Package ‘GE’

January 6, 2023

Type Package

Title General Equilibrium Modeling

Version 0.3.4

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

NeedsCompilation no

RoxygenNote 7.2.0

Repository CRAN

Date/Publication 2023-01-06 08:10:05 UTC

R topics documented:

`AMSD` *Additive-Mean-Variance Utility Function and Additive-Mean-Standard-Deviation Utility Function*

Description

Compute the utility function mean(x) - (gamma * sd.p(x))^theta / theta or weighted.mean(x, wt) - (gamma * sd.p(x, wt))^theta / theta.

Usage

```
AMSD(x, gamma = 1, wt = NULL, theta = 1)

AMV(x, gamma = 1, wt = NULL)
```

Arguments

`x`	a numeric vector.
`gamma`	the risk aversion coefficient.
`wt`	a numeric vector of weights (or probability). If wt is NULL, all elements of x are given the same weight.
`theta`	a non-negative scalar with a default value of 1.

Functions

- `AMSD`: Computes the utility function mean(x) - (gamma * sd.p(x))^theta / theta or weighted.mean(x, wt) - (gamma * sd.p(x, wt))^theta / theta. When theta == 2, it is the additive mean-variance utility function (i.e. the function AMV). When theta == 1 (the default value), it is the additive mean and standard deviation utility function.
- `AMV`: Compute the additive mean-variance utility function mean(x) - 0.5 * gamma * var.p(x) or weighted.mean(x, wt) - 0.5 * gamma * var.p(x, wt).

References

Nakamura, Yutaka (2015). Mean-Variance Utility. Journal of Economic Theory, 160: 536-556.

Examples

```
AMSD(1:2, gamma = 0.05)
AMSD(1:2, gamma = 1, theta = 2)

marginal_utility(
  c(1, 1.001),
  c(0, 1), function (x) AMSD(x, gamma = 0.5)
)
marginal_utility(
  c(1.001, 1),
  c(0, 1), function (x) AMSD(x, gamma = 0.5)
)
```

`AMSDP` *Additive-Mean-Standard-Deviation Portfolio Utility Function*

Description

Compute the utility function x %*% mp - gamma^theta * (t(x) %*% Cov %*% x)^(0.5 * theta) / theta for a portfolio x.

Usage

```
AMSDP(x, mp, Cov, gamma = 1, theta = 1)
```

Arguments

`x`	a numeric n-vector representing a portfolio.
`mp`	a numeric n-vector representing the mean payoff of each of the n securities.
`Cov`	the n-by-n covariance matrix of the payoff vectors of n securities.
`gamma`	the risk aversion coefficient.
`theta`	a non-negative scalar with a default value of 1.

References

Danthine, J. P., Donaldson, J. (2005, ISBN: 9780123693808) Intermediate Financial Theory. Elsevier Academic Press.

Nakamura, Yutaka (2015) Mean-Variance Utility. Journal of Economic Theory, 160: 536-556.

Sharpe, William F (2008, ISBN: 9780691138503) Investors and Markets: Portfolio Choices, Asset Prices, and Investment Advice. Princeton University Press.

Xu Gao (2018, ISBN: 9787300258232) Twenty-five Lectures on Financial Economics. Beijing: China Renmin University Press. (In Chinese)

See Also

AMSD

---

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))
```

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, prob = rep(1/length(x), length(x)))
```

Arguments

x	a payoff k-vector.
gamma	the Arrow-Pratt measure of absolute risk aversion.
prob	a probability k-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 = 1 - 1/es, alpha, beta, x, theta = NULL, es = NA)
```

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.
`es`	the elasticity of substitution. If es is not NA, the value of sigma will be ignored.

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, prob = rep(1/length(x), length(x)))
```

Arguments

`x`	a payoff k-vector.
`gamma`	the relative risk aversion coefficient.
`prob`	a probability k-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)
```

DCES *Displaced CES Utility Function and Displaced CES Demand Function*

Description

The displaced CES utility function and the displaced CES demand function (Fullerton, 1989).

Usage

```
DCES(es, beta, xi, x)

DCES_demand(es, beta, xi, w, p)

DCES_compensated_demand(es, beta, xi, u, p)

DCES_indirect(es, beta, xi, w, p)
```

Arguments

es	the elasticity of substitution.
beta	a n-vector consisting of the marginal expenditure share coefficients.
xi	a n-vector. Each element of xi parameterizes whether the particular good is a necessity for the household (Acemoglu, 2009, page 152). For example, xi[i] > 0 may mean that the household needs to consume at least a certain amount of good i to survive.
x	a n-vector consisting of the inputs.
w	a scalar indicating the income.
p	a n-vector indicating the prices.
u	a scalar indicating the utility level.

Functions

- `DCES`: Compute the displaced CES utility function (Fullerton, 1989), e.g. (beta1 ^ (1 / es) * (x1 - xi1) ^ (1 - 1 / es) + beta2 ^ (1 / es) * (x2 - xi2) ^ (1 - 1 / es)) ^ (es / (es - 1) wherein beta1 + beta2 == 1.

  When es==1, the DCES utility function becomes the Stone-Geary utility function.
- `DCES_demand`: The displaced CES demand function (Fullerton, 1989).
- `DCES_compensated_demand`: The displaced CES compensated demand function (Fullerton, 1989).
- `DCES_indirect`: The displaced CES indirect utility function (Fullerton, 1989).

References

Acemoglu, D. (2009, ISBN: 9780691132921) Introduction to Modern Economic Growth. Princeton University Press.

Fullerton, D. (1989) Notes on Displaced CES Functional Forms. Available at: https://works.bepress.com/don_fullerton/39/

Examples

```
es <- 0.99
beta <- prop.table(1:5)
xi <- 0
w <- 500
p <- 2:6

x <- DCES_demand (
  es = es,
  beta = beta,
  xi = xi,
  w = w,
  p = p
)

u <- DCES(es = es,
          beta = beta,
          xi = xi,
          x = x)

SCES(es = es,
     alpha = 1,
     beta = beta,
     x = x)

DCES_compensated_demand(
  es = es,
  beta = beta,
  xi = xi,
  u = u,
  p = p
)

DCES_compensated_demand(
  es = es,
  beta = beta,
  xi = seq(10, 50, 10),
  u = u,
  p = p
)

#### A 2-by-2 general equilibrium model
#### with a DCES utility function.
ge <- sdm2(
  A = function(state) {
    a.consumer <- DCES_demand(
      es = 2, beta = c(0.2, 0.8), xi = c(1000, 500),
      w = state$w[1], p = state$p
    )
    a.firm <- c(1.1, 0)
    cbind(a.consumer, a.firm)
```

```
  },
  B = diag(c(0, 1)),
  S0Exg = matrix(c(
    3500, NA,
    NA, NA
  ), 2, 2, TRUE),
  names.commodity = c("corn", "iron"),
  names.agent = c("consumer", "firm"),
  numeraire = "corn"
)

ge$p
ge$z
ge$A
ge$D

#### A 3-by-3 general equilibrium model
#### with a DCES utility function.
lab <- 1 # the amount of labor supplied by each laborer
n.laborer <- 100 # the number of laborers
ge <- sdm2(
  A = function(state) {
    a.firm.corn <- CD_A(alpha = 1, Beta = c(0, 0.5, 0.5), state$p)
    a.firm.iron <- CD_A(alpha = 5, Beta = c(0, 0.5, 0.5), state$p)
    a.laborer <- DCES_demand(
      es = 0, beta = c(0, 1, 0), xi = c(0.1, 0, 0),
      w = state$w[3] / n.laborer, p = state$p
    )

    cbind(a.firm.corn, a.firm.iron, a.laborer)
  },
  B = matrix(c(
    1, 0, 0,
    0, 1, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, NA, NA,
    NA, NA, lab * n.laborer
  ), 3, 3, TRUE),
  names.commodity = c("corn", "iron", "lab"),
  names.agent = c("firm.corn", "firm.iron", "laborer"),
  numeraire = "lab",
  priceAdjustmentVelocity = 0.1
)

ge$z
ge$A
ge$D
```

`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. 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`).
- CES. In this case, this node also has parameters alpha, beta, theta (optional) and es (or sigma = 1 - 1 / es) (see CGE::CES_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.
- StickyLinear or SL. That is the sticky linear type. In this case, this node also has an attribute named beta that contains the coefficients of the linear utility or production function. In order to avoid too drastic changes in the demand structure, the adjustment process of the demand structure has a certain stickiness when prices change.

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, TRUE)

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)
```

`demCreditPolicy` *A Disequilibrium Model with Credit*

Description

This is an example to illustrate that credit policies may lead to business cycles. When the firm's profit rate is high, the laborer lends labor or labor income to the firm; when the firm's profit rate is low, the firm repays the loan with products.

Usage

```
demCreditPolicy(...)
```

Arguments

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

Examples

```
dst.firm <- node_new("output",
  type = "CD", alpha = 1.2,
  beta = c(0.5, 0.5),
  "prod", "lab"
)

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

f <- function(policy = NULL) {
  ge <- sdm2(
    A = list(dst.firm, dst.consumer),
    B = matrix(c(
      1, 0,
      0, 1
    ), 2, 2, TRUE),
    S0Exg = matrix(c(
      NA, NA,
      NA, 100
    ), 2, 2, TRUE),
    names.commodity = c("prod", "lab"),
    names.agent = c("firm", "consumer"),
    ts = TRUE,
    policy = policy,
    numberOfPeriods = 200,
    maxIteration = 1,
    priceAdjustmentVelocity = 0.05
  )

  matplot(ge$ts.z, type = "o", pch = 20)
  ge
}

## no credit policy
f()

## credit policy
policy.credit <- function(time, state) {
  profit.rate <- state$p[1] / sum(state$last.A[, 1] * state$p) - 1

  if (profit.rate > 0.01) {
    state$S[2, 2] <- 50
    state$S[2, 1] <- 50
  } else if (profit.rate < -0.01) {
    state$S[1, 2] <- state$S[1, 1] * 0.5
    state$S[1, 1] <- state$S[1, 1] * 0.5
  } else {
    state$S[2, 2] <- 100
```

```
    state$S[2, 1] <- 0
  }

  state
}

f(policy = policy.credit)
```

---

gemAssetPricing *Compute Asset Market Equilibria for Some Simple Cases*

Description

Compute the equilibrium of an asset market by the function sdm2 and by computing marginal utility of assets. This function is similar to the function gemSecurityPricing. The difference is that this function uses portfolio utility functions, while the function gemSecurityPricing uses payoff utility functions.

Usage

```
gemAssetPricing(S, uf, numeraire = nrow(S), ratio_adjust_coef = 0.1, ...)
```

Arguments

`S`	an n-by-m supply matrix of assets.
`uf`	a portfolio utility function or a list of m portfolio utility functions.
`numeraire`	the index of the numeraire commodity.
`ratio_adjust_coef`	a scalar indicating the adjustment velocity of demand structure.
`...`	arguments to be passed to the function sdm2.

Value

A general equilibrium containing a value marginal utility matrix (VMU).

References

Danthine, J. P., Donaldson, J. (2005, ISBN: 9780123693808) Intermediate Financial Theory. Elsevier Academic Press.

Sharpe, William F. (2008, ISBN: 9780691138503) Investors and Markets: Portfolio Choices, Asset Prices, and Investment Advice. Princeton University Press.

https://web.stanford.edu/~wfsharpe/apsim/index.html

See Also

gemSecurityPricing.

Examples

```
#### an example of Danthine and Donaldson (2005, section 8.3).
ge <- gemAssetPricing(
  S = matrix(c(
    10, 5,
    1, 4,
    2, 6
  ), 3, 2, TRUE),
  uf = function(x) 0.5 * x[1] + 0.9 * (1 / 3 * log(x[2]) + 2 / 3 * log(x[3])),
  maxIteration = 1,
  numberOfPeriods = 500,
  ts = TRUE
)
matplot(ge$ts.p, type = "l")
ge$p

#### an example of Sharpe (2008, chapter 2, case 1)
secy1 <- c(1, 0, 0, 0, 0)
secy2 <- c(0, 1, 1, 1, 1)
secy3 <- c(0, 5, 3, 8, 4) - 3 * secy2
secy4 <- c(0, 3, 5, 4, 8) - 3 * secy2
# unit security payoff matrix
UP <- cbind(secy1, secy2, secy3, secy4)

prob <- c(0.15, 0.25, 0.25, 0.35)
wt <- prop.table(c(1, 0.96 * prob)) # weights

ge <- gemAssetPricing(
  S = matrix(c(
    49, 49,
    30, 30,
    10, 0,
    0, 10
  ), 4, 2, TRUE),
  uf = list(
    function(portfolio) CES(alpha = 1, beta = wt, x = UP %*% portfolio, es = 1 / 1.5),
    function(portfolio) CES(alpha = 1, beta = wt, x = UP %*% portfolio, es = 1 / 2.5)
  ),
  maxIteration = 1,
  numberOfPeriods = 1000,
  numeraire = 1,
  ts = TRUE
)
matplot(ge$ts.p, type = "l")
ge$p
ge$p[3:4] + 3 * ge$p[2]
```

```
#### a 3-by-2 example of asset pricing with two heterogeneous agents who
## have different beliefs and predict different payoff vectors.
## the predicted payoff vectors of agent 1 on the two assets.
asset1.1 <- c(1, 2, 2, 0)
asset2.1 <- c(2, 2, 0, 2)

## the predicted payoff vectors of agent 2 on the two assets.
asset1.2 <- c(1, 0, 2, 0)
asset2.2 <- c(2, 1, 0, 2)

asset3 <- c(1, 1, 1, 1)

## the unit (asset) payoff matrix of agent 1.
UP1 <- cbind(asset1.1, asset2.1, asset3)

## the unit (asset) payoff matrix of agent 2.
UP2 <- cbind(asset1.2, asset2.2, asset3)

mp1 <- colMeans(UP1)
Cov1 <- cov.wt(UP1, method = "ML")$cov

mp2 <- colMeans(UP2)
Cov2 <- cov.wt(UP2, method = "ML")$cov

ge <- gemAssetPricing(
  S = matrix(c(
    1, 5,
    2, 5,
    3, 5
  ), 3, 2, TRUE),
  uf = list(
    # the utility function of agent 1.
    function(x) AMSDP(x, mp1, Cov1, gamma = 0.2, theta = 2),
    function(x) AMSDP(x, mp2, Cov2) # the utility function of agent 2.
  ),
  maxIteration = 1,
  numberOfPeriods = 1000,
  ts = TRUE
)
matplot(ge$ts.p, type = "l")
ge$p
ge$VMU

#### another 3-by-2 example.
asset1.1 <- c(0, 0, 1, 1, 2)
asset2.1 <- c(1, 2, 1, 2, 0)
asset3.1 <- c(1, 1, 1, 1, 1)

asset1.2 <- c(0, 0, 1, 2)
asset2.2 <- c(1, 2, 2, 1)
asset3.2 <- c(1, 1, 1, 1)

## the unit asset payoff matrix of agent 1.
```

```
UP1 <- cbind(asset1.1, asset2.1, asset3.1)

## the unit asset payoff matrix of agent 2.
UP2 <- cbind(asset1.2, asset2.2, asset3.2)

mp1 <- colMeans(UP1)
Cov1 <- cov.wt(UP1, method = "ML")$cov

mp2 <- colMeans(UP2)
Cov2 <- cov.wt(UP2, method = "ML")$cov

ge <- gemAssetPricing(
  S = matrix(c(
    1, 5,
    2, 5,
    3, 5
  ), 3, 2, TRUE),
  uf = list(
    # the utility function of agent 1.
    function(x) AMSDP(x, mp1, Cov1),
    function(x) AMSDP(x, mp2, Cov2) # the utility function of agent 2.
  ),
  maxIteration = 1,
  numberOfPeriods = 3000,
  ts = TRUE
)

ge$p
ge$D

#### a 5-by-3 example.
set.seed(1)
n <- 5 # the number of asset types
m <- 3 # the number of agents
Supply <- matrix(runif(n * m, 10, 100), n, m)

# the risk aversion coefficients of agents.
gamma <- runif(m, 0.25, 1)

# predicted mean payoff, which may be gross return rates, price indices or prices.
PMP <- matrix(runif(n * m, min = 0.8, max = 1.5), n, m)

# predicted standard deviation of payoff
PSD <- matrix(runif(n * m, min = 0.01, max = 0.2), n, m)
PSD[n, ] <- 0

# Suppose the predicted payoff correlation matrices of agents are the same.
Cor <- cor(matrix(runif(2 * n^2), 2 * n, n))
Cor[, n] <- Cor[n, ] <- 0
Cor[n, n] <- 1

# the list of utility functions.
lst.uf <- list()
```

```
make.uf <- function(mp, Cov, gamma) {
  force(mp)
  force(Cov)
  force(gamma)
  function(x) {
    AMSDP(x, mp = mp, Cov = Cov, gamma = gamma, theta = 1)
  }
}

for (k in 1:m) {
  sigma <- PSD[, k]
  if (is.matrix(Cor)) {
    Cov <- dg(sigma) %*% Cor %*% dg(sigma)
  } else {
    Cov <- dg(sigma) %*% Cor[[k]] %*% dg(sigma)
  }

  lst.uf[[k]] <- make.uf(mp = PMP[, k], Cov = Cov, gamma = gamma[k])
}

ge <- gemAssetPricing(
  S = Supply, uf = lst.uf,
  maxIteration = 1,
  numberOfPeriods = 2000,
  priceAdjustmentVelocity = 0.05,
  policy = makePolicyMeanValue(150),
  ts = TRUE
)

matplot(ge$ts.p, type = "l")
ge$p
ge$D
ge$VMU

#### a 2-by-2 example with outside position.
asset1 <- c(1, 0, 0)
asset2 <- c(0, 1, 1)

# unit (asset) payoff matrix
UP <- cbind(asset1, asset2)
wt <- c(0.5, 0.25, 0.25) # weights

uf1 <- function(portfolio) prod((UP %*% portfolio + c(0, 0, 2))^wt)
uf2 <- function(portfolio) prod((UP %*% portfolio)^wt)

ge <- gemAssetPricing(
  S = matrix(c(
    1, 1,
    0, 2
  ), 2, 2, TRUE),
  uf = list(uf1, uf2),
  numeraire = 1
```

```
)

ge$p
ge$z
uf1(ge$D[,1])
uf2(ge$D[,2])
```

`gemBalancedGrowthPath` *Some Examples of Balanced Growth Paths*

Description

Some examples of market clearing paths which converge to balanced growth paths.

Usage

```
gemBalancedGrowthPath(...)
```

Arguments

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

Examples

```
#### an example with a firm and a laborer
dst.firm <- node_new(
  "prod",
  type = "CD", alpha = 5, beta = c(0.5, 0.5),
  "prod", "lab"
)

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

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

ge <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 1
```

```
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "lab",
  z0 = c(1, 1),
  ts = TRUE,
  policy = policyMarketClearingPrice,
  numberOfPeriods = 40,
  maxIteration = 1,
  GRExg = 0.03
)

matplot(ge$ts.z, type = "o", pch = 20)
matplot(growth_rate(ge$ts.z), type = "o", pch = 20)

#### an example with two firms and a laborer
dst.firm.corn <- node_new(
  "corn",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "iron", "lab"
)

dst.firm.iron <- node_new(
  "iron",
  type = "CD", alpha = 5, beta = c(0.5, 0.5),
  "iron", "lab"
)

dst.consumer <- node_new(
  "util",
  type = "Leontief", a = 1,
  "corn"
)

ge <- sdm2(
  A = list(dst.firm.corn, dst.firm.iron, dst.consumer),
  B = matrix(c(
    1, 0, 0,
    0, 1, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, NA, NA,
    NA, NA, 100
  ), 3, 3, TRUE),
  names.commodity = c("corn", "iron", "lab"),
  names.agent = c("firm.corn", "firm.iron", "consumer"),
  numeraire = "lab",
  ts = TRUE,
  policy = policyMarketClearingPrice,
  numberOfPeriods = 30,
  maxIteration = 1,
```

```
  GRExg = 0.03
)

matplot(ge$ts.z, type = "o", pch = 20)
matplot(growth_rate(ge$ts.z), type = "o", pch = 20)

#### another example with two firms and a laborer
dst.manu <- node_new("manu",
                     type = "SCES", es = 1, alpha = 1,
                     beta = c(0.6, 0.4),
                     "manu", "lab"
)

dst.serv <- node_new("serv",
                     type = "SCES", es = 1, alpha = 1,
                     beta = c(0.4, 0.6),
                     "manu", "lab"
)

dst.consumer <- node_new("util",
                         type = "SCES", es = 1, alpha = 1,
                         beta = c(0.4, 0.6),
                         "manu", "serv"
)

dstl <- list(dst.manu, dst.serv, dst.consumer)

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

ge <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0, 0,
    0, 1, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = S0Exg,
  names.commodity = c("manu", "serv", "lab"),
  names.agent = c("manu", "serv", "consumer"),
  numeraire = c("manu"),
  ts = TRUE,
  policy = list(
    function(time, state) {
      if (time >= 5) {
        state$S[3, 3] <- 100 * 1.03^(time - 4)
      }
      state
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 20,
  maxIteration = 1,
```

```
  z0 = c(160, 60, 100),
  p0 = c(1, 1, 1)
)

ge$p
ge$D
ge$S

matplot(ge$ts.z, type = "o", pch = 20)
matplot(growth_rate(ge$ts.z), type = "o", pch = 20)
```

---

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 equity shares) 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)

See Also

The market clearing path (alias temporary equilibrium path, instantaneous equilibrium path) can be computed with the function `policyMarketClearingPrice`.

