
               },
               S0Exg = {
                 tmp <- matrix(NA, 4, 4)
                 tmp[3, 3] <- 30
                 tmp[4, 3] <- 20
                 tmp[3, 4] <- 20
                 tmp[4, 4] <- 5
                 tmp
               }
)
```

`structural_function` *Structural Function*

Description

A structured function is a function generated by connecting two functions through a transition region. This function calculates the value of a structured function.

Usage

```
structural_function(theta, transition.interval, f1, f2, ...)
```

Arguments

`theta`	the track switching parameter, which is a scalar.
`transition.interval`	a 2-vector.
`f1`	the first function (or a value).
`f2`	the second function (or a value).
`...`	parameters of f1 and f2.

Value

The value of the structural function.

Examples

```
x <- seq(1, 5, 0.1)
y <- c()
for (theta in x) y <- c(y, structural_function(theta, c(2, 3), log, sqrt, theta))
plot(x, y)
lines(x, log(x), col = "blue")
lines(x, sqrt(x), col = "red")
```

Index