Examples

```
gemCanonicalDynamicMacroeconomic_3_2()

#### a market-clearing path (alias temporary 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$ts.z, type = "o", pch = 20)
matplot(ge$ts.p, type = "o", pch = 20)

#### technology change in a market-clearing path
policyTechnologyChange <- function(time, A) {
  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]) {
      A[[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$ts.z, type = "o", pch = 20)
  matplot(ge$ts.p, type = "o", 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
  )
```

`gemCanonicalDynamicMacroeconomic_4_3`
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_4_3(
  discount.factor = 0.97,
```

```
  depreciation.rate = 0.06,
  beta.lab.firm1 = 0.65,
  beta.lab.consumer = 0.6,
  ...
)
```

Arguments

`discount.factor`
: the intertemporal discount factor.

`depreciation.rate`
: the physical depreciation rate of capital stock.

`beta.lab.firm1`
: the share parameter for labor of the Cobb-Douglas production function of the production firm.

`beta.lab.consumer`
: the share parameter for labor of the Cobb-Douglas utility function of the consumer.

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

Details

A general equilibrium model with 4 commodities (i.e. product, labor, capital and equity shares) and 3 agents (i.e. a production firm, a consumer and a capital-leasing firm).

Value

A general equilibrium (see `sdm2`)

References

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

Examples

```
#### a market-clearing path that converges to the steady-state equilibrium
ge <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = 100,
  policy = policyMarketClearingPrice,
  z0 = c(1, 1, 1)
)

matplot(ge$ts.z, type = "o", pch = 20)
matplot(ge$ts.p, type = "o", pch = 20)

## population growth: a market-clearing path
## that converges a balanced growth path
ge <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = 100,
```

```
  GRExg = 0.01,
  z0 = c(1, 1, 1),
  policy = policyMarketClearingPrice
)

matplot((ge$ts.p), type = "l")
matplot((ge$ts.z), type = "l")
matplot(growth_rate(ge$ts.z), type = "l")

#### a disequilibrium path and the steady-state equilibrium
ge <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = 5000,
  priceAdjustmentVelocity = 0.03,
  z0 = c(1, 1, 1)
)

ge$p
ge$z
matplot(ge$ts.z, type = "l")
node_plot(ge$dstl[[3]], param = TRUE)

## a small disturbance to the product supply
ge <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = 4000,
  priceAdjustmentVelocity = 0.03,
  z0 = c(1, 1, 1),
  policy = function(time, state) {
    if (time == 1500) {
      state$S[1, 1] <- state$S[1, 1] * 0.999
    }
    state
  }
)

#### business cycle
de <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = 1000,
  priceAdjustmentVelocity = 0.15
)

## A tax rate policy is implemented from the 600th period to stabilize the economy.
ge <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = 1500,
  priceAdjustmentVelocity = 0.15,
  policy = Example9.10.policy.tax
)

matplot(ge$ts.z, type = "l")
plot(ge$policy.data, type = "l") # tax rates

#### a market-clearing path with a productivity shock
nPeriod <- 100 # the number of periods of the market-clearing path
set.seed(1)
```

```
alpha.shock <- rep(1, nPeriod)
alpha.shock[11] <- exp(0.01)
for (t in 12:nPeriod) {
  alpha.shock[t] <- exp(0.95 * log(alpha.shock[t - 1]))
}
plot(alpha.shock)

ge <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = nPeriod,
  p0 = c(1, 1.34312, 0.09093, 0.08865),
  z0 = c(0.7447, 0.6120, 2.8665),
  policy = list(
    function(time, A) {
      A[[1]]$alpha <- alpha.shock[time]
    },
    policyMarketClearingPrice
  )
)

matplot(ge$ts.z[, 1], type = "o", pch = 20)
```

`gemCapitalAccumulation` *Some Examples of Market-Clearing Paths with Capital Accumulation*

Description

Some examples of market-clearing paths (MCP) with capital accumulation. The economy contains a production firm, a capital-leasing firm and a consumer.

Usage

```
gemCapitalAccumulation(...)
```

Arguments

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

See Also

`gemPersistentTechnologicalProgress`

Examples

```
#### a 3-by-3 example
dst.firm1 <- node_new(
  "prod",
  type = "CD",
```

```
  alpha=2, beta = c(0.5, 0.5),
  "cap", "cc1"
)
node_set(dst.firm1, "cc1",
         type="Leontief", a=1,
         "lab")

dst.consumer <- dst.firm2 <- node_new(
  "util",
  type = "Leontief",
  a= 1,
  "prod"
)

dstl <- list(dst.firm1, dst.consumer, dst.firm2)

B <-  matrix(c(
  1, 0, 0.5,
  0, 0, 1,
  0, 0, 0
), 3, 3, TRUE)

S0Exg <-  matrix(c(
  NA, NA, NA,
  NA, NA, NA,
  NA, 100,NA
), 3, 3, TRUE)

ge <- sdm2(
  A = dstl,
  B = B,
  S0Exg = S0Exg,
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm1", "laborer","firm2"),
  numeraire = "prod",
  z0 = c(100, 0, 50),
  policy = policyMarketClearingPrice,
  maxIteration = 1,
  numberOfPeriods = 30,
  ts = TRUE
)

matplot((ge$ts.z), type="o",pch=20)
matplot((ge$ts.p), type="o",pch=20)

## a MCP with labor supply change
ge <- sdm2(
  A = dstl,
  B = B,
  S0Exg = S0Exg,
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm1", "laborer","firm2"),
  numeraire = "prod",
```

```
  z0 = c(400,200,400),
  policy = list(
    function(time, state) {
      if (time >= 5) state$S[3, 2] <- 150
      state
    },
    policyMarketClearingPrice
  ),
  maxIteration = 1,
  numberOfPeriods = 30,
  ts = TRUE
)

matplot((ge$ts.z), type="o",pch=20)
matplot((ge$ts.p), type="o",pch=20)

## a MCP with transitory technological progress
ge <- sdm2(
  A = dstl,
  B = B,
  S0Exg = S0Exg,
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm1", "laborer","firm2"),
  numeraire = "prod",
  z0 = c(400,200,400),
  policy = list(
    function(time, A) {
      if (time == 5) {
        A[[1]]$alpha = 3
      } else {
        A[[1]]$alpha = 2
      }
    },
    policyMarketClearingPrice
  ),
  maxIteration = 1,
  numberOfPeriods = 30,
  ts = TRUE
)

matplot((ge$ts.z), type="o",pch=20)
matplot((ge$ts.p), type="o",pch=20)

## a MCP with permanent technological progress
ge <- sdm2(
  A = dstl,
  B = B,
  S0Exg = S0Exg,
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm1", "laborer","firm2"),
  numeraire = "prod",
  z0 = c(400,200,400),
  policy = list(
```

```
    function(time, A) {
      if (time >= 5) {
        A[[1]]$alpha = 3
      } else {
        A[[1]]$alpha = 2
      }
    },
    policyMarketClearingPrice
  ),
  maxIteration = 1,
  numberOfPeriods = 30,
  ts = TRUE
)

matplot((ge$ts.z), type="o",pch=20)
matplot((ge$ts.p), type="o",pch=20)

#### A 4-by-4 example wherein the capital goods
#### have a useful life of two periods.
ge <- sdm2(
  A = function(state) {
    a.firm1 <- CD_A(alpha = 2, Beta = c(0, 0.5, 0.5, 0), state$p)
    a.consumer <- c(1, 0, 0, 0)
    a.firm2 <- c(1, 0, 0, 0)
    a.firm3 <- c(0, 0, 0, 1)
    cbind(a.firm1, a.consumer, a.firm2, a.firm3)
  },
  B = matrix(c(
    1, 0, 0, 0,
    0, 0, 1, 1,
    0, 0, 0, 0,
    0, 0, 1, 0
  ), 4, 4, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,NA,
    NA, NA, NA,NA,
    NA, 100,NA,NA,
    NA, NA, NA,NA
  ), 4, 4, TRUE),
  names.commodity = c("prod", "cap", "lab","prod.used"),
  names.agent = c("firm1", "consumer","firm2","firm3"),
  numeraire = "prod",
  policy = policyMarketClearingPrice,
  z0 = c(100, 100, 100, 100),
  ts = TRUE,
  numberOfPeriods = 30,
  maxIteration = 1
)

matplot(ge$ts.z, type = "o", pch = 20)
```

`gemCoffeeProblem_3_3` *Coffee Problem: Some Examples of Equilibrium and Disequilibrium Pure Exchange Economies*

Description

Some examples of equilibrium and disequilibrium pure exchange economies.

Usage

```
gemCoffeeProblem_3_3(...)
```

Arguments

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

References

Bapat, R. B., Raghavan, T. E. S. (1997, ISBN: 9780521571678) Nonnegative Matrices and Applications. Cambridge University Press.

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

Examples

```
#### equilibrium coffee problem (Bapat, Raghavan, 1997, example 7.1; Li, 2019, example 8.1)
ge <- sdm2(
  A = matrix(c(
    0.05, 0.05, 0.1,
    0.1, 0, 0.1,
    0, 0.15, 0.05
  ), 3, 3, TRUE),
  B = matrix(0, 3, 3),
  S0Exg = diag(3),
  names.commodity = c("coffee powder", "milk", "sugar"),
  names.agent = c("consumer1", "consumer2", "consumer3"),
  numeraire = "sugar"
)

ge$p

#### disequilibrium coffee problem with exogenous prices (Li, 2019, example 8.3).
de <- sdm2(
  A = matrix(c(
    0.05, 0.05, 0.1,
    0.1, 0, 0.1,
    0, 0.15, 0.05
  ), 3, 3, TRUE),
```

```
  B = matrix(0, 3, 3),
  S0Exg = diag(3),
  names.commodity = c("coffee powder", "milk", "sugar"),
  names.agent = c("consumer1", "consumer2", "consumer3"),
  pExg = c(1, 1, 1),
  maxIteration = 1,
  numberOfPeriods = 50,
  ts = TRUE
)

de$z
addmargins(de$D)
addmargins(de$S)

matplot(de$ts.z, type = "o", pch = 20)
matplot(de$ts.q, type = "o", pch = 20)
```

---

gemConstantGrowthPath_TechnologyProgress_3_3 *Constant Growth Paths with Technology Progress*

Description

This is an example of a market-clearing path converging to a constant growth path. In a constant growth path, the supply of each commodity grows at a constant rate. The balanced growth path is a special case of the constant growth path.

Usage

```
gemConstantGrowthPath_TechnologyProgress_3_3(...)
```

Arguments

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

Examples

```
dst.firm1 <- node_new(
  "output",
  type = "CD", alpha = 1, beta = c(0.35, 0.65),
  "prod1", "lab"
)

dst.firm2 <- node_new(
  "output",
  type = "CD", alpha = 1, beta = c(0.4, 0.6),
  "prod1", "lab"
)
```

```
dst.consumer <- node_new(
  "util",
  type = "Leontief", a = 1,
  "prod2"
)

ge <- sdm2(
  A = list(dst.firm1, dst.firm2, dst.consumer),
  B = matrix(c(
    1, 0, 0,
    0, 1, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, NA, NA,
    NA, NA, 1
  ), 3, 3, TRUE),
  names.commodity = c("prod1", "prod2", "lab"),
  names.agent = c("firm1", "firm2", "consumer"),
  numeraire = "lab",
  z0 = c(0.2, 0.2, 1),
  ts = TRUE,
  policy = list(
    function(time, A, state) {
      A[[1]]$alpha <- exp(time * 0.01)
      A[[2]]$alpha <- exp(time * 0.01)
      state$S[3, 3] <- exp(time * 0.01)
      state
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 20,
  maxIteration = 1
)

matplot(ge$ts.z, type = "l")
matplot(log(ge$ts.z[, 1:2]), type = "l")
matplot(growth_rate(ge$ts.z[, 1:2], log = TRUE), type = "o", pch = 20)
matplot(growth_rate(ge$ts.p[, 1:2], log = TRUE), type = "o", pch = 20)
```

---

gemDCES_5_3 *A Model with a Displaced CES Utility Function*

Description

A model with a displaced CES utility function (Zhang, 2008, page 134; Li, 2019, example 3.12, page 130).

Usage

```
gemDCES_5_3(...)
```

Arguments

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

References

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

Zhang Jinshui (2008, ISBN: 9787040224818) Mathematical Economics. Beijing: Higher Education Press. (In Chinese)

Examples

```
ge <-  sdm2(
  A = function(state) {
    a.firm1 <- CD_A(alpha = 1, Beta = c(0, 0, 0.5, 0.5, 0), state$p)
    a.firm2 <- CD_A(alpha = 2, Beta = c(0, 0, 0.5, 0, 0.5), state$p)
    a.consumer <- DCES_demand (
      es = 1,
      beta = c(1 / 3, 1 / 3, 1 / 3, 0, 0),
      xi = c(0, 0, 0.4, 0, 0),
      w = state$w[3] / 10 ^ 4,
      p = state$p
    )
    cbind(a.firm1, a.firm2, a.consumer)
  },
  B = matrix(c(1, 0, 0,
               0, 1, 0,
               0, 0, 0,
               0, 0, 0,
               0, 0, 0), 5, 3, TRUE),
  S0Exg = {
    tmp <- matrix(NA, 5, 3)
    tmp[3, 3] <- 10000
    tmp[4:5, 3] <- 1
    tmp
  },
  names.commodity = c("prod1", "prod2", "lab", "land1", "land2"),
  names.agent = c("firm1", "firm2", "consumer"),
  numeraire = "lab"
)

ge$p
```

`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, lumber and two types of skilled labor) 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.
ge <- gemDualLinearProgramming(A = A, B = B, S0Exg = S0Exg)

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

## 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,
  names.commodity = c("dollar", "lumber", "lab1", "lab2"),
  names.agent = c("desk producer", "table producer", "chair producer", "consumer"),
  numeraire = "dollar"
)

ge$z
ge$p
```

```
## Compute the general equilibrium above and
## the market-clearing path by the function sdm2.
## Warning: time consuming.
ge2 <- sdm2(
  A = A, B = B, S0Exg = S0Exg,
  policy = makePolicyStickyPrice(stickiness = 0, tolCond = 1e-4),
  maxIteration = 1,
  numberOfPeriods = 60,
  names.commodity = c("dollar", "lumber", "lab1", "lab2"),
  names.agent = c("desk producer", "table producer", "chair producer", "consumer"),
  numeraire = "dollar",
  ts = TRUE
)

matplot(ge2$ts.p, type = "l")
ge2$z
ge2$p

#### 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

#### 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

#### 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

#### 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
```

gemEquityShare_3_3 *A General Equilibrium Model with Equity Shares*

Description

A general equilibrium model with equity shares and dividend.

Usage

```
gemEquityShare_3_3(...)
```

Arguments

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

Examples

```
dst.firm <- node_new("output",
                     type = "FIN",
                     rate = c(1, dividend.rate = 0.25),
                     "cc1", "equity.share"
)
node_set(dst.firm, "cc1",
         type = "CD",
         alpha = 2, beta = c(0.5, 0.5),
         "prod", "lab"
)

dst.laborer <- node_new("util",
                        type = "Leontief",  a = 1,
                        "prod"
)

dst.shareholder <- Clone(dst.laborer)

ge <- sdm2(
  A = list(dst.firm, dst.laborer, dst.shareholder),
  B = diag(c(1, 0, 0)),
  S0Exg = {
    S0Exg <- matrix(NA, 3, 3)
    S0Exg[2, 2] <- S0Exg[3, 3] <- 100
    S0Exg
  },
  names.commodity = c("prod", "lab", "equity.share"),
  names.agent = c("firm", "laborer", "shareholder"),
  numeraire = "prod"
)
```

```
ge$p #The third component is the dividend per unit of share.
ge$DV
ge$SV
```

gemEquityShare_Bond_4_4
A General Equilibrium Model with Equity Shares and Bond

Description

A general equilibrium model with equity shares and bond.

Usage

```
gemEquityShare_Bond_4_4(...)
```

Arguments

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

Examples

```
dst.firm <- node_new("output",
                     type = "FIN",
                     rate = c(1, dividend.rate = 0.15, bond.interest.rate = 0.1),
                     "cc1", "equity.share", "bond"
)
node_set(dst.firm, "cc1",
         type = "CD",
         alpha = 2, beta = c(0.5, 0.5),
         "prod", "lab"
)

dst.laborer <- dst.shareholder <- dst.bondholder <-
  node_new("util",
           type = "Leontief",  a = 1,
           "prod"
  )

ge <- sdm2(
  A = list(dst.firm, dst.laborer, dst.shareholder, dst.bondholder),
  B = diag(c(1, 0, 0, 0)),
  S0Exg = {
    S0Exg <- matrix(NA, 4, 4)
    S0Exg[2, 2] <- S0Exg[3, 3] <-
      S0Exg[4, 4] <- 100
    S0Exg
```

```
  },
  names.commodity = c("prod", "lab", "equity.share", "bond"),
  names.agent = c("firm", "laborer", "shareholder", "bondholder"),
  numeraire = "prod"
)

ge$p
ge$DV
ge$SV
```

---

gemExogenousPrice *Some Examples with Exogenous Price (Price Regulation)*

Description

Some examples with exogenous price (price regulation). When a price regulation policy is imposed in a structural dynamic model, the economy may converge to a fixed point (i.e. a price regulation equilibrium) where the market does not clear.

Usage

```
gemExogenousPrice(...)
```

Arguments

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

Examples

```
dst.firm <- node_new("output",
  type = "CD", alpha = 5,
  beta = c(0.5, 0.5),
  "prod", "lab"
)

dst.consumer <- node_new("utility",
  type = "CD", alpha = 1,
  beta = c(0.5, 0.5),
  "prod", "lab"
)

f <- function(pExg = NULL, policy = NULL) {
  ge <- sdm2(
    A = list(dst.firm, dst.consumer),
    B = diag(c(1, 0)),
    S0Exg = {
      S0Exg <- matrix(NA, 2, 2)
      S0Exg[2, 2] <- 100
```

```
      S0Exg
    },
    names.commodity = c("prod", "lab"),
    names.agent = c("firm", "consumer"),
    numeraire = "lab",
    maxIteration = 1,
    numberOfPeriods = 100,
    p0 = c(0.16, 1),
    ts = TRUE,
    pExg = pExg,
    policy = policy
  )

  print(ge$p)
  print(ge$z)
  par(mfrow = c(1, 2))
  matplot(ge$ts.p, type = "l")
  matplot(ge$ts.z, type = "l")
  ge
}

## No price regulation policy.
f()

## Set the market prices to the steady equilibrium prices from the beginning.
## The labor market keeps oversupplied.
result <- f(pExg = c(0.16, 1))
matplot(result$ts.q, type = "l") # sale rates

## the same as above
f(policy = function(state) {
  state$p <- c(0.16, 1)
  state
})

## The price regulation policy is implemented from the 10th period.
f(policy = function(time, state) {
  if (time >= 10) state$p <- c(0.16, 1)
  state
})

## The price regulation policy is implemented from the 30th period.
f(policy = function(time, state) {
  if (time >= 30) state$p <- c(0.16, 1)
  state
})

## price ceil
f(policy = function(time, state) {
  if (time >= 30) {
    state$p <- state$p / state$p[2]
    if (state$p[1] > 0.15) state$p[1] <- 0.15
  }
```

```
  state
})

##
ge <- f(policy = function(time, state) {
  if (time >= 30) state$p <- c(0.17, 1)
  state
})

tail(ge$ts.q)
```

`gemExogenousPrice_EndogenousLaborSupply_3_3`
An Example of Price Regulation and Endogenous Labor Supply (Example 9.5 of Li, 2019)

Description

This is an example of price regulation and endogenous labor supply. See CGE::Example9.5.

Usage

```
gemExogenousPrice_EndogenousLaborSupply_3_3(...)
```

Arguments

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

Examples

```
## the exogenous labor price with product as numeraire.
p.labor <- 0.625

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

dst.land.owner <- node_new(
  "util",
  type = "Leontief", a = 1,
  "prod"
)

dst.laborer <- Clone(dst.land.owner)

ge <- sdm2(
```

```
    A = list(
      dst.firm,
      dst.land.owner,
      dst.laborer
    ),
    B = diag(3),
    S0Exg = matrix(c(
      NA, NA, NA,
      NA, 100, NA,
      NA, NA, 100
    ), 3, 3, TRUE),
    GRExg = 0,
    names.commodity = c("prod", "land", "lab"),
    names.agent = c("firm", "land.owner", "laborer"),
    maxIteration = 1,
    numberOfPeriods = 200,
    depreciationCoef = 0,
    numeraire = "prod",
    ts = TRUE,
    policy = function(time, state, state.history) {
      if (time > 1) {
        ratio <- state$p[3] / state$p[1] / p.labor
        last.labor.supply <- state.history$S[3, 3, time - 1]
        state$S[3, 3] <- last.labor.supply * ratio
      }

      state
    }
  )

  matplot(ge$ts.p, type = "l")
  tail(ge$ts.S[3, 3, ])
  plot(ge$ts.S[3, 3, ], type = "l")
```

---

gemExogenousUtilityLevel_EndogenousLaborSupply_3_3
Some Examples with Exogenous Utility Level and Endogenous Labor Supply

---

Description

Some examples with exogenous utility level and endogenous labor supply.

Usage

```
gemExogenousUtilityLevel_EndogenousLaborSupply_3_3(...)
```

Arguments

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

Examples

```
## a market-clearing path
utility.level.laborer <- 0.625

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

dst.land.owner <- node_new(
  "util",
  type = "Leontief", a = 1,
  "prod"
)

dst.laborer <- Clone(dst.land.owner)

dstl <- list(dst.firm, dst.land.owner, dst.laborer)

ge <- sdm2(
  A = dstl,
  B = diag(c(1, 0, 0)),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 100, NA,
    NA, NA, 100
  ), 3, 3, TRUE),
  GRExg = 0,
  names.commodity = c("prod", "land", "lab"),
  names.agent = c("firm", "land.owner", "laborer"),
  maxIteration = 1,
  numberOfPeriods = 30,
  numeraire = "prod",
  ts = TRUE,
  policy = list(
    function(time, state, state.history) {
      if (time > 1) {
        last.labor.supply <- state.history$S[3, 3, time - 1]

        ratio <- state$last.z[3] / last.labor.supply / utility.level.laborer
        state$S[3, 3] <- last.labor.supply * ratio
      }
      state
    },
    policyMarketClearingPrice
  )
)

matplot(ge$ts.p, type = "l")
```

```
plot(ge$ts.S[3, 3, ], type = "l")
ge$S


## Regard the laborer as a firm.
dstl[[3]] <- node_new(
  "lab",
  type = "Leontief", a = utility.level.laborer,
  "prod"
)

ge <- sdm2(
  A = dstl,
  B = diag(c(1, 0, 1)),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 100, NA,
    NA, NA, NA
  ), 3, 3, TRUE),
  names.commodity = c("prod", "land", "lab"),
  names.agent = c("firm", "land.owner", "laborer"),
  maxIteration = 1,
  numberOfPeriods = 30,
  numeraire = "prod",
  ts = TRUE,
  policy = policyMarketClearingPrice
)

matplot(ge$ts.p, type = "l")
plot(ge$ts.S[3, 3, ], type = "l")
ge$p
ge$S
```

---

gemFirmAsConsumer *Some Examples of Treating Firms as Consumer-Type Agents*

Description

Some examples of equilibrium models wherein firms are treated as consumer-type agents instead of producer-type agents.

Usage

```
gemFirmAsConsumer(...)
```

Arguments

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

See Also

gemIntertemporal_2_2

Examples

```
#### an intertemporal model with firm
## (see gemIntertemporal_2_2)
np <- 3 # the number of periods.

S0Exg <- matrix(c(0, 0, 150,
                  1000, 0, 0,
                  0, 1000, 0,
                  0, 0, 100,
                  0, 0, 100), 5, 3, TRUE)


B <- matrix(0, 5, 3, TRUE)

dst.firm1 <- node_new("util",
                      type = "StickyLinear",
                      beta=c(1, 1),
                      "prod2", "cc1")
node_set(dst.firm1, "cc1",
         type = "CD",
         alpha = 2, beta=c(0.5, 0.5),
         "prod1", "lab1")


dst.firm2 <- node_new("util",
                      type = "StickyLinear",
                      beta=c(1, 1),
                      "prod3", "cc1")
node_set(dst.firm2, "cc1",
         type = "CD",
         alpha = 2, beta=c(0.5, 0.5),
         "prod2", "lab2")

dst.consumer <- node_new(
  "util",
  type = "CD",
  alpha = 1,  beta = prop.table(rep(1, np)),
  paste0("prod", 1:np)
)

ge <-  sdm2(
  A = list(dst.firm1, dst.firm2, dst.consumer),
  B = B,
  S0Exg = S0Exg,
  names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:(np - 1))),
  names.agent = c(paste0("firm", 1:(np - 1)), "consumer"),
  numeraire = "prod1",
```

```
  ts = TRUE
)

#### an intertemporal model with bank
R <- 1.1
beta.bank <- c(1, 1/R, 1/R^2)

dst.bank <- node_new(
  "output",
  type = "StickyLinear",
  beta = beta.bank,
  "payoff1", "payoff2", "payoff3")

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1, beta = c(1 / 2, 1 / 6, 1 / 3),
  "payoff1", "payoff2", "payoff3"
)

ge <- sdm2(
  A = list(dst.bank, dst.consumer),
  B = matrix(0, 3, 2, TRUE),
  S0Exg = matrix(c(
    100, 0,
    100, 2,
    100, 1
  ), 3, 2, TRUE),
  names.commodity = c("payoff1", "payoff2", "payoff3"),
  names.agent = c("bank", "consumer"),
  numeraire = "payoff1"
)

#### an instantaneous sequential model
dst.firm <- node_new("output",
                     type = "StickyLinear",
                     beta=c(1, 1),
                     "prod", "cc1")
node_set(dst.firm, "cc1",
         type = "CD",
         alpha = 2, beta=c(0.5, 0.5),
         "cap", "lab")

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

ge <- sdm2(
  A = list(dst.firm, dst.consumer),
  B = matrix(0, 3, 2, TRUE),
  S0Exg = matrix(c(
    1000, 0,
    0, 50,
```

```
    0, 100
  ), 3, 2, TRUE),
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "prod"
)
ge$p
ge$z
ge$D

## the corresponding model treating firm as producer-type agent
ge <- sdm2(
  A = function(state) {
    a1<- CD_A(alpha = 2, Beta = c(0, 0.5, 0.5), p = state$p)
    a2 <- c(1, 0, 0)
    cbind(a1, a2)
  },
  B = diag(c(1,0), 3, 2),
  S0Exg = matrix(c(
    NA, NA,
    NA, 50,
    NA, 100
  ), 3, 2, TRUE),
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "prod"
)
ge$p
ge$z
ge$D
```

---

gemHeterogeneousFirms_2_3

Market Clearing Paths with Heterogeneous Firms

Description

This is an example of market-clearing paths with heterogeneous firms.

Usage

```
gemHeterogeneousFirms_2_3(...)
```

Arguments

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

Examples

```
dst.firm1 <- node_new(
  "output",
  type = "CD", alpha = 1, beta = c(0.35, 0.65),
  "prod", "lab"
)

dst.firm2 <- node_new(
  "output",
  type = "CD", alpha = 1.3, beta = c(0.9, 0.1),
  "prod", "lab"
)

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

ge <- sdm2(
  A = list(dst.firm1, dst.firm2, dst.consumer),
  B = matrix(c(
    1, 1, 0,
    0, 0, 0
  ), 2, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, NA, 100
  ), 2, 3, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm1", "firm2", "consumer"),
  numeraire = "lab",
  z0 = c(1, 1, 1),
  ts = TRUE,
  policy = policyMarketClearingPrice,
  numberOfPeriods = 200,
  maxIteration = 1
)

matplot(ge$ts.z, type = "l")
```

`gemInputOutputTable_2_2` *A General Equilibrium Model based on a 2x2 (Unbalanced) Input-Output Table*

Description

A general equilibrium model based on a 2x2 (unbalanced) input-output table (unit: yuan).

Usage

```
gemInputOutputTable_2_2(...)
```

Arguments

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

Examples

```
names.commodity <- c("prod", "lab")
names.agent <- c("firm", "laborer")

IT <- matrix(c(
  40, 40,
  40, 60
), 2, 2, TRUE)

OT <- matrix(c(
  100, 0,
  0, 100
), 2, 2, TRUE)

dimnames(IT) <- dimnames(OT) <- list(names.commodity, names.agent)

addmargins(IT)
addmargins(OT)

#### the model
dst.firm <- node_new(
  "prod",
  type = "SCES",
  es = 1,
  alpha = 1.25, # 100 / (40 + 40)
  beta = prop.table(c(40, 40)),
  "prod", "lab"
)

dst.consumer <- node_new(
  "util",
  type = "SCES",
  es = 1, alpha = 1,
  beta = prop.table(c(40, 60)),
  "prod", "lab"
)

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

ge.benchmark <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0,
```

```
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "lab"
)

ge.benchmark$p
ge.benchmark$D
ge.benchmark$S

addmargins(ge.benchmark$DV)
addmargins(ge.benchmark$SV)

## the same as above
ge <- sdm2(
  A = function(state) {
    a.firm <- SCES_A(es = 1, alpha = 1.25, Beta = prop.table(c(40, 40)), p = state$p)
    a.consumer <- SCES_A(es = 1, alpha = 1.25, Beta = prop.table(c(40, 60)), p = state$p)
    cbind(a.firm, a.consumer)
  },
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "lab"
)

## technology progress
dstl[[1]]$alpha <- 2.5

ge.TP <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
```

```
  numeraire = "lab"
)

ge.TP$p
ge.TP$D
ge.TP$S

addmargins(ge.TP$DV)
addmargins(ge.TP$SV)
```

---

gemInputOutputTable_2_7_2

A Two-Country General Equilibrium Model

Description

A two-country general equilibrium model. This general equilibrium model is based on a two-country (i.e. CHN and ROW) input-output table consisting of an input part and an output part. Each country contains 2 sectors and 7 commodities (or subjects). The 2 sectors are firm and household. The 7 commodities (or subjects) are product, labor, capital goods, bond, tax, dividend, tariff. Hence the input-output table has 14 rows and 4 columns.

Usage

```
gemInputOutputTable_2_7_2(
  IT,
  OT,
  es.DIProduct.firm.CHN = 3,
  es.DIProduct.firm.ROW = 3,
  es.laborCapital.firm.CHN = 0.75,
  es.laborCapital.firm.ROW = 0.75,
  es.household.CHN = 3,
  es.household.ROW = 3,
  return.dstl = FALSE,
  ...
)
```

Arguments

`IT` the input part of the input-output table.

`OT` the output part of the input-output table.

`es.DIProduct.firm.CHN`
: the elasticity of substitution between domestic product and imported product used by the production sector of CHN.

`es.DIProduct.firm.ROW`
: the elasticity of substitution between domestic product and imported product used by the production sector of ROW.

`es.laborCapital.firm.CHN`
: the elasticity of substitution between labor and capital goods used by the production sector of CHN.

`es.laborCapital.firm.ROW`
: the elasticity of substitution between labor and capital goods used by the production sector of ROW.

`es.household.CHN`
: the elasticity of substitution between domestic product and imported product used by the consumption sector of CHN.

`es.household.ROW`
: the elasticity of substitution between domestic product and imported product used by the consumption sector of ROW.

`return.dstl`
: If TRUE, the demand structure tree will be returned.

`...`
: arguments to be transferred to the function `sdm2`.

Value

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

- p - the price vector with CHN labor as numeraire.
- dstl - the demand structure tree list of sectors if return.dstl == TRUE.
- ... - some elements returned by the function `sdm2`.

Examples

```
IT <- matrix(c(
  142, 84, 13, 4.1,
  47, 0, 0, 0,
  13, 0, 0, 0,
  0, 0, 0, 3.4,
  9.3, 0, 0, 0,
  22, 0, 0, 0,
  0.15, 0.091, 0, 0,
  10, 6, 381, 451,
  0, 0, 252, 0,
  0, 0, 81, 0,
  0, 4.9, 0, 0,
  0, 0, 26, 0,
  0, 0, 92, 0,
  0, 0, 1.9, 0.59
), 14, 4, TRUE)

OT <- matrix(c(
  244, 0, 0, 0,
  0, 47, 0, 0,
  0, 13, 0, 0,
  0, 3.4, 0, 0,
  0, 9.3, 0, 0,
  0, 22, 0, 0,
```

```
  0, 0.24, 0, 0,
  0, 0, 849, 0,
  0, 0, 0, 252,
  0, 0, 0, 81,
  0, 0, 0, 4.9,
  0, 0, 0, 26,
  0, 0, 0, 92,
  0, 0, 0, 2.5
), 14, 4, TRUE)

dimnames(IT) <- dimnames(OT) <- list(
  c(
    "product.CHN", "labor.CHN", "capital.CHN", "bond.CHN",
    "tax.CHN", "dividend.CHN", "tariff.CHN",
    "product.ROW", "labor.ROW", "capital.ROW", "bond.ROW",
    "tax.ROW", "dividend.ROW", "tariff.ROW"
  ),
  c(
    "firm.CHN", "household.CHN",
    "firm.ROW", "household.ROW"
  )
)

ge <- gemInputOutputTable_2_7_2(IT, OT, return.dstl = TRUE)
ge$p
ge$z
node_plot(ge$dstl[[1]])
ge$dstl[[1]]$a

## tariff rate change in CHN
dstl <- lapply(ge$dstl, Clone)
tmp <- node_set(dstl[[1]], "cc1.1")
tmp$beta[2] <- tmp$beta[2] * 10

ge.TRC <- sdm2(
  A = dstl, B = ge$B, S0Exg = ge$S0Exg,
  names.commodity = rownames(ge$B),
  names.agent = colnames(ge$B),
  numeraire = "labor.CHN"
)

ge.TRC$p
ge.TRC$z
#### technology progress in CHN
OT.TP <- OT
OT.TP["product.CHN", "firm.CHN"] <- OT["product.CHN", "firm.CHN"] * 1.2

ge.TP <- gemInputOutputTable_2_7_2(IT, OT.TP, return.dstl = TRUE)
ge.TP$p
ge.TP$z
ge.TP$dstl[[1]]$a

#### capital accumulation in CHN
```

```
OT.CA <- OT
OT.CA["capital.CHN", "household.CHN"] <- OT["capital.CHN", "household.CHN"] * 2

ge.CA <- gemInputOutputTable_2_7_2(IT, OT.CA)
ge.CA$p
ge.CA$z

#### labor supply change in CHN
OT.LSC <- OT
OT.LSC["labor.CHN", "household.CHN"] <- OT["labor.CHN", "household.CHN"] * 0.5

ge.LSC <- gemInputOutputTable_2_7_2(IT, OT.LSC)
ge.LSC$p
ge.LSC$z
```

---

gemInputOutputTable_2_7_4 *A Two-Country General Equilibrium Model*

Description

A two-country general equilibrium model. This general equilibrium model is based on a two-country (i.e. CHN and ROW) input-output table consisting of an input part and an output part. Each country contains 4 sectors and 7 commodities (or subjects). The 4 sectors are production, consumption, investment and foreign trade. The 7 commodities (or subjects) are product, labor, capital goods, bond, tax, dividend, imported product. Hence the input-output table has 14 rows and 8 columns.

Usage

```
gemInputOutputTable_2_7_4(
  IT,
  OT,
  es.DIProduct.production.CHN = 3,
  es.DIProduct.production.ROW = 3,
  es.laborCapital.production.CHN = 0.75,
  es.laborCapital.production.ROW = 0.75,
  es.consumption.CHN = 3,
  es.consumption.ROW = 3,
  es.investment.CHN = 3,
  es.investment.ROW = 3,
  return.dstl = FALSE,
  ...
)
```

Arguments

`IT`	the input part of the input-output table.
`OT`	the output part of the input-output table.
`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.
`return.dstl`	If TRUE, the demand structure tree will be returned.
`...`	arguments to be transferred to the function `sdm2`.

Value

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

- p - the price vector with CHN labor as numeraire.
- dstl - the demand structure tree list of sectors if return.dstl == TRUE.
- ... - some elements returned by the function `sdm2`.

Examples

```
IT <- matrix(c(
  30, 12, 9, 0, 0, 0, 0, 13,
  15, 0, 0, 0, 0, 0, 0, 0,
  2, 0, 0, 0, 0, 0, 0, 0,
  0, 9, 0, 0, 0, 2, 0, 0,
```

```
  3, 0, 0, 1, 0, 0, 0, 0,
  6, 0, 0, 0, 0, 0, 0, 0,
  8, 3, 3, 0, 0, 0, 0, 0,
  0, 0, 0, 13, 150, 316, 258, 0,
  0, 0, 0, 0, 288, 0, 0, 0,
  0, 0, 0, 0, 92, 0, 0, 0,
  0, 2, 0, 0, 0, 269, 0, 0,
  0, 0, 0, 0, 35, 0, 0, 1,
  0, 0, 0, 0, 172, 0, 0, 0,
  0, 0, 0, 0, 1, 5, 13, 0
), 14, 8, TRUE)

OT <- matrix(c(
  64, 0, 0, 0, 0, 0, 0, 0,
  0, 15, 0, 0, 0, 0, 0, 0,
  0, 2, 0, 0, 0, 0, 0, 0,
  0, 0, 11, 0, 0, 0, 0, 0,
  0, 3, 0, 0, 0, 0, 0, 0,
  0, 6, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 13, 0, 0, 0, 0,
  0, 0, 0, 0, 738, 0, 0, 0,
  0, 0, 0, 0, 0, 288, 0, 0,
  0, 0, 0, 0, 0, 92, 0, 0,
  0, 0, 0, 0, 0, 0, 271, 0,
  0, 0, 0, 0, 0, 36, 0, 0,
  0, 0, 0, 0, 0, 172, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 14
), 14, 8, TRUE)

dimnames(IT) <- dimnames(OT) <- list(
  c(
    "product.CHN", "labor.CHN", "capital.CHN", "bond.CHN",
    "tax.CHN", "dividend.CHN", "imported.product.CHN",
    "product.ROW", "labor.ROW", "capital.ROW", "bond.ROW",
    "tax.ROW", "dividend.ROW", "imported.product.ROW"
  ),
  c(
    "production.CHN", "consumption.CHN", "investment.CHN", "foreign.trade.CHN",
    "production.ROW", "consumption.ROW", "investment.ROW", "foreign.trade.ROW"
  )
)

ge <- gemInputOutputTable_2_7_4(IT, OT, return.dstl = TRUE)
ge$p
ge$z
node_plot(ge$dstl[[1]])
ge$dstl[[1]]$a

#### technology progress in CHN
OT.TP <- OT
OT.TP["product.CHN", "production.CHN"] <- OT["product.CHN", "production.CHN"] * 1.2

ge.TP <- gemInputOutputTable_2_7_4(IT, OT.TP, return.dstl = TRUE)
```

```
ge.TP$p
ge.TP$z
ge.TP$dstl[[1]]$a

#### capital accumulation in CHN
OT.CA <- OT
OT.CA["capital.CHN", "consumption.CHN"] <- OT["capital.CHN", "consumption.CHN"] * 2

ge.CA <- gemInputOutputTable_2_7_4(IT, OT.CA)
ge.CA$p
ge.CA$z

#### labor supply change in CHN
OT.LSC <- OT
OT.LSC["labor.CHN", "consumption.CHN"] <- OT["labor.CHN", "consumption.CHN"] * 0.5

ge.LSC <- gemInputOutputTable_2_7_4(IT, OT.LSC)
ge.LSC$p
ge.LSC$z

#### tariff rate change in CHN
IT.TRC <- IT
IT.TRC["tax.CHN","foreign.trade.CHN"] <- IT.TRC["tax.CHN","foreign.trade.CHN"] * 1.2
ge.TRC <- gemInputOutputTable_2_7_4(IT.TRC, OT)
ge.TRC$p
ge.TRC$z
```

`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, Table 8.6.1)*

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)

IT <- cbind(d.agri, d.manu, d.serv, d.hh)
OT <- matrix(c(
  1365, 0, 0, 0,
  0, 1725, 0, 0,
  0, 0, 1470, 0,
  0, 0, 0, 850,
  0, 0, 0, 770
), 5, 4, TRUE)

dimnames(IT) <- dimnames(OT) <-
  list(
    c("agri", "manu", "serv", "lab", "cap"),
```

```
    c("agri", "manu", "serv", "hh")
  )

addmargins(IT)
addmargins(OT)

dst.agri <- node_new("sector.agri",
                     type = "SCES", es = es.agri,
                     alpha = 1,
                     beta = prop.table(
                       c(sum(d.agri[1:3]), sum(d.agri[4:5]))
                     ),
                     "cc1.agri", "cc2.agri"
)
node_set(dst.agri, "cc1.agri",
         type = "Leontief",
         a = prop.table(d.agri[1:3]),
         "agri", "manu", "serv"
)
node_set(dst.agri, "cc2.agri",
         type = "SCES", es = es.VA.agri,
         alpha = 1,
         beta = prop.table(d.agri[4:5]),
         "lab", "cap"
)


dst.manu <- node_new("sector.manu",
                     type = "SCES", es = es.manu,
                     alpha = 1,
                     beta = prop.table(
                       c(sum(d.manu[1:3]), sum(d.manu[4:5]))
                     ),
                     "cc1.manu", "cc2.manu"
)
node_set(dst.manu, "cc1.manu",
         type = "Leontief",
         a = prop.table(d.manu[1:3]),
         "agri", "manu", "serv"
)
node_set(dst.manu, "cc2.manu",
         type = "SCES", es = es.VA.manu,
         alpha = 1,
         beta = prop.table(d.manu[4:5]),
         "lab", "cap"
)

dst.serv <- node_new("sector.serv",
                     type = "SCES", es = es.serv,
                     alpha = 1,
                     beta = prop.table(
                       c(sum(d.serv[1:3]), sum(d.serv[4:5]))
                     ),
```

```
                     "cc1.serv", "cc2.serv"
)
node_set(dst.serv, "cc1.serv",
         type = "Leontief",
         a = prop.table(d.serv[1:3]),
         "agri", "manu", "serv"
)
node_set(dst.serv, "cc2.serv",
         type = "SCES", es = es.VA.serv,
         alpha = 1,
         beta = prop.table(d.serv[4:5]),
         "lab", "cap"
)

##
dst.hh <- node_new("sector.hh",
                   type = "SCES", es = 0.5,
                   alpha = 1,
                   beta = prop.table(d.hh[1:3]),
                   "agri", "manu", "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

## capital supply change
ge.CSC <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1,
    0, 0, 0, 1
  ), 5, 4, TRUE),
  S0Exg = {
    tmp <- matrix(NA, 5, 4)
    tmp[4, 4] <- 850
    tmp[5, 4] <- 770
    tmp
  },
  names.commodity = c("agri", "manu", "serv", "lab", "cap"),
  names.agent = c("agri", "manu", "serv", "hh"),
  numeraire = "lab",
  ts = TRUE,
  numberOfPeriods = 100,
  maxIteration = 1,
```

```
  z0 = c(1365, 1725, 1470, 1620),
  p0 = rep(1, 5),
  policy = function(time, state) {
    if (time >= 5) {
      state$S[5, 4] <- 880
    }
    state
  }
)

matplot(ge.CSC$ts.p, type = "l")
matplot(ge.CSC$ts.z, type = "l")

## economic fluctuation: a sticky-price path
de <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1,
    0, 0, 0, 1
  ), 5, 4, TRUE),
  S0Exg = {
    tmp <- matrix(NA, 5, 4)
    tmp[4, 4] <- 850
    tmp[5, 4] <- 770
    tmp
  },
  names.commodity = c("agri", "manu", "serv", "lab", "cap"),
  names.agent = c("agri", "manu", "serv", "hh"),
  numeraire = "lab",
  ts = TRUE,
  numberOfPeriods = 50,
  maxIteration = 1,
  z0 = c(1365, 1725, 1470, 1620),
  p0 = rep(1, 5),
  policy = list(
    function(time, state) {
      if (time >= 5) {
        state$S[5, 4] <- 880
      }
      state
    },
    makePolicyStickyPrice(0.5)
  ),
  priceAdjustmentVelocity = 0
)

matplot(de$ts.p, type = "o", pch = 20)
matplot(de$ts.z, type = "o", pch = 20)
```

gemInputOutputTable_5_5

General Equilibrium Models based on a 5x5 Input-Output Table (see Zhang Xin, 2017, Table 3.2.1)

Description

Some general equilibrium models based on a 5x5 input-output table (see Zhang Xin, 2017, Table 3.2.1).

Usage

```
gemInputOutputTable_5_5(...)
```

Arguments

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

References

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

Examples

```
names.commodity <- c("agri", "manu", "serv", "lab", "cap")
names.agent <- c("agri", "manu", "serv", "consumer", "investor")

IT <- matrix(c(
  200, 300, 150, 280, 70,
  80, 400, 250, 550, 320,
  30, 420, 240, 350, 110,
  500, 250, 330, 0, 0,
  190, 230, 180, 0, 0
), 5, 5, TRUE)

OT <- matrix(c(
  1000, 0, 0, 0, 0,
  0, 1600, 0, 0, 0,
  0, 0, 1150, 0, 0,
  0, 0, 0, 758.5714, 321.4286,
  0, 0, 0, 421.4286, 178.5714
), 5, 5, TRUE)

dimnames(IT) <- dimnames(OT) <- list(names.commodity, names.agent)

addmargins(IT)
addmargins(OT)
```

```
#### a model with non-nested production functions (demand structure trees)
dst.agri <- node_new("output",
  type = "SCES", es = 1, alpha = 1,
  beta = prop.table(c(200, 80, 30, 500, 190)),
  "agri", "manu", "serv", "lab", "cap"
)

dst.manu <- node_new("output",
  type = "SCES", es = 1, alpha = 1,
  beta = prop.table(c(300, 400, 420, 250, 230)),
  "agri", "manu", "serv", "lab", "cap"
)

dst.serv <- node_new("output",
  type = "SCES", es = 1, alpha = 1,
  beta = prop.table(c(150, 250, 240, 330, 180)),
  "agri", "manu", "serv", "lab", "cap"
)

dst.consumer <- node_new("util",
  type = "SCES", es = 0.5, alpha = 1,
  beta = prop.table(c(280, 550, 350)),
  "agri", "manu", "serv"
)

dst.investor <- node_new("util",
  type = "SCES", es = 0.5, alpha = 1,
  beta = prop.table(c(70, 320, 110)),
  "agri", "manu", "serv"
)

ge1.benchmark <- sdm2(list(dst.agri, dst.manu, dst.serv, dst.consumer, dst.investor),
  B = matrix(c(
    1, 0, 0, 0, 0,
    0, 1, 0, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0
  ), 5, 5, TRUE),
  S0Exg = {
    S0Exg <- matrix(NA, 5, 5)
    S0Exg[4:5, 4] <- c(1080, 600) * (1180 / (1180 + 500))
    S0Exg[4:5, 5] <- c(1080, 600) * (500 / (1180 + 500))
    S0Exg
  },
  names.commodity = c("agri", "manu", "serv", "lab", "cap"),
  names.agent = c("agri", "manu", "serv", "consumer", "investor"),
  numeraire = c("lab")
)

addmargins(ge1.benchmark$D)
addmargins(ge1.benchmark$S)
```

```
#### a model with nested production functions (demand structure trees)
dst.agri <- node_new("output",
  type = "SCES", es = 0, alpha = 1,
  beta = prop.table(c(200 + 80 + 30, 500 + 190)),
  "cc.II", "cc.VA"
)
node_set(dst.agri, "cc.II",
  type = "SCES", es = 0, alpha = 1,
  beta = prop.table(c(200, 80, 30)),
  "agri", "manu", "serv"
)
node_set(dst.agri, "cc.VA",
  type = "SCES", es = 0.5, alpha = 1,
  beta = prop.table(c(500, 190)),
  "lab", "cap"
)

dst.manu <- node_new("output",
  type = "SCES", es = 0, alpha = 1,
  beta = prop.table(c(300 + 400 + 420, 250 + 230)),
  "cc.II", "cc.VA"
)
node_set(dst.manu, "cc.II",
  type = "SCES", es = 0, alpha = 1,
  beta = prop.table(c(300, 400, 420)),
  "agri", "manu", "serv"
)
node_set(dst.manu, "cc.VA",
  type = "SCES", es = 0.5, alpha = 1,
  beta = prop.table(c(250, 230)),
  "lab", "cap"
)

dst.serv <- node_new("output",
  type = "SCES", es = 0, alpha = 1,
  beta = prop.table(c(150 + 250 + 240, 330 + 180)),
  "cc.II", "cc.VA"
)
node_set(dst.serv, "cc.II",
  type = "SCES", es = 0, alpha = 1,
  beta = prop.table(c(150, 250, 240)),
  "agri", "manu", "serv"
)
node_set(dst.serv, "cc.VA",
  type = "SCES", es = 0.5, alpha = 1,
  beta = prop.table(c(330, 180)),
  "lab", "cap"
)

dst.consumer <- node_new("util",
  type = "SCES", es = 0.5, alpha = 1,
  beta = prop.table(c(280, 550, 350)),
```

```
  "agri", "manu", "serv"
)

dst.investor <- node_new("util",
  type = "SCES", es = 0.5, alpha = 1,
  beta = prop.table(c(70, 320, 110)),
  "agri", "manu", "serv"
)

ge2.benchmark <- sdm2(list(dst.agri, dst.manu, dst.serv, dst.consumer, dst.investor),
  B = matrix(c(
    1, 0, 0, 0, 0,
    0, 1, 0, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0
  ), 5, 5, TRUE),
  S0Exg = {
    S0Exg <- matrix(NA, 5, 5)
    S0Exg[4:5, 4] <- c(1080, 600) * (1180 / (1180 + 500))
    S0Exg[4:5, 5] <- c(1080, 600) * (500 / (1180 + 500))
    S0Exg
  },
  names.commodity = c("agri", "manu", "serv", "lab", "cap"),
  names.agent = c("agri", "manu", "serv", "consumer", "investor"),
  numeraire = c("lab")
)

addmargins(ge2.benchmark$D)
addmargins(ge2.benchmark$S)
```

gemInputOutputTable_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

```
gemInputOutputTable_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 sdm2 function.

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 <- gemInputOutputTable_7_4(
  IT = IT2017,
  OT = OT2017
)

#### labor supply reduction
OTLSR <- OT2017
OTLSR["lab", "sector.hh"] <- OTLSR["lab", "sector.hh"] * 0.9
geLSR <- gemInputOutputTable_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 <- gemInputOutputTable_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 <- gemInputOutputTable_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 <- gemInputOutputTable_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 <- gemInputOutputTable_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 <- gemInputOutputTable_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 <- gemInputOutputTable_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 <- gemInputOutputTable_7_4(
  IT = IT.TC2,
  OT = OT2017
)

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.CI = 0,
  es.FT = 0,
  es.VA.agri = 0.25,
  es.VA.manu = 0.5,
  es.VA.serv = 0.8,
  es.prodDM = 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.CI`	the elasticity of substitution among products used by the CI 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.prodDM`	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 consumption and (temporarily unproductive) investment sector (CI sector), 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 CI sector uses 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.CI",
  "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.CI"] <- OT.CA["cap", "sector.CI"] * 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_easy_5_4

An Easy General Equilibrium Model based on a 5x4 Input-Output Table (see Zhang Xin, 2017, Table 8.6.1)

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)
)
```

---

gemInputOutputTable_SCES_3_3

A SCES-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_SCES_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_SCES_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
)
```

gemIntertemporalStochastic_Bank_3Periods *An Intertemporal Stochastic Model with a Consumer and Some Banks*

Description

An intertemporal stochastic model with a consumer and some banks. In the model the consumer will live for three periods. There is one natural state in the first period, and two natural states in the second and third period.

Usage

```
gemIntertemporalStochastic_Bank_3Periods(...)
```

Arguments

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

Examples

```
dst.bank1a <- node_new(
  "payoff2.1&2.2",
  type = "Leontief", a = 1,
  "payoff1"
)

dst.bank1b <- node_new(
  "payoff3.1&3.2",
  type = "Leontief", a = 1,
  "payoff1"
)

dst.bank2.1 <- node_new(
  "payoff3.1",
  type = "Leontief", a = 1,
  "payoff2.1"
)

dst.bank2.2 <- node_new(
  "payoff3.2",
  type = "Leontief", a = 1,
  "payoff2.2"
)

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1, beta = c(1/3, 1/6, 1/6, 1/6, 1/6),
  "payoff1", "payoff2.1", "payoff2.2", "payoff3.1", "payoff3.2"
)
```

```
ge <- sdm2(
  A = list(dst.bank1a, dst.bank1b,dst.bank2.1,dst.bank2.2,dst.consumer),
  B = matrix(c(
    0,   0,0, 0,0,
    1.1, 0,0, 0,0,
    1.1, 0,0, 0, 0,
    0,   1.5, 1.1, 0,0,
    0,   1.5, 0, 1.1,0
  ), 5, 5, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA, NA, 1,
    NA, NA, NA, NA, 1,
    NA, NA, NA, NA, 0,
    NA, NA, NA, NA, 0,
    NA, NA, NA, NA, 0
  ), 5, 5, TRUE),
  names.commodity = c("payoff1", "payoff2.1", "payoff2.2", "payoff3.1", "payoff3.2"),
  names.agent = c("bank1a","bank1b","bank2.1","bank2.2", "consumer"),
  numeraire = "payoff1"
)

ge$p
round(ge$D, 4)
round(ge$S, 4)

#### the general equilibrium in the first natural state in period 2
dst.bank2.1 <- node_new(
  "payoff3.1",
  type = "Leontief", a = 1,
  "payoff2.1"
)

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "payoff2.1", "payoff3.1"
)

ge2.1 <- sdm2(
  A = list(dst.bank2.1,
           dst.consumer),
  B = matrix(c(
    0,   0,
    1.1, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, 1.3084,
    NA, 0.4599
  ), 2, 2, TRUE),
  names.commodity = c("payoff2.1", "payoff3.1"),
  names.agent = c("bank2.1", "consumer"),
  numeraire = "payoff2.1"
```

```
)

ge2.1$p
round(ge2.1$D, 4)
round(ge2.1$S, 4)

## the general equilibrium in an unanticipated natural state in period 2
ge2.3 <- sdm2(
  A = list(dst.bank2.1,
           dst.consumer),
  B = matrix(c(
    0,   0,
    1.1, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, 0.4,
    NA, 0.4599
  ), 2, 2, TRUE),
  names.commodity = c("payoff2.1", "payoff3.1"),
  names.agent = c("bank2.1", "consumer"),
  numeraire = "payoff2.1"
)

ge2.3$p
round(ge2.3$D, 4)
round(ge2.3$S, 4)
```

---

gemIntertemporal_2_2 *Some Examples of a 2-by-2 Intertemporal Equilibrium Model*

Description

Some examples of a 2-by-2 intertemporal equilibrium model.

In these examples, there is an np-period-lived consumer maximizing intertemporal utility, and there is a type of firm which produces from period 1 to np-1. There are two commodities, i.e. product and labor. Suppose the consumer has some product in the first period. That is, the product supply in the first period is an exogenous variable.

Usage

```
gemIntertemporal_2_2(...)
```

Arguments

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

Examples

```
#### an example with a Cobb-Douglas intertemporal utility function
np <- 5 # the number of periods.
y1 <- 150
S <- matrix(NA, 2 * np - 1, np)
S[(np + 1):(2 * np - 1), np] <- 100
S[1, np] <- y1

B <- matrix(0, 2 * np - 1, np)
B[2:np, 1:(np - 1)] <- diag(np - 1)

dstl.firm <- list()
for (k in 1:(np - 1)) {
  dstl.firm[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 2, beta = c(0.5, 0.5),
    paste0("prod", k), paste0("lab", k)
  )
}

dst.consumer.CD <- node_new(
  "util",
  type = "CD",
  alpha = 1,  beta = prop.table(rep(1, np)),
  paste0("prod", 1:np)
)

f <- function(dstl) {
  sdm2(
    A = dstl,
    B = B,
    S0Exg = S,
    names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:(np - 1))),
    names.agent = c(paste0("firm", 1:(np - 1)), "consumer"),
    numeraire = "prod1",
    ts = TRUE
  )
}

ge <- f(c(dstl.firm, dst.consumer.CD))

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV

## an example with a Leontief intertemporal utility function
dst.consumer.Leontief <- node_new(
```

```
    "util",
    type = "Leontief",
    a = rep(1, np),
    paste0("prod", 1:np)
  )

  ge2 <- f(c(dstl.firm, dst.consumer.Leontief))

  ge2$p
  ge2$z
  ge2$D
  ge2$S
  ge2$DV
  ge2$SV
```

---

gemIntertemporal_3_3 *Some Examples of Intertemporal Models with One Consumer and Two Types of Firms*

Description

Some examples of intertemporal models with one consumer and two types of firms. There are three commodities (i.e. corn, iron and labor). The consumer may consume corn and iron in each period, and may have a nested intertemporal utility function.

Usage

```
gemIntertemporal_3_3(...)
```

Arguments

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

References

Zen Xiangjin (1995, ISBN: 7030046560). Basics of Economic Cybernetics. Beijing: Science Press. (In Chinese)

Examples

```
#### an example with a consumer with a nested intertemporal utility function
np <- 5 # the number of periods, firms.

## exogenous supply matrix
n <- 3 * np - 1
m <- 2 * (np - 1) + 1
S <- matrix(NA, n, m)
```

```
S[(2 * np + 1):(3 * np - 1), m] <- 100
S[1, m] <- 25 #corn1
S[np + 1, m] <- 100 #iron1

B <- matrix(0, n, m)
B[2:np, 1:(np - 1)] <- B[(np + 2):(2 * np), np:(m - 1)] <-
  diag(np - 1)

dstl.firm.corn <- dstl.firm.iron  <- list()
for (k in 1:(np - 1)) {
  dstl.firm.corn[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 1,
    beta = c(0.5, 0.5),
    paste0("iron", k),
    paste0("lab", k)
  )

  dstl.firm.iron[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 2,
    beta = c(0.5, 0.5),
    paste0("iron", k),
    paste0("lab", k)
  )
}

dst.consumer <- node_new(
  "util",
  type = "CD",
  alpha = 1,
  beta = prop.table(rep(1, np)),
  paste0("cc", 1:np)
)
for (k in 1:np) {
  node_set(
    dst.consumer,
    paste0("cc", k),
    type = "CD",
    alpha = 1,
    beta = c(0.5, 0.5),
    paste0("corn", k),
    paste0("iron", k)
  )
}

ge <-   sdm2(
  A = c(dstl.firm.corn, dstl.firm.iron, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("corn", 1:np),
```

```
                      paste0("iron", 1:np),
                      paste0("lab", 1:(np - 1))),
  names.agent = c(paste0("firm.corn", 1:(np - 1)),
                  paste0("firm.iron", 1:(np - 1)),
                  "consumer"),
  numeraire = "lab1",
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV

#### an example with a consumer with a non-nested intertemporal utility function
np <- 3 # the number of periods, firms.

## There are np types of corn, np-1 types of iron and np-1 types of labor.
## There are np-1 corn firms, np-2 iron firms and one consumer.
n <- 3 * np - 2
m <- 2 * np - 2

## exogenous supply matrix
S <- matrix(NA, n, m)
S[(2 * np):n, m] <- 100
S[1, m] <- 25 #corn1
S[np + 1, m] <- 100 #iron1

B <- matrix(0, n, m)
B[2:np, 1:(np - 1)]  <- diag(np - 1)
B[(np + 2):(2 * np - 1), np:(m - 1)] <- diag(np - 2)

dstl.firm.corn <- dstl.firm.iron  <- list()
for (k in 1:(np - 1)) {
  dstl.firm.corn[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 1,
    beta = c(0.5, 0.5),
    paste0("iron", k),
    paste0("lab", k)
  )
}

for (k in seq_along(np:(2 * np - 3))) {
  dstl.firm.iron[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 2,
    beta = c(0.5, 0.5),
    paste0("iron", k),
```

```
    paste0("lab", k)
  )
}

dst.consumer <- node_new(
  "util",
  type = "CD",
  alpha = 1,
  beta = prop.table(rep(1, np)),
  paste0("corn", 1:np)
)

ge <-   sdm2(
  A = c(dstl.firm.corn, dstl.firm.iron, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("corn", 1:np),
                      paste0("iron", 1:(np - 1)),
                      paste0("lab", 1:(np - 1))),
  names.agent = c(paste0("firm.corn", 1:(np - 1)),
                  paste0("firm.iron", 1:(np - 2)),
                  "consumer"),
  numeraire = "lab1",
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV

#### an example of Zeng (1995, page 227)
ic1 <- 1 / 10 #input coefficient
ic2 <- 1 / 7
dc1 <- 2 / 3 #depreciation coefficient
dc2 <- 9 / 10

ge <- sdm2(
  A = {
    #corn, iron1, iron2, iron3, iron4
    a1.1 <- c(0, ic1, 0, 0, 0)
    a1.2 <- c(0, ic2, 0, 0, 0)
    a2.1 <- c(0, 0, ic1, 0, 0)
    a2.2 <- c(0, 0, ic2, 0, 0)
    a3.1 <- c(0, 0, 0, ic1, 0)
    a3.2 <- c(0, 0, 0, ic2, 0)
    a4.1 <- c(0, 0, 0, 0, ic1)
    a4.2 <- c(0, 0, 0, 0, ic2)

    a.consumer <- c(1, 0, 0, 0, 0)
```

```
    cbind(a1.1, a1.2, a2.1, a2.2, a3.1, a3.2, a4.1, a4.2, a.consumer)
  },
  B = {
    b1.1 <- c(1, 0, ic1 * dc1, 0, 0)
    b1.2 <- c(1, 0, ic2 * dc2, 0, 0)
    b2.1 <- c(1, 0, 0, ic1 * dc1, 0)
    b2.2 <- c(1, 0, 0, ic2 * dc2, 0)
    b3.1 <- c(1, 0, 0, 0, ic1 * dc1)
    b3.2 <- c(1, 0, 0, 0, ic2 * dc2)
    b4.1 <- c(1, 0, 0, 0, 0)
    b4.2 <- c(1, 0, 0, 0, 0)
    b.consumer <- c(0, 0, 0, 0, 0)

    cbind(b1.1, b1.2, b2.1, b2.2, b3.1, b3.2, b4.1, b4.2, b.consumer)
  },
  S0Exg = {
    tmp <- matrix(NA, 5, 9)
    tmp[2, 9] <- 100
    tmp
  },
  names.commodity = c("corn", paste0("iron", 1:4)),
  names.agent = c(paste0("firm", 1:8), "consumer"),
  numeraire = "corn",
  policy = makePolicyMeanValue(30),
  priceAdjustmentVelocity = 0.05,
  maxIteration = 1,
  numberOfPeriods = 1000,
  ts = TRUE
)

matplot(ge$ts.z, type = "l")

ge$p
ge$z
ge$D
ge$S
```

`gemIntertemporal_3_4` *Some Examples of Intertemporal Models with Two Consumers and Two Types of Firms*

Description

Some examples of intertemporal models with two consumers and two types of firms.

Usage

```
gemIntertemporal_3_4(...)
```

Arguments

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

Examples

```
#### an example with a Cobb-Douglas intertemporal utility function
np <- 5 # the number of periods, firms.

n <- 3 * np - 1
m <- 2 * (np - 1) + 2

## exogenous supply matrix
S <- matrix(NA, n, m)
S[(2 * np + 1):(3 * np - 1), (m - 1):m] <- 100
S[1, (m - 1):m] <- 25 #corn1
S[np + 1, (m - 1):m] <- 100 #iron1

B <- matrix(0, n, m)
B[2:np, 1:(np - 1)] <- B[(np + 2):(2 * np), np:(m - 2)] <-
  diag(np - 1)

dstl.firm.corn <- dstl.firm.iron  <- list()
for (k in 1:(np - 1)) {
  dstl.firm.corn[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 1,
    beta = c(0.5, 0.5),
    paste0("iron", k),
    paste0("lab", k)
  )

  dstl.firm.iron[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 2,
    beta = c(0.5, 0.5),
    paste0("iron", k),
    paste0("lab", k)
  )
}

dst.consumer1 <- node_new(
  "util",
  type = "CD",
  alpha = 1,
  beta = prop.table(rep(1, np)),
  paste0("corn", 1:np)
)

dst.consumer2 <- node_new(
```

```
    "util",
    type = "CD",
    alpha = 1,
    beta = prop.table(rep(1, np)),
    paste0("cc", 1:np)
  )
  for (k in 1:np) {
    node_set(
      dst.consumer2,
      paste0("cc", k),
      type = "CD",
      alpha = 1,
      beta = c(0.5, 0.5),
      paste0("corn", k),
      paste0("iron", k)
    )
  }

  ge <-   sdm2(
    A = c(dstl.firm.corn, dstl.firm.iron, dst.consumer1, dst.consumer2),
    B = B,
    S0Exg = S,
    names.commodity = c(paste0("corn", 1:np),
                        paste0("iron", 1:np),
                        paste0("lab", 1:(np - 1))),
    names.agent = c(
      paste0("firm.corn", 1:(np - 1)),
      paste0("firm.iron", 1:(np - 1)),
      "consumer1",
      "consumer2"
    ),
    numeraire = "lab1",
    ts = TRUE
  )

  ge$p
  ge$z
  ge$D
  ge$S
  ge$DV
  ge$SV
```

---

gemIntertemporal_Bank_1_2

Some Examples of an Intertemporal Model with a Consumer and a Type of Bank

---

Description

Some examples of an intertemporal model with a consumer and a type of bank. These models can be used to solve some intertemporal savings problems. Below is an example.

A np-period-lived consumer has some payoff (or cash, exhaustible resource etc.) in each period. In each period the consumer can use payoff for consumption or save payoff into bank. The interest rate is given. The consumer has a SCES intertemporal utility function and attempts to maximize intertemporal utility by saving.

Usage

```
gemIntertemporal_Bank_1_2(...)
```

Arguments

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

Examples

```
#### an example with a 5-period-lived consumer
np <- 5 # the number of periods

interest.rate <- 0.1
S <- matrix(NA, np, np)
S[1:np, np] <- 100 / (np:1)

B <- matrix(0, np, np)
B[2:np, 1:(np - 1)] <- diag(np - 1)

dstl.bank <- list()
for (k in 1:(np - 1)) {
  dstl.bank[[k]] <- node_new("output",
                             type = "Leontief",
                             a = 1 / (1 + interest.rate),
                             paste0("payoff", k))
}

dst.consumer <- node_new(
  "util",
  type = "SCES",
  es = 1,
  alpha = 1,
  beta = prop.table(1:np),
  paste0("payoff", 1:np)
)

ge <- sdm2(
  A = c(dstl.bank, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = paste0("payoff", 1:np),
```

```
  names.agent = c(paste0("bank", 1:(np - 1)), "consumer"),
  numeraire = "payoff1",
  policy = makePolicyMeanValue(30),
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
growth_rate(ge$p)

##
dst.consumer$es <- 0
dst.consumer$beta <- rep(1 / np, np)
S[1:np, np] <- 100 / (1:np)
ge <- sdm2(
  A = c(dstl.bank, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = paste0("payoff", 1:np),
  names.agent = c(paste0("bank", 1:(np - 1)), "consumer"),
  numeraire = "payoff1",
  policy = makePolicyMeanValue(30),
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
```

---

gemIntertemporal_Bank_1_3 *Some Examples of an Intertemporal Model with Two Consumers and a Type of Bank*

Description

Some examples of an intertemporal model with two consumers and a type of bank.

Usage

```
gemIntertemporal_Bank_1_3(...)
```

Arguments

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

See Also

gemIntertemporal_Bank_1_2

Examples

```
#### an example with a 5-period-lived consumer
np <- 5 # the number of periods

interest.rate <- 0.1
S <- matrix(NA, np, np+1)
S[1:np, np] <- 100/(np:1)
S[1:np, np+1] <- 100/(1:np)

B <- matrix(0, np, np+1)
B[2:np, 1:(np-1)] <- diag(np-1)

dstl.bank <- list()
for (k in 1:(np-1)) {
  dstl.bank[[k]] <- node_new(
    "output",
    type = "Leontief",
    a = 1/(1+interest.rate),
    paste0("payoff", k)
  )
}

dst.consumer1 <- node_new(
  "util",
  type = "SCES",
  es = 1, alpha = 1,
  beta = prop.table(1:np),
  paste0("payoff", 1:np)
)

dst.consumer2 <- node_new(
  "util",
  type = "SCES",
  es = 1, alpha = 1,
  beta = prop.table(np:1),
  paste0("payoff", 1:np)
)

ge <- sdm2(
  A = c(dstl.bank, dst.consumer1, dst.consumer2),
  B = B,
  S0Exg = S,
  names.commodity = paste0("payoff", 1:np),
```

```
  names.agent = c(paste0("bank", 1:(np-1)), "consumer1","consumer2"),
  numeraire = "payoff1",
  policy = makePolicyMeanValue(30),
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
growth_rate(ge$p)

##
dst.consumer1$es <- 0
dst.consumer1$beta <- rep(1/np, np)

ge <- sdm2(
  A = c(dstl.bank, dst.consumer1, dst.consumer2),
  B = B,
  S0Exg = S,
  names.commodity = paste0("payoff", 1:np),
  names.agent = c(paste0("bank", 1:(np-1)), "consumer1","consumer2"),
  numeraire = "payoff1",
  policy = makePolicyMeanValue(30),
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
growth_rate(ge$p)
```

---

gemIntertemporal_Stochastic_2_2 *An Intertemporal Model with Uncertainty*

Description

An intertemporal model with uncertainty. In the model the consumer will live for three periods and has a von Neumann-Morgenstern expected utility function. There is one natural state in the first period, and two natural states in the second and third period.

Usage

```
gemIntertemporal_Stochastic_2_2(...)
```

Arguments

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

Examples

```
dst.firm1 <- node_new(
  "prod2",
  type = "CD", alpha = 2,
  beta = c(0.5, 0.5),
  "lab1", "prod1"
)

dst.firm2.1 <- node_new(
  "prod3.1",
  type = "CD", alpha = 2,
  beta = c(0.5, 0.5),
  "prod2.1","lab2.1"
)

dst.firm2.2 <- node_new(
  "prod3.2",
  type = "CD", alpha = 1,
  beta = c(0.4, 0.6),
  "prod2.2","lab2.2"
)

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = rep(1/5, 5),
  "prod1","prod2.1","prod2.2",
  "prod3.1","prod3.2"
)

ge <- sdm2(
  A = c(dst.firm1, dst.firm2.1,dst.firm2.2,
        dst.consumer),
  B = matrix(c(0, 0, 0, 0,
               1, 0, 0, 0,
               1, 0, 0, 0,
               0, 1, 0, 0,
               0, 0, 1, 0,
               0, 0, 0, 0,
               0, 0, 0, 0,
               0, 0, 0, 0),8,4,TRUE),
  S0Exg = matrix(c(NA, NA, NA, 50,
                   NA, NA, NA, NA,
                   NA, NA, NA, NA,
                   NA, NA, NA, NA,
                   NA, NA, NA, NA,
                   NA, NA, NA,100,
```

```
                   NA, NA, NA,100,
                   NA, NA, NA,100),8,4,TRUE),
  names.commodity = c("prod1", "prod2.1","prod2.2",
                      "prod3.1","prod3.2",
                      "lab1", "lab2.1","lab2.2"),
  names.agent = c("firm1", "firm2.1","firm2.2",
                  "consumer"),
  numeraire = "lab1",
  policy = makePolicyMeanValue(30),
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
```

`gemIntertemporal_TimeCircle_2_2`
Some Examples of a 2-by-2 Time Circle Equilibrium Model

Description

Some examples of a 2-by-2 (intertemporal) time circle equilibrium model. In a time circle model, the economy borrows some resources from the outside in the beginning, and will repay it after the economy ends.

In these examples, there is an np-period-lived consumer maximizing intertemporal utility, and there is a type of firm which produces from period 1 to np. There are two commodities, i.e. product and labor. Suppose the firm can borrow some product from outside in the first period and return them in the (np+1)-th period. And the supply of product in the first period can be regarded as the output of the firm in the (np+1)-th period. Hence the product supply in the first period is an endogenous variable. Suppose that the amount returned is zeta times the amount borrowed.

Usage

```
gemIntertemporal_TimeCircle_2_2(...)
```

Arguments

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

See Also

gemOLGTimeCircle

Examples

```
#### an example with a Cobb-Douglas intertemporal utility function
np <- 5 # the number of periods, firms.

zeta <- 1.25
S <- matrix(NA, 2 * np, np + 1)
S[(np + 1):(2 * np), np + 1] <- 100

B <- matrix(0, 2 * np, np + 1)
B[1:np, 1:np] <- diag(np)[, c(2:np, 1)]
B[1, np] <- 1 / zeta

dstl.firm <- list()
for (k in 1:np) {
  dstl.firm[[k]] <- node_new(
    "prod",
    type = "CD", alpha = 2,
    beta = c(0.5, 0.5),
    paste0("lab", k), paste0("prod", k)
  )
}

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = prop.table(rep(1, np)),
  paste0("prod", 1:np)
)

ge <- sdm2(
  A = c(dstl.firm, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:np)),
  names.agent = c(paste0("firm", 1:np), "consumer"),
  numeraire = "lab1",
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV

## an example with a Leontief intertemporal utility function
dst.consumer <- node_new(
  "util",
  type = "Leontief",
  a = rep(1, np),
```

```
  paste0("prod", 1:np)
)

ge2 <- sdm2(
  A = c(dstl.firm, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:np)),
  names.agent = c(paste0("firm", 1:np), "consumer"),
  numeraire = "lab1",
  ts = TRUE
)

ge2$p
ge2$z
ge2$D
ge2$S
ge2$DV
ge2$SV

## Use a mean-value policy function to accelerate convergence.
ge3 <- sdm2(
  A = c(dstl.firm, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:np)),
  names.agent = c(paste0("firm", 1:np), "consumer"),
  numeraire = "lab1",
  ts = TRUE,
  policy = makePolicyMeanValue(30)
)

#### an example with a linear intertemporal utility function (e.g. beta1*x1+beta2*x2)
## The demand structure of the consumer will be adjusted sluggishly to accelerate convergence.
np <- 5 # the number of periods, firms.
rho <- 0.9 # subjective discount factor

beta.consumer <- rep(rho ^ (0:(np - 1)))

zeta <-  (1 / rho) ^ np

S <- matrix(NA, 2 * np, np + 1)
S[(np + 1):(2 * np), np + 1] <- 100

B <- matrix(0, 2 * np, np + 1)
B[1:np, 1:np] <- diag(np)[, c(2:np, 1)]
B[1, np] <- 1 / zeta

dstl.firm <- list()
for (k in 1:np) {
  dstl.firm[[k]] <- node_new(
    "prod",
    type = "CD",
```

```
      alpha = 2,
      beta = c(0.5, 0.5),
      paste0("lab", k),
      paste0("prod", k)
    )
  }

  dst.consumer <- node_new(
    "util",
    type = "FUNC",
    last.a = rep(1, np),
    func = function(p) {
      value.marginal.utility <- beta.consumer / p
      ratio <- value.marginal.utility / mean(value.marginal.utility)
      a <- dst.consumer$last.a
      a <- prop.table(a * ratio_adjust(ratio, 0.15))
      dst.consumer$last.a <- a
      a
    },
    paste0("prod", 1:np)
  )

  ge <- sdm2(
    A = c(dstl.firm, dst.consumer),
    B = B,
    S0Exg = S,
    names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:np)),
    names.agent = c(paste0("firm", 1:np), "consumer"),
    numeraire = "lab1",
    ts = TRUE,
    priceAdjustmentVelocity = 0.1
  )

  ge$p
  ge$z
  ge$D
  ge$S
  growth_rate(ge$p[1:np])
  growth_rate(ge$p[(np + 1):(2 * np)])
```

`gemIntertemporal_TimeCircle_3_3`
A Time Circle Model with One Consumer and Two Types of Firms

Description

An (intertemporal) time circle model with one consumer and two types of firms.

Usage

```
gemIntertemporal_TimeCircle_3_3(...)
```

Arguments

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

Examples

```
#### an example with a Cobb-Douglas intertemporal utility function
np <- 5 # the number of periods, firms.
n <- 3 * np
m <- 2*np+1

zeta <- 1.25
S <- matrix(NA, n, m)
S[(n-np+ 1):n, m] <- 100

B <- matrix(0, n, m)
B[1:np, 1:np] <- B[(np+1):(2*np), (np+1):(2*np)] <- diag(np)[, c(2:np, 1)]
B[1, np] <-  B[np+1, 2*np]<- 1 / zeta

dstl.firm.corn <- dstl.firm.iron  <- list()
for (k in 1:np) {
  dstl.firm.corn[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 1, beta = c(0.5, 0.5),
    paste0("iron", k), paste0("lab", k)
  )

  dstl.firm.iron[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 2, beta = c(0.5, 0.5),
    paste0("iron", k), paste0("lab", k)
  )
}

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = prop.table(rep(1, np)),
  paste0("corn", 1:np)
)

ge <- sdm2(
  A = c(dstl.firm.corn, dstl.firm.iron, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("corn", 1:np),
```

```
                      paste0("iron", 1:np),
                      paste0("lab", 1:np)),
  names.agent = c(paste0("firm.corn", 1:np),
                  paste0("firm.iron", 1:np),
                  "consumer"),
  numeraire = "lab1",
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
```

---

gemIntertemporal_TimeCircle_3_4
A Time Circle Model with Two Consumers and Two Types of Firms

---

Description

An (intertemporal) time circle model with two consumers and two types of firms.

Usage

```
gemIntertemporal_TimeCircle_3_4(...)
```

Arguments

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

Examples

```
#### an example with a Cobb-Douglas intertemporal utility function
np <- 5 # the number of periods, firms.
n <- 3 * np
m <- 2 * np + 2

zeta <- 1.25
S <- matrix(NA, n, m)
S[(n - np + 1):n, (m - 1):m] <- 100

B <- matrix(0, n, m)
B[1:np, 1:np] <-
  B[(np + 1):(2 * np), (np + 1):(2 * np)] <- diag(np)[, c(2:np, 1)]
B[1, np] <-  B[np + 1, 2 * np] <- 1 / zeta
```

```
dstl.firm.corn <- dstl.firm.iron  <- list()
for (k in 1:np) {
  dstl.firm.corn[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 1,
    beta = c(0.5, 0.5),
    paste0("iron", k),
    paste0("lab", k)
  )

  dstl.firm.iron[[k]] <- node_new(
    "prod",
    type = "CD",
    alpha = 2,
    beta = c(0.5, 0.5),
    paste0("iron", k),
    paste0("lab", k)
  )
}

dst.consumer1 <- node_new(
  "util",
  type = "CD",
  alpha = 1,
  beta = prop.table(rep(1, np)),
  paste0("corn", 1:np)
)

dst.consumer2 <- node_new(
  "util",
  type = "CD",
  alpha = 1,
  beta = prop.table(rep(1, np)),
  paste0("cc", 1:np)
)
for (k in 1:np) {
  node_set(
    dst.consumer2,
    paste0("cc", k),
    type = "CD",
    alpha = 1,
    beta = c(0.5, 0.5),
    paste0("corn", k),
    paste0("iron", k)
  )
}

ge <- sdm2(
  A = c(dstl.firm.corn, dstl.firm.iron, dst.consumer1, dst.consumer2),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("corn", 1:np),
```

```
                      paste0("iron", 1:np),
                      paste0("lab", 1:np)),
  names.agent = c(
    paste0("firm.corn", 1:np),
    paste0("firm.iron", 1:np),
    "consumer1",
    "consumer2"
  ),
  numeraire = "lab1",
  ts = TRUE
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
```

`gemIntertemporal_TimeCircle_Bank_1_2`
Some Examples of a Time Circle Model with a Consumer and a Type of Bank

---

Description

Some examples of a time circle model with a consumer and a type of bank. These models can be used to solve some intertemporal savings problems.

In these example, an np-period-lived consumer gets some payoff (or cash, exhaustible resource etc.) in each period. In each period the consumer can use payoff for consumption, save payoff into bank or get a loan from the bank. The interest rate is given. The consumer has a CES intertemporal utility function and attempts to maximize intertemporal utility by saving and borrowing.

Usage

```
gemIntertemporal_TimeCircle_Bank_1_2(...)
```

Arguments

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

Examples

```
#### an example with a 5-period-lived consumer (see Zhang, 2008, section 1.3)
np <- 5 # the number of periods

interest.rate <- 0.1
```

```
zeta <- (1+interest.rate)^np
S <- matrix(NA, np, np + 1)
S[1:np, np + 1] <- 100/(1:np)

B <- matrix(0, np, np + 1)
B[1:np, 1:np] <- diag(np)[, c(2:np, 1)]
B[1, np] <- 1 / zeta

dstl.bank <- list()
for (k in 1:np) {
  dstl.bank[[k]] <- node_new(
    "output",
    type = "Leontief",
    a = 1/(1+interest.rate),
    paste0("payoff", k)
  )
}

dst.consumer <- node_new(
  "util",
  type = "CES",
  es = 0.5, alpha = 1,
  beta = prop.table(1:np),
  paste0("payoff", 1:np)
)

ge <- sdm2(
  A = c(dstl.bank, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = paste0("payoff", 1:np),
  names.agent = c(paste0("bank", 1:np), "consumer"),
  numeraire = "payoff1",
  ts = TRUE,
  policy = makePolicyMeanValue(30)
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
growth_rate(ge$p)

####
dst.consumer$es <- 0

ge <- sdm2(
  A = c(dstl.bank, dst.consumer),
  B = B,
  S0Exg = S,
  names.commodity = paste0("payoff", 1:np),
```

```
  names.agent = c(paste0("bank", 1:np), "consumer"),
  numeraire = "payoff1",
  ts = TRUE,
  policy = makePolicyMeanValue(30)
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV
growth_rate(ge$p)
```

---

gemIntertemporal_TimeCircle_Stochastic_2_2
A Time Circle Model with Uncertainty

Description

A time circle model with uncertainty. In the model, the consumer will live for two periods and has a von Neumann-Morgenstern expected utility function. There is one natural state in the first period, and two natural states in the second period.

Usage

```
gemIntertemporal_TimeCircle_Stochastic_2_2(...)
```

Arguments

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

Examples

```
zeta <- 1.25
dst.firm1 <- node_new(
  "prod2",
  type = "CD", alpha = 2,
  beta = c(0.5, 0.5),
  "lab1", "prod1"
)

dst.firm2.1 <- node_new(
  "prod3.1",
  type = "CD", alpha = 2,
  beta = c(0.5, 0.5),
  "lab2.1", "prod2.1"
)
```

```
dst.firm2.2 <- node_new(
  "prod3.2",
  type = "CD", alpha = 1,
  beta = c(0.5, 0.5),
  "lab2.2", "prod2.2"
)

dst.firm3 <- node_new(
  "prod1",
  type = "Leontief",
  a = c(1, 1)*zeta,
  "prod3.1", "prod3.2"
)

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = c(1/3, 1/3, 1/3),
  "prod1","prod2.1","prod2.2"
)

ge <- sdm2(
  A = c(dst.firm1, dst.firm2.1,dst.firm2.2,
        dst.firm3,dst.consumer),
  B = matrix(c(0, 0, 0, 1,0,
               1, 0, 0, 0,0,
               1, 0, 0, 0,0,
               0, 1, 0, 0,0,
               0, 0, 1, 0,0,
               0, 0, 0, 0,0,
               0, 0, 0, 0,0,
               0, 0, 0, 0,0),8,5,TRUE),
  S0Exg = matrix(c(NA, NA, NA, NA,NA,
                   NA, NA, NA, NA,NA,
                   NA, NA, NA, NA,NA,
                   NA, NA, NA, NA,NA,
                   NA, NA, NA, NA,NA,
                   NA, NA, NA, NA,100,
                   NA, NA, NA, NA,100,
                   NA, NA, NA, NA,100),8,5,TRUE),
  names.commodity = c("prod1", "prod2.1","prod2.2",
                      "prod3.1","prod3.2",
                      "lab1", "lab2.1","lab2.2"),
  names.agent = c("firm1", "firm2.1","firm2.2",
                  "firm3.1",
                  "consumer"),
  numeraire = "lab1",
  policy = makePolicyMeanValue(30),
  ts = TRUE
)

ge$p
```

```
ge$z
ge$D
ge$S
ge$DV
ge$SV
```

---

gemLand_Labor *Some Examples of Market Clearing Paths Involving Land and Labor*

Description

Some examples of market clearing paths involving land and labor. The labor supply may increase from the fifth period.

Usage

```
gemLand_Labor(...)
```

Arguments

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

Examples

```
#### a 3-by-3 economy
f <- function(GRLabor = 0,
              es.land.labor = 1) {
  dst.firm <- node_new("output",
                       type = "SCES",
                       es = es.land.labor, alpha = 1,
                       beta = c(0.5, 0.5),
                       "land", "lab"
  )

  dst.land.owner <- node_new(
    "util",
    type = "Leontief", a = 1,
    "prod"
  )

  dst.laborer <- Clone(dst.land.owner)

  dstl <- list(dst.firm, dst.land.owner, dst.laborer)

  ge <- sdm2(
    A = dstl,
    B = diag(c(1, 0, 0)),
    S0Exg = matrix(c(
```

```
      NA, NA, NA,
      NA, 100, NA,
      NA, NA, 100
    ), 3, 3, TRUE),
    names.commodity = c("prod", "land", "lab"),
    names.agent = c("firm", "land.owner", "laborer"),
    maxIteration = 1,
    numberOfPeriods = 30,
    numeraire = "lab",
    ts = TRUE,
    policy = list(
      function(time, state) {
        if (time >= 5) {
          state$S[3, 3] <- 100 * (1 + GRLabor)^(time - 4)
        }
        state
      },
      policyMarketClearingPrice
    ),
    z0 = c(200, 100, 100),
    p0 = c(1, 1, 1)
  )

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

  ge
}

ge <- f()
ge$p
ge$z


f(GRLabor = 0.03)
f(GRLabor = -0.03)
f(GRLabor = 0.03, es.land.labor = 0.5)
f(GRLabor = 0.03, es.land.labor = 1.5)

#### a 4-by-3 economy
GRLabor <- 0.03

dst.agri <- node_new("agri",
                     type = "SCES", es = 0.5, alpha = 1,
                     beta = c(0.75, 0.25),
                     "land", "lab"
)

dst.manu <- node_new("manu",
                     type = "SCES", es = 0.5, alpha = 1,
                     beta = c(0.25, 0.75),
                     "land", "lab"
```

```
)

dst.consumer <- node_new("util",
                         type = "SCES", es = 0.5, alpha = 1,
                         beta = c(0.5, 0.5),
                         "agri", "manu"
)

dstl <- list(dst.agri, dst.manu, dst.consumer)

ge <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0, 0,
    0, 1, 0,
    0, 0, 0,
    0, 0, 0
  ), 4, 3, TRUE),
  S0Exg = {
    S0Exg <- matrix(NA, 4, 3)
    S0Exg[3:4, 3] <- 100
    S0Exg
  },
  names.commodity = c("agri", "manu", "land", "lab"),
  names.agent = c("agri", "manu", "consumer"),
  numeraire = c("manu"),
  ts = TRUE,
  policy = list(
    function(time, state) {
      if (time >= 5) {
        state$S[4, 3] <- 100 * (1 + GRLabor)^(time - 4)
      }
      state
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 40,
  maxIteration = 1,
  z0 = c(100, 100, 200),
  p0 = c(1, 1, 1, 1)
)

matplot(ge$ts.z, type = "o", pch = 20)
matplot(growth_rate(ge$ts.z), type = "o", pch = 20)
matplot(growth_rate(ge$ts.p), type = "o", pch = 20)
```

`gemLand_Labor_Capital_4_3`
Some Examples of Market Clearing Paths Involving Land, Labor and Capital

---

Description

Some examples of market clearing paths involving land, labor and capital.

Usage

```
gemLand_Labor_Capital_4_3(...)
```

Arguments

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

Examples

```
depreciation.rate <- 0.05

dst.firm.production <- node_new("prod",
  type = "CD",
  alpha = 1, beta = c(0.4, 0.4, 0.2),
  "lab", "cap", "land"
)

dst.firm.capital.leasing <- node_new("cap",
  type = "Leontief", a = 1,
  "prod"
)

dst.consumer <- node_new("util",
  type = "CD",
  alpha = 1, beta = c(0.5, 0.5),
  "prod", "lab"
)

dstl <- list(dst.firm.production, dst.consumer, dst.firm.capital.leasing)
f <- function(policy = policyMarketClearingPrice,
              p0 = c(1, 1, 1, 1),
              z0 = c(10, 10, 10),
              numberOfPeriods = 100) {
  sdm2(
    A = dstl,
    B = matrix(c(
      1, 0, 1 - depreciation.rate,
      0, 0, 0,
      0, 0, 1,
      0, 0, 0
    ), 4, 3, TRUE),
    S0Exg = {
      S0Exg <- matrix(NA, 4, 3)
      S0Exg[2, 2] <- S0Exg[4, 2] <- 1
      S0Exg
    },
    names.commodity = c("prod", "lab", "cap", "land"),
```

```
    names.agent = c("firm.production", "consumer", "firm.capital.leasing"),
    numeraire = "prod",
    maxIteration = 1,
    numberOfPeriods = numberOfPeriods,
    p0 = p0,
    z0 = z0,
    policy = policy,
    ts = TRUE
  )
}

ge1 <- f()
ge1$p
ge1$DV
ge1$SV
matplot(ge1$ts.z, type = "l")

## a market clearing path with population growth
policy.population.growth <- function(time, state) {
  if (time >= 5) {
    state$S[2, 2] <- 1.01^(time - 4)
  }
  state
}

ge2 <- f(
  policy = list(
    policy.population.growth,
    policyMarketClearingPrice
  ),
  p0 = ge1$p, z0 = ge1$z,
  numberOfPeriods = 30
)
matplot(ge2$ts.z, type = "o", pch = 40)
matplot(growth_rate(ge2$ts.z), type = "o", pch = 20)

## a market clearing path with technology progress
policy.technology.progress <- function(time, A) {
  if (time >= 5) {
    A[[1]]$alpha <- 1.02^(time - 4)
  }
}

ge3 <- f(
  policy = list(
    policy.technology.progress,
    policyMarketClearingPrice
  ),
  p0 = ge1$p, z0 = ge1$z,
  numberOfPeriods = 30
)

matplot(ge3$ts.z, type = "o", pch = 20)
```

```
matplot(growth_rate(ge3$ts.z), type = "o", pch = 20)

## a market clearing path with population growth and technology progress
ge4 <- f(
  policy = list(
    policy.population.growth,
    policy.technology.progress,
    policyMarketClearingPrice
  ),
  p0 = ge1$p, z0 = ge1$z,
  numberOfPeriods = 30
)

matplot(ge4$ts.z, type = "o", pch = 20)
matplot(growth_rate(ge4$ts.z), type = "o", pch = 20)
```

---

gemMarketClearingPath_2_2 *Some Examples of Market Clearing Paths*

Description

Some examples of market clearing paths containing a firm and a laborer (consumer).

Usage

```
gemMarketClearingPath_2_2(...)
```

Arguments

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

Examples

```
dst.firm <- node_new(
  "prod",
  type = "CD", alpha = 5, beta = c(0.5, 0.5),
  "prod", "lab"
)

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

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

```
f <- function(policy = NULL) {
  sdm2(
    A = dstl,
    B = matrix(c(
      1, 0,
      0, 0
    ), 2, 2, TRUE),
    S0Exg = matrix(c(
      NA, NA,
      NA, 1
    ), 2, 2, TRUE),
    names.commodity = c("prod", "lab"),
    names.agent = c("firm", "consumer"),
    numeraire = "lab",
    z0 = c(1, 1),
    ts = TRUE,
    policy = policy,
    numberOfPeriods = 40,
    maxIteration = 1
  )
}

ge <- f(policy = policyMarketClearingPrice)
matplot(ge$ts.S[1, 1, ], type = "o", pch = 20)
matplot(ge$ts.z, type = "o", pch = 20)

## labor supply change
ge.LSC <- f(policy = list(
  function(time, state) {
    if (time >= 21) state$S[2, 2] <- state$S[2, 2] * 2
    state
  },
  policyMarketClearingPrice
))

matplot(ge.LSC$ts.z, type = "o", pch = 20)

## technology progress
ge.TP <- f(policy = list(
  makePolicyTechnologyChange(
    adjumentment.ratio = 2,
    agent = "firm",
    time.win = c(21, 21)
  ),
  policyMarketClearingPrice
))

matplot(ge.TP$ts.z, type = "o", pch = 20)

## the same as above
ge.TP2 <- f(policy = list(
  function(time, A) {
    if (time >= 21) {
```

```
      A[[1]]$alpha <- 10
    } else {
      A[[1]]$alpha <- 5
    }
  },
  policyMarketClearingPrice
))

matplot(ge.TP2$ts.z, type = "o", pch = 20)
```

---

gemMoney_3_2 *A General Equilibrium Model with Money*

Description

A general equilibrium model with money as a medium of exchange and a means of payment.

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

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

node_plot(dst.SCES.firm)

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

ge.SCES <- gemMoney_3_2(dstl.SCES)
ge.SCES$p
p.money <- ge.SCES$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.SCES,
             numeraire = c("money" = interest.rate)
)
```

---

gemMoney_3_3 *Some 3-by-3 General Equilibrium Models with Money*

Description

Some 3-by-3 general equilibrium models with money as a medium of exchange and a means of payment.

Usage

```
gemMoney_3_3(...)
```

Arguments

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

References

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

Examples

```
interest.rate <- 0.25

dst.firm <- node_new("prod",
                     type = "FIN", rate = c(1, interest.rate),
                     "cc1", "money"
)
node_set(dst.firm, "cc1",
         type = "CD", alpha = 2, beta = c(0.5, 0.5),
         "prod", "lab"
)

dst.laborer <- dst.money.owner <-
  node_new("util",
           type = "FIN", rate = c(1, interest.rate),
           "prod", "money"
  )

ge <- sdm2(
  A = list(dst.firm, dst.laborer, dst.money.owner),
  B = diag(c(1, 0, 0)),
```

```
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 100, NA,
    NA, NA, 100
  ), 3, 3, TRUE),
  names.commodity = c("prod", "lab", "money"),
  names.agent = c("firm", "laborer", "money.owner"),
  numeraire = "prod"
)

ge$p
ge$z
ge$D
ge$S
ge$DV
ge$SV

## Take money as numeraire, that is, let the asset price of money equal to 1,
## and let the interest per unit of money equal to the exogenous interest rate.
ge2 <- sdm2(
  A = list(dst.firm, dst.laborer, dst.money.owner),
  B = diag(c(1, 0, 0)),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 100, NA,
    NA, NA, 100
  ), 3, 3, TRUE),
  names.commodity = c("prod", "lab", "money"),
  names.agent = c("firm", "laborer", "money.owner"),
  numeraire = c(money = interest.rate)
)

ge2$p
ge2$z
ge2$D
ge2$S
ge2$DV
ge2$SV

#### another model (Li, 2019, example 7.2)
interest.rate <- 0.25
dst <- node_new("demand",
                type = "FIN", rate = c(1, interest.rate),
                "cc1", "money"
)
node_set(dst, "cc1",
         type = "CD", alpha = 1, beta = c(0.5, 0.5),
         "prod", "lab"
)

ge <- sdm2(
  A = list(dst, dst, dst),
  B = diag(c(1, 0, 0)),
```

```
    S0Exg = matrix(c(
      NA, NA, NA,
      NA, 100, NA,
      NA, NA, 100
    ), 3, 3, TRUE),
    names.commodity = c("prod", "lab", "money"),
    names.agent = c("firm", "laborer", "money.lender"),
    numeraire = c(money = interest.rate)
  )

  ge$p
  ge$z
  ge$D
  ge$S
  ge$DV
  ge$SV
```

`gemOLGFOneFirm` *Overlapping Generations Financial Models with One Firm*

Description

Some examples of overlapping generations financial models with one firm.

When there is a population growth, we will take the security-split assumption (see `gemOLGFPureExchange`).

Usage

```
gemOLGFOneFirm(...)
```

Arguments

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

References

Samuelson, P. A. (1958) An Exact Consumption-Loan Model of Interest with or without the Social Contrivance of Money. Journal of Political Economy, vol. 66(6): 467-482.

de la Croix, David and Philippe Michel (2002, ISBN: 9780521001151) A Theory of Economic Growth: Dynamics and Policy in Overlapping Generations. Cambridge University Press.

See Also

`gemOLGPureExchange_2_2` `gemOLGTimeCircle`

Examples

```
#### an OLGF economy with a firm and two-period-lived consumers
beta.firm <- c(1 / 3, 2 / 3)
# the population growth rate
GRExg <- 0.03
saving.rate <- 0.5
ratio.saving.consumption <- saving.rate / (1 - saving.rate)

dst.firm <- node_new(
  "prod",
  type = "CD", alpha = 5,
  beta = beta.firm,
  "lab", "prod"
)

dst.age1 <- node_new(
  "util",
  type = "FIN",
  rate = c(1, ratio.saving.consumption),
  "prod", "secy" # security, the financial instrument
)

dst.age2 <- node_new(
  "util",
  type = "Leontief", a = 1,
  "prod"
)

ge <- sdm2(
  A = list(
    dst.firm, dst.age1, dst.age2
  ),
  B = matrix(c(
    1, 0, 0,
    0, 0, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 1, NA,
    NA, NA, 1
  ), 3, 3, TRUE),
  names.commodity = c("prod", "lab", "secy"),
  names.agent = c("firm", "age1", "age2"),
  numeraire = "lab",
  GRExg = GRExg,
  maxIteration = 1,
  ts = TRUE
)

ge$p
```

```
matplot(ge$ts.p, type = "l")
matplot(growth_rate(ge$ts.z), type = "l") # GRExg
addmargins(ge$D, 2) # the demand matrix of the current period
addmargins(ge$S, 2) # the supply matrix of the current period
addmargins(ge$S * (1 + GRExg), 2) # the supply matrix of the next period
addmargins(ge$DV)
addmargins(ge$SV)

## Suppose consumers consume product and labor (i.e. service) and
## age1 and age2 may have different instantaneous utility functions.
dst.age1 <- node_new(
  "util",
  type = "FIN",
  rate = c(1, ratio.saving.consumption),
  "cc1", "secy" # security, the financial instrument
)
node_set(dst.age1,  "cc1",
  type = "Leontief",
  a = c(0.5, 0.5),
  "prod", "lab"
)
node_plot(dst.age1)

dst.age2 <- node_new("util",
  type = "Leontief",
  a = c(0.2, 0.8),
  "prod", "lab"
)

ge <- sdm2(
  A = list(
    dst.firm, dst.age1, dst.age2
  ),
  B = matrix(c(
    1, 0, 0,
    0, 0, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 1, NA,
    NA, NA, 1
  ), 3, 3, TRUE),
  names.commodity = c("prod", "lab", "secy"),
  names.agent = c("firm", "age1", "age2"),
  numeraire = "lab",
  GRExg = GRExg,
  priceAdjustmentVelocity = 0.05
)

ge$p
addmargins(ge$D, 2)
addmargins(ge$S, 2)
```

```
addmargins(ge$DV)
addmargins(ge$SV)

## Aggregate the above consumers into one infinite-lived consumer,
## who always spends the same amount on cc1 and cc2.
dst.consumer <- node_new("util",
                         type = "CD", alpha = 1,
                         beta = c(0.5, 0.5),
                         "cc1", "cc2"
)
node_set(dst.consumer, "cc1",
         type = "Leontief",
         a = c(0.5, 0.5),
         "prod", "lab"
)
node_set(dst.consumer,  "cc2",
         type = "Leontief",
         a = c(0.2, 0.8),
         "prod", "lab"
)

ge <- sdm2(
  A = list(
    dst.firm, dst.consumer
  ),
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 1
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "lab",
  GRExg = GRExg,
  priceAdjustmentVelocity = 0.05
)

ge$p
addmargins(ge$D, 2)
addmargins(ge$S, 2)
addmargins(ge$DV)
addmargins(ge$SV)

#### an OLGF economy with a firm and two-period-lived consumers
## Suppose each consumer has a Leontief-type utility function min(c1, c2/a).
beta.firm <- c(1 / 3, 2 / 3)
# the population growth rate, the equilibrium interest rate and profit rate
GRExg <- 0.03
rho <- 1 / (1 + GRExg)
a <- 0.9
```

```
dst.firm <- node_new(
  "prod",
  type = "CD", alpha = 5,
  beta = beta.firm,
  "lab", "prod"
)

dst.age1 <- node_new(
  "util",
  type = "FIN",
  rate = c(1, ratio.saving.consumption = a * rho),
  "prod", "secy" # security, the financial instrument
)

dst.age2 <- node_new(
  "util",
  type = "Leontief", a = 1,
  "prod"
)

ge <- sdm2(
  A = list(
    dst.firm, dst.age1, dst.age2
  ),
  B = matrix(c(
    1, 0, 0,
    0, 0, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 1, NA,
    NA, NA, 1
  ), 3, 3, TRUE),
  names.commodity = c("prod", "lab", "secy"),
  names.agent = c("firm", "age1", "age2"),
  numeraire = "lab",
  GRExg = GRExg,
  maxIteration = 1,
  ts = TRUE
)

ge$p
matplot(ge$ts.p, type = "l")
matplot(growth_rate(ge$ts.z), type = "l") # GRExg
addmargins(ge$D, 2)
addmargins(ge$S, 2)
addmargins(ge$DV)
addmargins(ge$SV)

## the corresponding time-cycle model
n <- 5 # the number of periods, consumers and firms.
```

```
S <- matrix(NA, 2 * n, 2 * n)

S.lab.consumer <- diag((1 + GRExg)^(0:(n - 1)), n)
S[(n + 1):(2 * n), (n + 1):(2 * n)] <- S.lab.consumer

B <- matrix(0, 2 * n, 2 * n)
B[1:n, 1:n] <- diag(n)[, c(2:n, 1)]
B[1, n] <- rho^n

dstl.firm <- list()
for (k in 1:n) {
  dstl.firm[[k]] <- node_new(
    "prod",
    type = "CD", alpha = 5,
    beta = beta.firm,
    paste0("lab", k), paste0("prod", k)
  )
}

dstl.consumer <- list()
for (k in 1:(n - 1)) {
  dstl.consumer[[k]] <- node_new(
    "util",
    type = "FIN",
    rate = c(1, ratio.saving.consumption = a * rho),
    paste0("prod", k), paste0("prod", k + 1)
  )
}

dstl.consumer[[n]] <- node_new(
  "util",
  type = "FIN",
  rate = c(1, ratio.saving.consumption = a * rho),
  paste0("prod", n), "cc1"
)
node_set(dstl.consumer[[n]], "cc1",
         type = "Leontief", a = rho^n, # a discounting factor
         "prod1"
)

ge2 <- sdm2(
  A = c(dstl.firm, dstl.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("prod", 1:n), paste0("lab", 1:n)),
  names.agent = c(paste0("firm", 1:n), paste0("consumer", 1:n)),
  numeraire = "lab1",
  policy = makePolicyMeanValue(30),
  maxIteration = 1,
  numberOfPeriods = 600,
  ts = TRUE
)
```

```
ge2$p
growth_rate(ge2$p[1:n]) + 1 # rho
growth_rate(ge2$p[(n + 1):(2 * n)]) + 1 # rho
ge2$D
```

---

gemOLGFPureExchange *Overlapping Generations Financial Models for Pure Exchange Economies*

Description

Some examples of overlapping generations models with financial instrument for pure exchange economies.

In these examples, there is a financial instrument (namely security) which serves as saving means and can be regarded as money, the shares of a firm, etc. Consumers use this security for saving, and this is the only use of the security. As Samuelson (1958) wrote, society by using money (i.e. security) will go from the non-optimal configuration to the optimal configuration.

Here financial demand structure trees are used, which contain financial nodes. A financial demand structure tree reflects the demand structure of a consumer who has a demand for financial instruments. Although CD-type nodes can be used instead of financial-type nodes in the consumer's demand structure tree, the use of financial-type nodes will make the demand structure tree easier to understand.

When there is a population growth, we will take the security-split assumption. That is, assume that in each period the security will be split just like share split, and the growth rate of the quantity of the security is equal to the growth rate of the population. Obviously, this assumption will not affect the calculation results essentially. And with this assumption, the equilibrium price vector can keep constant in each period, and the nominal rates of profit and interest will equal the real rates of profit and interest (i.e. the population growth rate). In contrast, in the time circle model the nominal rates of profit and interest equal zero and the real rates of profit and interest equal the population growth rate.

Usage

```
gemOLGFPureExchange(...)
```

Arguments

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

Note

As can be seen from the first example below, in a pure exchange economy with two-period-lived consumers, if age1 (i.e. young) has one unit of labor and age2 (i.e. old) does not, then the optimal allocation can be obtained by introducing securities. Here it is assumed that each consumer consumes one unit of labor in total.

However, if age2 (i.e. adult) has one unit of labor and age1 (i.e. child) does not, we cannot get the optimal allocation by introducing securities. So we need the family system.

See Also

gemOLGPureExchange_2_2

Examples

```
#### an OLGF pure exchange economy with two-period-lived consumers
## Suppose each consumer has one unit of labor in her first period
## and she has a constant saving rate (i.e. a log instantaneous utility
## function, a C-D type intertemporal utility function).
saving.rate <- 0.5
ratio.saving.consumption <- saving.rate / (1 - saving.rate)

dst.age1 <- node_new(
  "util",
  type = "FIN",
  rate = c(1, ratio.saving.consumption),
  "lab", "secy" # security, the financial instrument
)

dst.age2 <- node_new(
  "util",
  type = "Leontief", a = 1,
  "lab"
)

ge <- sdm2(
  A = list(
    dst.age1, dst.age2
  ),
  B = matrix(0, 2, 2, TRUE),
  S0Exg = matrix(c(
    1, NA,
    NA, 1
  ), 2, 2, TRUE),
  names.commodity = c("lab", "secy"),
  names.agent = c("age1", "age2"),
  numeraire = "lab"
)

ge$p
ge$D
ge$DV
ge$S

#### the basic overlapping generations (inefficient) exchange model
## Here the lab2 is regarded as a financial instrument (saving instrument).
## See gemOLGPureExchange_2_2.
dst.age1 <- node_new(
```

```
  "util",
  type = "FIN",
  rate = c(1, ratio.totalSaving.consumption = 2),
  "lab1", "lab2"
)

dst.age2 <- node_new(
  "util",
  type = "FIN",
  rate = c(1, ratio.saving.consumption = 1),
  "lab1", "lab2"
)

ge <- sdm2(
  A = list(dst.age1, dst.age2),
  B = matrix(0, 2, 2),
  S0Exg = matrix(c(
    1, 1,
    1, 0
  ), 2, 2, TRUE),
  names.commodity = c("lab1", "lab2"),
  names.agent = c("age1", "age2"),
  numeraire = "lab1",
  policy = function(time, state) {
    pension <- (state$last.A[, 2] * state$last.z[2])[2]
    if (time > 1) state$S[1, 2] <- 1 - pension
    state
  }
)

ge$p
ge$S
ge$D
ge$DV

#### the basic financial overlapping generations exchange model (see Samuelson, 1958)
## Suppose each consumer has a utility function log(c1) + log(c2) + log(c3).
GRExg <- 0.03 # the population growth rate
rho <- 1 / (1 + GRExg)

dst.age1 <- node_new(
  "util",
  type = "FIN",
  rate = {
    saving.rate <- (2 - rho) / 3
    c(1, ratio.saving.consumption = saving.rate / (1 - saving.rate))
  },
  "lab", "secy"
)

dst.age2 <- node_new(
  "util",
  type = "FIN",
```

```
  rate = c(1, ratio.saving.consumption = 1),
  "lab", "secy"
)

dst.age3 <- node_new(
  "util",
  type = "Leontief", a = 1,
  "lab"
)

ge <- sdm2(
  A = list(dst.age1, dst.age2, dst.age3),
  B = matrix(0, 2, 3),
  S0Exg = matrix(c(
    1 + GRExg, 1, 0,
    0, 0.5, 0.5
  ), 2, 3, TRUE),
  names.commodity = c("lab", "secy"),
  names.agent = c("age1", "age2", "age3"),
  numeraire = "lab",
  policy = function(time, state) {
    # Assume that unsold security will be void.
    last.Demand <- state$last.A %*% dg(state$last.z)
    secy.holding <- prop.table(last.Demand[2, ])
    if (time > 1) {
      state$S[2, 2:3] <- secy.holding[1:2]
    }
    state
  }
)

ge$p
ge$S
ge$D

#### a pure exchange economy with three-period-lived consumers
## Suppose each consumer has a Leontief-type utility function min(c1, c2, c3).
GRExg <- 0.03 # the population growth rate
R <- 1 + GRExg

dst.age1 <- node_new(
  "util",
  type = "FIN",
  rate = {
    saving.rate <- 1 / (1 + R + R^2)
    c(1, ratio.saving.consumption = saving.rate / (1 - saving.rate))
  },
  "lab", "secy"
)

dst.age2 <- node_new(
  "util",
  type = "FIN",
```

```
  rate = {
    saving.rate <- 1 / (1 + R)
    c(1, ratio.saving.consumption = saving.rate / (1 - saving.rate))
  },
  "lab", "secy"
)

dst.age3 <- node_new(
  "util",
  type = "Leontief", a = 1,
  "lab"
)

ge <- sdm2(
  A = list(dst.age1, dst.age2, dst.age3),
  B = matrix(0, 2, 3),
  S0Exg = matrix(c(
    1 + GRExg, 1, 0,
    0, 0.5, 0.5
  ), 2, 3, TRUE),
  names.commodity = c("lab", "secy"),
  names.agent = c("age1", "age2", "age3"),
  numeraire = "lab",
  policy = function(time, state) {
    # Assume that unsold security will be void.
    last.Demand <- state$last.A %*% dg(state$last.z)
    secy.holding <- prop.table(last.Demand[2, ])
    if (time > 1) {
      state$S[2, 2:3] <- secy.holding[1:2]
    }
    state
  }
)

ge$p
ge$S
ge$D

## Assume that the unsold security of age3 will be void.
## The calculation results are the same as above.
ge <- sdm2(
  A = list(dst.age1, dst.age2, dst.age3),
  B = matrix(0, 2, 3),
  S0Exg = matrix(c(
    1 + GRExg, 1, 0,
    0, 0.5, 0.5
  ), 2, 3, TRUE),
  names.commodity = c("lab", "secy"),
  names.agent = c("age1", "age2", "age3"),
  numeraire = "lab",
  policy = function(time, state, state.history) {
    secy.unsold <- state.history$S[2, , time - 1] * (1 - state.history$q[time - 1, 2])
    last.Demand <- state$last.A %*% dg(state$last.z)
```

```
    secy.purchased <- last.Demand[2, ]

    if (time > 1) {
      # Assume that the unsold security of age3 will be void.
      state$S[2, 2:3] <- prop.table(secy.purchased[1:2] + secy.unsold[1:2])
    }
    state
  },
  maxIteration = 1
)

ge$p
ge$S
ge$D
```

`gemOLGFTwoFirms` *Overlapping Generations Financial Models with Two Firms*

Description

Some examples of overlapping generations financial models with two firms.

Usage

```
gemOLGFTwoFirms(...)
```

Arguments

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

See Also

`gemOLGFPureExchange`

Examples

```
#### an example with two-period-lived consumers
dst.firm.corn <- node_new(
  "corn",
  type = "CD", alpha = 5,
  beta = c(1 / 2, 1 / 2),
  "iron", "lab"
)

dst.firm.iron <- node_new(
  "iron",
  type = "CD", alpha = 5,
```

```
  beta = c(1 / 2, 1 / 2),
  "iron", "lab"
)

dst.age1 <- node_new(
  "util",
  type = "FIN",
  rate = c(1, ratio.saving.consumption = 1),
  "corn", "secy" # security, the financial instrument
)

dst.age2 <- node_new(
  "util",
  type = "Leontief", a = 1,
  "corn"
)

ge <- sdm2(
  A = list(
    dst.firm.corn, dst.firm.iron, dst.age1, dst.age2
  ),
  B = matrix(c(
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 0, 0,
    0, 0, 0, 0
  ), 4, 4, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA, NA,
    NA, NA, NA, NA,
    NA, NA, 1, NA,
    NA, NA, NA, 1
  ), 4, 4, TRUE),
  names.commodity = c("corn", "iron", "lab", "secy"),
  names.agent = c("firm.corn", "firm.iron", "age1", "age2"),
  numeraire = "lab"
)

ge$p
ge$D
ge$DV
ge$S

## an example with three-period-lived consumers
dst.age1$rate <- c(1, ratio.saving.consumption = 1 / 2)

dst.age3 <- Clone(dst.age2)

dst.age2 <- Clone(dst.age1)
dst.age2$rate <- c(1, ratio.saving.consumption = 1)

ge <- sdm2(
  A = list(
```

```
    dst.firm.corn, dst.firm.iron, dst.age1, dst.age2, dst.age3
  ),
  B = matrix(c(
    1, 0, 0, 0, 0,
    0, 1, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0
  ), 4, 5, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA, NA, NA,
    NA, NA, NA, NA, NA,
    NA, NA, 1, 1, NA,
    NA, NA, NA, 1, 1
  ), 4, 5, TRUE),
  names.commodity = c("corn", "iron", "lab", "secy"),
  names.agent = c("firm.corn", "firm.iron", "age1", "age2", "age3"),
  numeraire = "lab",
  policy = function(time, state) {
    # Assume that unsold security will be void.
    last.Demand <- state$last.A %*% dg(state$last.z)
    secy.holding <- prop.table(last.Demand[4, ])
    if (time > 1) {
      state$S[4, 4:5] <- secy.holding[3:4]
    }
    state
  }
)

ge$p
ge$D
ge$DV
ge$S
```

`gemOLGLand_4_3` *An Overlapping Generations Model with Land*

Description

Some examples of an overlapping generations model with land.

Usage

```
gemOLGLand_4_3(...)
```

Arguments

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

Details

In this model, consumers live for two periods. Age2 owns a unit of land and age1 owns a unit of labor. Here the land use rights and land ownership are regarded as two commodities. Age2 gets land rent by selling land use rights to the firm. Then land ownership is sold to age1. Age1 saves by purchasing land ownership.

Here, the ratio of land rent to wage (denoted as xi) is determined by the production function. No matter what the saving rate of age1 is, and at what price the land ownership is transferred to age1, the consumption ratio of age2 and age1 will not be less than xi.

When the consumer's intertemporal utility function is min(c1, c2), the intertemporal substitution elasticity is 0, and each consumer desires the same amount of consumption in the two periods. It can also be assumed that the consumer's intertemporal substitution elasticity is close to 0 rather than exactly zero.

If xi>1, then this economy will inevitably have efficiency loss (dynamic inefficiency). In this case, in order to achieve the optimal allocation, not only age2 should give land to age1 for free, but age2 also needs to distribute part of the land rent to age1. However, age2 has no incentive to do so.

References

Rhee, Changyong (1991) Dynamic Inefficiency in an Economy with Land. Review of Economic Studies. 58(4), pp:791-797.

See Also

gemOLGPureExchange_2_2

Examples

```
saving.rate <- 0.001
ratio.saving.consumption <- saving.rate / (1 - saving.rate)

dst.firm <- node_new(
  "prod",
  type = "CD", alpha = 5,
  beta = c(1 / 6, 2 / 6, 3 / 6),
  "lab", "prod", "land.use.rights"
)

dst.age1 <- node_new(
  "util",
  type = "FIN",
  rate = c(1, ratio.saving.consumption),
  "prod", "land.ownership"
)

dst.age2 <- node_new(
  "util",
  type = "Leontief", a = 1,
  "prod"
)
```

```
ge <- sdm2(
  A = list(
    dst.firm, dst.age1, dst.age2
  ),
  B = matrix(c(
    1, 0, 0,
    0, 0, 0,
    0, 0, 0,
    0, 0, 0
  ), 4, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 1, NA,
    NA, NA, 1,
    NA, NA, 1
  ), 4, 3, TRUE),
  names.commodity = c("prod", "lab", "land.use.rights", "land.ownership"),
  names.agent = c("firm", "age1", "age2"),
  numeraire = "lab"
)

ge$p
ge$D
ge$DV
ge$S
ge$SV

## Change the saving-consumption ratio.
dst.age1$rate <- c(1, ratio.saving.consumption = 99)

ge <- sdm2(
  A = list(
    dst.firm, dst.age1, dst.age2
  ),
  B = matrix(c(
    1, 0, 0,
    0, 0, 0,
    0, 0, 0,
    0, 0, 0
  ), 4, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 1, NA,
    NA, NA, 1,
    NA, NA, 1
  ), 4, 3, TRUE),
  names.commodity = c("prod", "lab", "land.use.rights", "land.ownership"),
  names.agent = c("firm", "age1", "age2"),
  numeraire = "lab"
)

ge$p
```

```
ge$D
ge$DV
ge$S
ge$SV
```

---

gemOLGPrivateFirm *Overlapping Generations Models with Private Firm*

Description

Some examples of overlapping generations models with private firm. A public (i.e. publicly held) firm exists permanently and operates independently. If a public firm ownership transfers between generations, this transfer will be done through the exchange of shares. In contrast, here a private firm is established by a consumer and only runs before she retires.

In the first example, there are some two-period-lived consumers and a private firm. Suppose age1 has a unit of labor and age2 has not. In each period age1 establishes a private firm and the firm gets some labor as investment from age1 and will sell it in the market to buy some inputs for production. In the next period, age2 (i.e. age1 in the previous period) gets the output of the firm. Age2 consumes product and labor (i.e. service). Hence age1 and the firm can sell labor to age2 and buy product from age2.

In the second example with three-period-lived consumers, there are two private firms (i.e. firm1 and firm2). In each period age1 establishes a new firm1 and age2 establishes a new firm2. Firm1 gets some labor as investment from age1 and firm2 gets some product and labor as investment from age2. The output of firm1 belongs to age2 in the next period and the output of firm2 belongs to age3 in the next period. In each period age2 (i.e. age1 in the previous period) takes away the output of firm1 and age3 (i.e. age2 in the previous period) takes away the output of firm2.

Usage

```
gemOLGPrivateFirm(...)
```

Arguments

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

References

Acemoglu, D. (2009, ISBN: 9780691132921) Introduction to Modern Economic Growth. Princeton University Press.

See Also

gemOLGFPureExchange

Examples

```
#### an example with a private firm and two-period-lived consumers
saving.rate <- 0.5
beta.consumer <-  c(1 / 2, 1 / 2) # c(9 / 10, 1 / 10)

dst.firm <- node_new(
  "prod",
  type = "CD", alpha = 5,
  beta = c(2 / 3, 1 / 3),
  "prod", "lab"
)

dst.age1 <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = beta.consumer,
  "prod", "lab"
)

dst.age2 <- Clone(dst.age1)

ge <- sdm2(
  A = list(
    dst.firm, dst.age1, dst.age2
  ),
  B = matrix(c(
    1, 0, 0,
    0, 0, 0
  ), 2, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, 1, NA
  ), 2, 3, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "age1", "age2"),
  numeraire = "lab",
  policy = function(time, state) {
    if (time > 1) {
      supply.prod <- state$S[1, 1]
      supply.lab <- state$S[2, 2]
      state$S <- 0 * state$S
      state$S[1, 3] <- supply.prod # age2 supplies prod.
      state$S[2, 1] <- saving.rate * supply.lab # The firm gets investment from age1.
      state$S[2, 2] <- (1 - saving.rate) * supply.lab
    }
    state
  }
)

ge$p
addmargins(ge$D, 2)
```

```
addmargins(ge$S, 2)
addmargins(ge$DV)
addmargins(ge$SV)

#### an example with two private firm and three-period-lived consumers
saving.rate.age1 <- 1 / 3
saving.rate.age2 <- 0.95

dst.firm1 <- dst.firm2 <- node_new(
  "prod",
  type = "CD", alpha = 5,
  beta = c(2 / 3, 1 / 3),
  "prod", "lab"
)

dst.age1 <- dst.age2 <- dst.age3 <- node_new(
  "util",
  type = "Leontief", a = 1,
  "prod"
)

ge <- sdm2(
  A = list(
    dst.firm1, dst.firm2, dst.age1, dst.age2, dst.age3
  ),
  B = matrix(c(
    1, 1, 0, 0, 0,
    0, 0, 0, 0, 0
  ), 2, 5, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA, NA, NA,
    NA, NA, 1, 1, NA
  ), 2, 5, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm1", "firm2", "age1", "age2", "age3"),
  numeraire = "lab",
  policy = function(time, state) {
    if (time > 1) {
      state$S[1, 5] <- state$S[1, 2] # Age3 takes away the output of firm2.
      state$S[1, 2] <- 0

      # Age2 takes away the output of firm1.
      prod.age2 <- state$S[1, 1]
      state$S[1, 1] <- 0
      # Age2 establishes a new firm2.
      lab.age2 <- state$S[2, 4]
      state$S[2, 2] <- lab.age2 * saving.rate.age2 # Firm2 sells labor.
      state$S[1, 2] <- prod.age2 * saving.rate.age2 # Firm2 sells product.

      state$S[2, 4] <- lab.age2 * (1 - saving.rate.age2) # Age2 sells labor.
      state$S[1, 4] <- prod.age2 * (1 - saving.rate.age2) # Age2 sells product.

      # Age1 establishes a new firm1.
```

```
      state$S[2, 1] <- state$S[2, 3] * saving.rate.age1 # Firm1 sells labor.
      state$S[2, 3] <- state$S[2, 3] * (1 - saving.rate.age1) # Age1 sells labor.
    }
    state
  }
)

ge$p
ge$z
addmargins(ge$D, 2)
addmargins(ge$S, 2)
addmargins(ge$DV)
addmargins(ge$SV)
```

`gemOLGPureExchange_2_2` *The Basic Overlapping Generations Pure Exchange Model (see Samuelson, 1958)*

Description

This is the basic overlapping generations pure exchange model.

Usage

```
gemOLGPureExchange_2_2(...)
```

Arguments

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

Details

As Samuelson (1958) wrote, break each life up into thirds. Agents get one unit of payoff in period 1 and one unit in period 2; in period 3 they retire and get nothing. Suppose there are three agents in each period, namely age1, age2 and age3. In the next period, the present age1 will become age2, the present age2 will become age3, the present age3 will disappear and a new age1 will appear. Let c1, c2 and c3 denote the consumption of an agent in each period. Suppose the utility function is (c1 * c2 * c3)^(1 / 3), which is actually the same as log(c1) + log(c2) + log(c3). In each period, age1 and age2 will exchange their payoffs of the present period and the next period. Age2 will sell some present payoff and buy some future payoff as pension, and age1 quite the contrary. Age3 just takes away the pension and need not take part in the exchange. Hence there are only two agents in the pure exchange economy. In the present exchange process, the utility function of age1 is c1^(1 / 3) * x2^(2 / 3), wherein x2 is the revenue of the next period, and the utility function of age2 is c2^(1 / 2) * c3^(1 / 2).

Note

We can also suppose only age2 gets payoff and age1 does not.

References

Samuelson, P. A. (1958) An Exact Consumption-Loan Model of Interest with or without the Social Contrivance of Money. Journal of Political Economy, vol. 66(6): 467-482.

See Also

gemOLGTimeCircle

Examples

```
#### the basic overlapping generations (inefficient) exchange model
dst.age1 <- node_new(
  "util",
  type = "CD", alpha = 1, beta = c(1 / 3, 2 / 3),
  "payoff1", "payoff2"
)

dst.age2 <- node_new(
  "util",
  type = "CD", alpha = 1, beta = c(1 / 2, 1 / 2),
  "payoff1", "payoff2"
)

ge <- sdm2(
  A = list(dst.age1, dst.age2),
  B = matrix(0, 2, 2),
  S0Exg = matrix(c(
    1, 1,
    1, 0
  ), 2, 2, TRUE),
  names.commodity = c("payoff1", "payoff2"),
  names.agent = c("age1", "age2"),
  numeraire = "payoff1",
  policy = function(time, state) {
    pension <- (state$last.A[, 2] * state$last.z[2])[2]
    if (time > 1) state$S[1, 2] <- 1 - pension
    state
  }
)

ge$p # c(1, 3 / 2 + sqrt(13) / 2)
ge$S
ge$D
ge$DV

#### another calculation method for the first economy
n <- 18 # the number of agents. The number of payoff types is n+2.
```

```
payoff.age2 <- 1
payoff.age3 <- 1e-10
S <- diag(1, n + 2, n)
S <- S + rbind(0, diag(payoff.age2, n + 1, n)) + rbind(0, 0, diag(payoff.age3, n, n))

dstl <- list()
for (k in 1:n) {
  dstl[[k]] <- node_new(
    "util",
    type = "CD", alpha = 1,
    beta = c(1 / 3, 1 / 3, 1 / 3),
    paste0("payoff", k), paste0("payoff", k + 1), paste0("payoff", k + 2)
  )
}

ge <- sdm2(
  A = dstl,
  B = matrix(0, n + 2, n, TRUE),
  S0Exg = S,
  names.commodity = paste0("payoff", 1:(n + 2)),
  numeraire = "payoff1"
)

growth_rate(ge$p) + 1    # 3 / 2 + sqrt(13) / 2
ge$D
```

---

gemOLGPureExchange_Bank   *Overlapping Generations Pure Exchange Models with Bank*

Description

Some examples of overlapping generations pure exchange models with bank. Under a pay-as-you-go system, banks may only redistribute payoffs among consumers in each period. This is, in each period a bank can get a part of payoff of age1 and pay it to age2 immediately. From the consumer's point of view, the bank converts the current period's payoff into the next period's payoff. Each consumer only transacts with the bank, and she can assume that there are no other consumers.

Usage

```
gemOLGPureExchange_Bank(...)
```

Arguments

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

See Also

gemOLGPureExchange_2_2

Examples

```
#### an example with a two-period-lived consumer
dst.bank <- node_new(
  "payoff2",
  type = "Leontief", a = 1,
  "payoff1"
)

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1, beta = c(1 / 2, 1 / 2),
  "payoff1", "payoff2"
)

ge <- sdm2(
  A = list(dst.bank, dst.consumer),
  B = matrix(c(
    0, 0,
    1, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, 1,
    NA, NA
  ), 2, 2, TRUE),
  names.commodity = c("payoff1", "payoff2"),
  names.agent = c("bank", "consumer"),
  numeraire = "payoff1"
)

ge$p
ge$D
ge$S

#### an example with a three-period-lived consumer.
dst.bank1 <- node_new(
  "payoff2",
  type = "Leontief", a = 1,
  "payoff1"
)

dst.bank2 <- node_new(
  "payoff3",
  type = "Leontief", a = 1,
  "payoff2"
)

dst.consumer <- node_new(
```

```
    "util",
    type = "CD", alpha = 1, beta = c(1 / 3, 1 / 3, 1 / 3),
    "payoff1", "payoff2", "payoff3"
  )

  ge <- sdm2(
    A = list(dst.bank1, dst.bank2, dst.consumer),
    B = matrix(c(
      0, 0, 0,
      1, 0, 0,
      0, 1, 0
    ), 3, 3, TRUE),
    S0Exg = matrix(c(
      NA, NA, 1,
      NA, NA, 1,
      NA, NA, NA
    ), 3, 3, TRUE),
    names.commodity = c("payoff1", "payoff2", "payoff3"),
    names.agent = c("bank1", "bank2", "consumer"),
    numeraire = "payoff1"
  )

  ge$p
  ge$S
  ge$D

  ## Assume that banks can earn interest through foreign investment.
  dst.bank1$a <- 0.8
  dst.bank2$a <- 0.8
  ge <- sdm2(
    A = list(dst.bank1, dst.bank2, dst.consumer),
    B = matrix(c(
      0, 0, 0,
      1, 0, 0,
      0, 1, 0
    ), 3, 3, TRUE),
    S0Exg = matrix(c(
      NA, NA, 1,
      NA, NA, 1,
      NA, NA, NA
    ), 3, 3, TRUE),
    names.commodity = c("payoff1", "payoff2", "payoff3"),
    names.agent = c("bank1", "bank2", "consumer"),
    numeraire = "payoff1"
  )

  ge$p
  ge$S
  ge$D
  ge$DV
```

gemOLGStochasticSequential_3_3

A 3-by-3 OLG Stochastic Sequential General Equilibrium Model

Description

A 3-by-3 OLG stochastic sequential general equilibrium model. There are two-period lived consumers and a type of firm. There are three types of commodities, i.e. product, labor and security (i.e. paper asset, store of value).

In each period there are a young (i.e. age1), an old (i.e. age2) and a firm. The young supplies a unit of labor and the old supplies a unit of security.

Consumers only consume products. The young buys the security to save.

The firm inputs labor to produce the product. Productivity is random. The amount of labor required to produce 1 unit of product is equal to 0.1 or 0.2, both of which occur with equal probability.

All labor is used in production. Therefore, it can be known that the product output in each period (that is, the product supply in the next period) is equal to 10 or 5, that is to say, there are two natural states with equal probability of occurrence in each period: high supply state and low supply state. The states are denoted as a and b, respectively. There are then four cases of natural states experienced by a consumer in two periods: aa, bb, ab and ba.

Take the product as numeraire. The ratio of the security price in the latter period to the previous period of the two periods is the gross rate of return. The gross rates of returns on the savings of the young in the four cases is denoted as Raa, Rbb, Rab and Rba, respectively.

The utility function of the young is -0.5*x1^-1 - 0.25*x2^-1 - 0.25*x3^-1, where x1, x2 and x3 are the (expected) consumption in the current period and the next two natural states, respectively. It can be seen that the intertemporal substitution elasticity and the inter-natural-state substitution elasticity in the utility function are both 0.5, that is, the relative risk aversion coefficient is 2.

In each period, the young determines the saving rate based on the expected gross rates of returns on the security and the utility function.

In each period the young adopts the following anticipation method for the gross rate of return of the savings: First, determine the two types of interest rates (Raa, Rab or Rbb, Rba) that need to be predicted according to the current state. Both interest rates are then forecasted using the adaptive expectation method.

A market clearing path and a disequilibrium path will be calculated below. The market clearing path will converge to the stochastic equilibrium consisting of two equilibrium states, and the disequilibrium paths not.

Usage

```
gemOLGStochasticSequential_3_3(...)
```

Arguments

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

Examples

```
#optimal saving rate
osr <- function(Ra, Rb, es=0.5, beta=c(0.5, 0.25, 0.25)){
  sigma <- 1-1/es
  ((beta[2]*Ra^sigma/beta[1]+beta[3]*Rb^sigma/beta[1])^-es+1)^-1
}

dst.firm <- node_new(
  "prod",
  type = "Leontief", a = 0.2,
  "lab"
)

dst.age1 <-
  node_new("util",
           type = "FIN", rate= c(1, 1),
           "prod","secy",
           last.price = c(1,1,1),
           last.output = 10,
           Raa = 1, Rbb = 1, Rab = 1,  Rba = 1,
           REbb = 1, REba = 1, REab = 1, REaa = 1,
           sr.ts.1 = vector(),
           sr.ts.2 = vector()
  )

dst.age2 <-
  node_new("util",
           type = "Leontief", a = 1,
           "prod"
  )


policyStochasticTechnology <- function(time,A){
  A[[1]]$a <- sample(c(0.1, 0.2), 1)
}

policySaving <- function(time, A, state){
  output <- state$S[1,1]

  lambda <- 0.9

  if (output<8) {
    A[[2]]$REba <- lambda*tail(A[[2]]$Rba, 1)+(1-lambda)*A[[2]]$REba
    A[[2]]$REbb <- lambda*tail(A[[2]]$Rbb, 1)+(1-lambda)*A[[2]]$REbb

    saving.rate <- osr(A[[2]]$REba, A[[2]]$REbb)
    A[[2]]$sr.ts.1 <- c(A[[2]]$sr.ts.1, saving.rate)
  }

  if (output>=8) {
    A[[2]]$REaa <- lambda*tail(A[[2]]$Raa, 1)+(1-lambda)*A[[2]]$REaa
```

```
    A[[2]]$REab <- lambda*tail(A[[2]]$Rab, 1)+(1-lambda)*A[[2]]$REab

    saving.rate  <- osr(A[[2]]$REaa, A[[2]]$REab)
    A[[2]]$sr.ts.2 <- c(A[[2]]$sr.ts.2, saving.rate)
  }

  A[[2]]$rate <- c(1, saving.rate/(1-saving.rate))
}


policyRecord <- function(time, A, state){
  last.p <- A[[2]]$last.price
  p <- state$p/state$p[1]

  last.output <- A[[2]]$last.output
  output <- A[[2]]$last.output <- state$S[1,1]

  if ((last.output<8)&&(output<8)) A[[2]]$Rbb <- c(A[[2]]$Rbb, p[3]/last.p[3])
  if ((last.output<8)&&(output>=8)) A[[2]]$Rba <- c(A[[2]]$Rba, p[3]/last.p[3])
  if ((last.output>=8)&&(output<8)) A[[2]]$Rab <- c(A[[2]]$Rab, p[3]/last.p[3])
  if ((last.output>=8)&&(output>=8)) A[[2]]$Raa <- c(A[[2]]$Raa, p[3]/last.p[3])

  A[[2]]$last.price <- p
}

dstl <- list(dst.firm, dst.age1, dst.age2)

B <-  matrix(c(
  1, 0, 0,
  0, 0, 0,
  0, 0, 0
), 3, 3, TRUE)

S0Exg <- matrix(c(
  NA, NA, NA,
  NA, 1, NA,
  NA, NA, 1
), 3, 3, TRUE)

## a market clearing path
set.seed(1)
ge <- sdm2(
  A = dstl,  B = B,  S0Exg = S0Exg,
  names.commodity = c("prod","lab", "secy"),
  names.agent = c("firm","age1", "age2"),
  numeraire = "prod",
  policy = list(policyStochasticTechnology,
                policySaving,
                policyMarketClearingPrice,
                policyRecord),
  z0 = c(5,1,1),
  maxIteration = 1,
  numberOfPeriods = 40,
```

```
  ts=TRUE
)

matplot(ge$ts.z, type="o", pch=20)
matplot(ge$ts.p, type="o", pch=20)

## a disequilibrium path
set.seed(1)
de <- sdm2(
  A = dstl,  B = B,  S0Exg = S0Exg,
  names.commodity = c("prod","lab", "secy"),
  names.agent = c("firm","age1", "age2"),
  numeraire = "prod",
  policy = list(policyStochasticTechnology,
                policySaving,
                policyRecord),
  maxIteration = 1,
  numberOfPeriods = 400,
  ts=TRUE
)

matplot(de$ts.z, type="o", pch=20)
matplot(de$ts.p, type="o", pch=20)
```

`gemOLGTimeCircle` *Time-Circle Models (Closed Loop Overlapping Generations Models)*

Description

Some examples of time-circle models, which usually contain overlapping generations. When we connect together the head and tail of time of a dynamic model, we get a time-circle model which treats time as a circle of finite length instead of a straight line of infinite length. The (discounted) output of the final period will enter the utility function and the production function of the first period, which implies the influence mechanism of the past on the present is just like the influence mechanism of the present on the future. A time-circle OLG model may be called a reincarnation model.

In a time-circle model, time can be thought of as having no beginning and no end. And we can assume all souls meet in a single market (see Shell, 1971), or there are implicit financial instruments that facilitate transactions between adjacent generations.

As in the Arrow–Debreu approach, in the following examples with production, we assume that firms operate independently and are not owned by consumers. That is, there is no explicit property of firms as the firms do not make pure profit at equilibrium (constant return to scale) (see de la Croix and Michel, 2002, page 292).

Usage

```
gemOLGTimeCircle(...)
```

Arguments

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

References

de la Croix, David and Philippe Michel (2002, ISBN: 9780521001151) A Theory of Economic Growth: Dynamics and Policy in Overlapping Generations. Cambridge University Press.

Diamond, Peter (1965) National Debt in a Neoclassical Growth Model. American Economic Review. 55 (5): 1126-1150.

Samuelson, P. A. (1958) An Exact Consumption-Loan Model of Interest with or without the Social Contrivance of Money. Journal of Political Economy, vol. 66(6): 467-482.

Shell, Karl (1971) Notes on the Economics of Infinity. Journal of Political Economy. 79 (5): 1002–1011.

See Also

gemOLGPureExchange_2_2

Examples

```
#### a time-circle pure exchange economy with two-period-lived consumers
## In this exampe, each agent sells some labor to the previous generation
## and buys some labor from the next generation.

# Here np can tend to infinity.
np <- 4 # the number of periods and consumers (i.e. laborers).
lab.age1 <- 1 # labor endowment of age1
lab.age2 <- 0 # labor endowment of age2

S <- lab.age1 * diag(np) + lab.age2 * diag(np)[, c(2:np, 1)]

# Suppose each consumer has a utility function log(c1) + log(c2).
beta.consumer <- c(0.5, 0.5)
index <- c(1:np, 1)
dstl.consumer <- list()
for (k in 1:np) {
  dstl.consumer[[k]] <- node_new(
    "util",
    type = "CD", alpha = 1,
    beta = beta.consumer,
    paste0("lab", index[k]), paste0("lab", index[k + 1])
  )
}

ge <- sdm2(
  A = dstl.consumer,
  B = matrix(0, np, np, TRUE),
  S0Exg = S,
  names.commodity = paste0("lab", 1:np),
```

```
  numeraire = "lab1"
)

ge$p
ge$D
ge$S

## Introduce population growth into the above pure exchange economy.
GRExg <- 0.03 # the population growth rate

S <- S %*% diag((1 + GRExg)^(0:(np - 1)))
S[1, np] <- S[1, np] * (1 + GRExg)^-np

dstl.consumer[[np]] <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = beta.consumer,
  paste0("lab", np), "cc1"
)
node_set(dstl.consumer[[np]],
         "cc1",
         type = "Leontief", a = (1 + GRExg)^(-np), # a discounting factor
         "lab1"
)

ge <- sdm2(
  A = dstl.consumer,
  B = matrix(0, np, np, TRUE),
  S0Exg = S,
  names.commodity = paste0("lab", 1:np),
  numeraire = "lab1"
)

ge$p
ge$D
ge$DV
ge$S

#### a time-circle pure exchange economy with three-period-lived consumers
## Suppose each consumer has a utility function log(c1) + log(c2) + log(c3).
## See gemOLGPureExchange_2_2.
np <- 5 # the number of periods and consumers (i.e. laborers).
lab.age1 <- 1 # labor endowment of age1
lab.age2 <- 1
lab.age3 <- 0
S <- lab.age1 * diag(np) + lab.age2 * diag(np)[, c(2:np, 1)] + lab.age3 * diag(np)[, c(3:np, 1:2)]

index <- c(1:np, 1, 2)
dstl.consumer <- list()
beta.consumer <- c(1 / 3, 1 / 3, 1 / 3)
for (k in 1:np) {
  dstl.consumer[[k]] <- node_new(
    "util",
```

```
    type = "CD", alpha = 1,
    beta = beta.consumer,
    paste0("lab", index[k]), paste0("lab", index[k + 1]), paste0("lab", index[k + 2])
  )
}

ge <- sdm2(
  A = dstl.consumer,
  B = matrix(0, np, np, TRUE),
  S0Exg = S,
  names.commodity = paste0("lab", 1:np),
  numeraire = "lab1"
)

ge$p
ge$D

## Introduce population growth into the above pure exchange economy.
GRExg <- 0.03 # the population growth rate
S <- S %*% diag((1 + GRExg)^(0:(np - 1)))
S[1, np - 1] <- S[1, np - 1] * (1 + GRExg)^-np
S[1:2, np] <- S[1:2, np] * (1 + GRExg)^-np

dstl.consumer[[np - 1]] <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = beta.consumer,
  paste0("lab", np - 1), paste0("lab", np), "cc1"
)
node_set(dstl.consumer[[np - 1]], "cc1",
         type = "Leontief", a = (1 + GRExg)^(-np), # a discounting factor
         "lab1"
)

dstl.consumer[[np]] <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = beta.consumer,
  paste0("lab", np), "cc1", "cc2"
)
node_set(dstl.consumer[[np]], "cc1",
         type = "Leontief", a = (1 + GRExg)^(-np), # a discounting factor
         "lab1"
)
node_set(dstl.consumer[[np]], "cc2",
         type = "Leontief", a = (1 + GRExg)^(-np), # a discounting factor
         "lab2"
)

ge <- sdm2(
  A = dstl.consumer,
  B = matrix(0, np, np, TRUE),
  S0Exg = S,
```

```
  names.commodity = paste0("lab", 1:np),
  numeraire = "lab1"
)

ge$p
growth_rate(ge$p) + 1   # 1 / (1 + GRExg)
ge$D
ge$DV

## Suppose each consumer has a Leontief-type utility function min(c1, c2, c3).
discount.factor <- c(1, 1 / (1 + GRExg), 1 / (1 + GRExg)^2)
beta.consumer <- prop.table(discount.factor)

lapply(dstl.consumer, function(dst) dst$beta <- beta.consumer)

ge <- sdm2(
  A = dstl.consumer,
  B = matrix(0, np, np, TRUE),
  S0Exg = S,
  names.commodity = paste0("lab", 1:np),
  numeraire = "lab1"
)

ge$p
growth_rate(ge$p) + 1 # 1 / (1 + GRExg)
ge$D
ge$DV

#### a time-circle model with production and two-period-lived consumers
## Suppose each consumer has a utility function log(c1) + log(c2).
np <- 5 # the number of periods, consumers and firms.
S <- matrix(NA, 2 * np, 2 * np)

S.lab.consumer <- diag(1, np)
S[(np + 1):(2 * np), (np + 1):(2 * np)] <- S.lab.consumer

B <- matrix(0, 2 * np, 2 * np)
B[1:np, 1:np] <- diag(np)[, c(2:np, 1)]

dstl.firm <- list()
beta.firm <- c(1 / 3, 2 / 3)
for (k in 1:np) {
  dstl.firm[[k]] <- node_new(
    "prod",
    type = "CD", alpha = 5,
    beta = beta.firm,
    paste0("lab", k), paste0("prod", k)
  )
}

index <- c(1:np, 1)
beta.consumer <- c(1 / 2, 1 / 2)
dstl.consumer <- list()
```

```
for (k in 1:np) {
  dstl.consumer[[k]] <- node_new(
    "util",
    type = "CD", alpha = 1,
    beta = beta.consumer,
    paste0("prod", index[k]), paste0("prod", index[k + 1])
  )
}

ge <- sdm2(
  A = c(dstl.firm, dstl.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:np)),
  names.agent = c(paste0("firm", 1:np), paste0("consumer", 1:np)),
  numeraire = "lab1"
)

ge$p
ge$D

## Introduce population growth into the above economy.
GRExg <- 0.03 # the population growth rate

S[(np + 1):(2 * np), (np + 1):(2 * np)] <- diag((1 + GRExg)^(0:(np - 1)), np)
B[1,np] <- (1 + GRExg)^(-np)

dstl.consumer[[np]] <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = beta.consumer,
  paste0("prod", np),"cc1"
)
node_set(dstl.consumer[[np]], "cc1",
         type = "Leontief", a = (1 + GRExg)^(-np), # a discounting factor
         "prod1"
)

ge <- sdm2(
  A = c(dstl.firm, dstl.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:np)),
  names.agent = c(paste0("firm", 1:np), paste0("consumer", 1:np)),
  numeraire = "lab1",
  policy = makePolicyMeanValue(30),
  maxIteration = 1,
  numberOfPeriods = 600,
  ts = TRUE
)

ge$p
growth_rate(ge$p[1:np]) + 1   # 1 / (1 + GRExg)
```

```
growth_rate(ge$p[(np + 1):(2 * np)]) + 1   # 1 / (1 + GREXg)
ge$D
ge$DV

#### a time-circle model with production and one-period-lived consumers
## These consumers also can be regarded as infinite-lived carpe diem agents,
## who have endowments in each period, maximize the instantaneous utility and do not
## consider the future.
np <- 5 # the number of periods, consumers and firms.
GRExg <- 0.03 # the population growth rate
discount.factor.last.output <- (1 + GRExg)^(-np)
S <- matrix(NA, 2 * np, 2 * np)

S.lab.consumer <- diag((1 + GRExg)^(0:(np - 1)), np)
S[(np + 1):(2 * np), (np + 1):(2 * np)] <- S.lab.consumer

B <- matrix(0, 2 * np, 2 * np)
B[1:np, 1:np] <- diag(np)[, c(2:np, 1)]
B[1, np] <- discount.factor.last.output

beta.firm <- c(1 / 3, 2 / 3)
beta.consumer <- c(1 / 2, 1 / 2)

dstl.firm <- list()
for (k in 1:np) {
  dstl.firm[[k]] <- node_new(
    "prod",
    type = "CD", alpha = 5,
    beta = beta.firm,
    paste0("lab", k), paste0("prod", k)
  )
}

dstl.consumer <- list()
for (k in 1:np) {
  dstl.consumer[[k]] <- node_new(
    "util",
    type = "CD", alpha = 1,
    beta = beta.consumer,
    paste0("lab", k), paste0("prod", k)
  )
}

ge <- sdm2(
  A = c(dstl.firm, dstl.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:np)),
  names.agent = c(paste0("firm", 1:np), paste0("consumer", 1:np)),
  numeraire = "lab1",
  policy = makePolicyMeanValue(30),
  maxIteration = 1,
  numberOfPeriods = 600,
```

```
  ts = TRUE
)

ge$p
growth_rate(ge$p[1:np]) + 1   # 1 / (1 + GRExg)
growth_rate(ge$p[(np + 1):(2 * np)]) + 1   # 1 / (1 + GRExg)
ge$D

## the reduced form of the above model
dst.firm <- node_new(
  "prod",
  type = "CD", alpha = 5,
  beta = beta.firm,
  "lab", "prod"
)

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = beta.consumer,
  "lab", "prod"
)

ge2 <- sdm2(
  A = list(
    dst.firm,
    dst.consumer
  ),
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 1
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "lab",
  GRExg = 0.03
)

ge2$p
ge2$D

#### a time-circle model with production and three-period-lived consumers
## Suppose each consumer has a utility function log(c1) + log(c2) + log(c3).
np <- 6 # the number of periods, consumers and firms.
GRExg <- 0.03 # the population growth rate
S <- matrix(NA, 2 * np, 2 * np)

S.lab.consumer <- rbind(diag((1 + GRExg)^(0:(np - 1)), np), 0) +
  rbind(0, diag((1 + GRExg)^(0:(np - 1)), np))
```

```
S.lab.consumer <- S.lab.consumer[1:np, ]
S.lab.consumer[1, np] <- (1 + GRExg)^-1

S[(np + 1):(2 * np), (np + 1):(2 * np)] <- S.lab.consumer

B <- matrix(0, 2 * np, 2 * np)
B[1:np, 1:np] <- diag(np)[, c(2:np, 1)]
B[1, np] <- (1 + GRExg)^(-np)

beta.firm <- c(2 / 5, 3 / 5)
beta.consumer <- c(1 / 3, 1 / 3, 1 / 3)

dstl.firm <- list()
for (k in 1:np) {
  dstl.firm[[k]] <- node_new(
    "prod",
    type = "CD", alpha = 5,
    beta = beta.firm,
    paste0("lab", k), paste0("prod", k)
  )
}

dstl.consumer <- list()
for (k in 1:(np - 2)) {
  dstl.consumer[[k]] <- node_new(
    "util",
    type = "CD", alpha = 1,
    beta = beta.consumer,
    paste0("prod", k), paste0("prod", k + 1), paste0("prod", k + 2)
  )
}

dstl.consumer[[np - 1]] <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = beta.consumer,
  paste0("prod", np - 1), paste0("prod", np), "cc1"
)
node_set(dstl.consumer[[np - 1]], "cc1",
         type = "Leontief", a = (1 + GRExg)^(-np), # a discounting factor
         "prod1"
)

dstl.consumer[[np]] <- node_new(
  "util",
  type = "CD", alpha = 1,
  beta = beta.consumer,
  paste0("prod", np), "cc1", "cc2"
)
node_set(dstl.consumer[[np]], "cc1",
         type = "Leontief", a = (1 + GRExg)^(-np), # a discounting factor
         "prod1"
)
```

```
node_set(dstl.consumer[[np]], "cc2",
         type = "Leontief", a = (1 + GRExg)^(-np), # a discounting factor
         "prod2"
)

ge <- sdm2(
  A = c(dstl.firm, dstl.consumer),
  B = B,
  S0Exg = S,
  names.commodity = c(paste0("prod", 1:np), paste0("lab", 1:np)),
  names.agent = c(paste0("firm", 1:np), paste0("consumer", 1:np)),
  numeraire = "lab1",
  policy = makePolicyMeanValue(30),
  maxIteration = 1,
  numberOfPeriods = 600,
  ts = TRUE
)

ge$p
growth_rate(ge$p[1:np]) + 1   # 1 / (1 + GRExg)
growth_rate(ge$p[(np + 1):(2 * np)]) + 1   # 1 / (1 + GRExg)
ge$D
ge$DV
```

---

gemOpenEconomy_4_4 *A 4-by-4 Open Economy with Bond*

Description

Some examples of a 4-by-4 open economy with bond.

Usage

```
gemOpenEconomy_4_4(...)
```

Arguments

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

Examples

```
#### an open economy with foreign bond (bond.ROW)
dst.firm <- node_new(
  "output",
  type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = 1,
  "prod.CHN", "lab"
)
```

```
dst.consumer <- node_new(
  "util",
  type = "FIN", beta = c(0.8, 0.2),
  "cc1", "bond.ROW"
)
node_set(dst.consumer, "cc1",
         type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = 1,
         "prod.CHN", "prod.ROW"
)

dst.FT <- node_new(
  "prod.ROW",
  type = "SCES", alpha = 1, beta = c(2/3, 1/3), es = 1,
  "prod.CHN", "lab"
)

dst.Bond <- node_new(
  "util",
  type = "SCES", alpha = 1, beta = c(2/3, 1/3), es = 1,
  "prod.CHN", "lab"
)

ge.open <- sdm2(
  A = list(dst.firm, dst.consumer, dst.FT, dst.Bond),
  B = matrix(c(
    1, 0, 0, 0,
    0, 0, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 0
  ), 4, 4, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA, NA,
    NA, 300, NA, NA,
    NA, NA, NA, NA,
    NA, NA, NA, 60
  ), 4, 4, TRUE),
  names.commodity = c("prod.CHN", "lab", "prod.ROW", "bond.ROW"),
  names.agent = c("firm", "consumer", "FT", "Bond"),
  numeraire = "lab"
)

ge.open$D
ge.open$p

## a corresponding two-country model
dst.firm <- node_new(
  "output",
  type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = 1,
  "prod", "lab"
)

dst.consumer <- node_new(
```

```
  "util",
  type = "FIN", beta = c(0.8, 0.2),
  "cc1", "bond.ROW"
)
node_set(dst.consumer, "cc1",
         type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = 1,
         "prod", "prod.ROW"
)

dst.firm.ROW <- node_new(
  "prod.ROW",
  type = "SCES", alpha = 1, beta = c(0.25, 0.25, 0.5), es = 1,
  "prod", "prod.ROW", "lab.ROW"
)

dst.consumer.ROW <- node_new(
  "util",
  type = "SCES", alpha = 1, beta = c(0.25, 0.25, 0.25, 0.25), es = 1,
  "prod", "prod.ROW", "lab", "lab.ROW"
)

ge <- sdm2(
  A = list(dst.firm, dst.consumer, dst.firm.ROW, dst.consumer.ROW),
  B = matrix(c(
    1, 0, 0, 0,
    0, 0, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 0,
    0, 0, 0, 0
  ), 5, 4, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA, NA,
    NA, 300, NA, NA,
    NA, NA, NA, NA,
    NA, NA, NA, 180,
    NA, NA, NA, 60
  ), 5, 4, TRUE),
  names.commodity = c("prod", "lab", "prod.ROW", "lab.ROW", "bond.ROW"),
  names.agent = c("firm", "consumer", "firm.ROW", "consumer.ROW"),
  numeraire = "lab"
)

ge$D
ge$p

#### an open economy with domestic bond (bond.CHN)
dst.firm <- node_new(
  "output",
  type = "CD", alpha = 5, beta = c(0.5, 0.5),
  "prod.CHN", "lab"
)

dst.consumer <- node_new(
```

```
    "util",
    type = "CD", alpha = 1, beta = c(0.5, 0.5),
    "prod.CHN", "prod.ROW"
  )

  dst.FT <- node_new(
    "prod.ROW",
    type = "Leontief", a = 2,
    "prod.CHN"
  )

  dst.Bond <- node_new(
    "prod.ROW",
    type = "Leontief", a = 1,
    "bond.CHN"
  )

  ge <- sdm2(
    A = list(dst.firm, dst.consumer, dst.FT, dst.Bond),
    B = matrix(c(
      1, 0, 0, 0,
      0, 0, 0, 0,
      0, 0, 1, 1,
      0, 0, 0, 0
    ), 4, 4, TRUE),
    S0Exg = matrix(c(
      NA, NA, NA, NA,
      NA, 1, NA, NA,
      NA, NA, NA, NA,
      NA, 0.2, NA, NA
    ), 4, 4, TRUE),
    names.commodity = c("prod.CHN", "lab", "prod.ROW", "bond.CHN"),
    names.agent = c("firm", "consumer", "FT", "Bond"),
    numeraire = "lab"
  )

  ge$D
  ge$p
```

---

gemOpenEconomy_6_6 *A 6-by-6 Open Economy with Bond*

Description

Some examples of a 6-by-6 open economy with bond.

Usage

```
gemOpenEconomy_6_6(...)
```

Arguments

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

Examples

```
#### an open economy with foreign bond (bond.ROW)
dst.firm1 <- node_new(
  "prod1.CHN",
  type = "SCES", es=1, alpha = 1, beta = c(0.4, 0.4, 0.2),
  "prod2.CHN", "prod2.ROW", "lab"
)

dst.firm2 <- node_new(
  "prod2.CHN",
  type = "SCES", es=1, alpha = 1, beta = c(0.4, 0.4, 0.2),
  "prod2.CHN", "prod2.ROW", "lab"
)

dst.consumer <- node_new(
  "util",
  type = "FIN", beta = c(0.8, 0.2),
  "cc1", "bond.ROW"
)
node_set(dst.consumer, "cc1",
         type = "SCES", es=1, alpha = 1, beta = c(0.5, 0.5),
         "prod1.CHN", "prod1.ROW"
)

dst.FT1 <- node_new(
  "prod1.ROW",
  type = "SCES", es=1, alpha = 1, beta = c(0.5, 0.5),
  "prod1.CHN", "prod2.CHN"
)

dst.FT2 <- node_new(
  "prod2.ROW",
  type = "SCES", es=1, alpha = 1, beta = c(0.5, 0.5),
  "prod1.CHN", "prod2.CHN"
)

dst.Bond <- node_new(
  "util",
  type = "SCES", es=1, alpha = 1, beta = c(0.5, 0.5),
  "prod1.CHN", "prod2.CHN"
)

ge <- sdm2(
  A = list(dst.firm1, dst.firm2, dst.consumer, dst.FT1, dst.FT2, dst.Bond),
  B = matrix(c(
    1, 0, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 0,
```

```
    0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, 0, 0,
    0, 0, 0, 0, 1, 0,
    0, 0, 0, 0, 0, 0
  ), 6, 6, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA, NA, NA, NA,
    NA, NA, NA, NA, NA, NA,
    NA, NA, 100, NA, NA, NA,
    NA, NA, NA, NA, NA, NA,
    NA, NA, NA, NA, NA, NA,
    NA, NA, NA, NA, NA, 20
  ), 6, 6, TRUE),
  names.commodity = c(
    "prod1.CHN", "prod2.CHN", "lab",
    "prod1.ROW", "prod2.ROW", "bond.ROW"
  ),
  names.agent = c("firm1", "firm2", "consumer", "FT1", "FT2", "Bond"),
  numeraire = "lab"
)

ge$D
ge$p
ge$z

## Suppose the domestic consumer owns some foreign product by borrowing.
ge <- sdm2(
  A = list(dst.firm1, dst.firm2, dst.consumer, dst.FT1, dst.FT2, dst.Bond),
  B = matrix(c(
    1, 0, 0, 0, 0, 0,
    0, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, 0, 0,
    0, 0, 0, 0, 1, 0,
    0, 0, 0, 0, 0, 0
  ), 6, 6, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA, NA, NA, NA,
    NA, NA, NA, NA, NA, NA,
    NA, NA, 100, NA, NA, NA,
    NA, NA, 10, NA, NA, NA,
    NA, NA, NA, NA, NA, NA,
    NA, NA, NA, NA, NA, 20
  ), 6, 6, TRUE),
  names.commodity = c(
    "prod1.CHN", "prod2.CHN", "lab",
    "prod1.ROW", "prod2.ROW", "bond.ROW"
  ),
  names.agent = c("firm1", "firm2", "consumer", "FT1", "FT2", "Bond"),
  numeraire = "lab"
)

ge$D
```

```
ge$p
ge$z
```

gemPersistentTechnologicalProgress

Some Examples of Market Clearing Paths with Persistent Technological Progress

Description

Some examples of market clearing paths with persistent technological progress. From the fifth period, technological progress occurs.

Usage

```
gemPersistentTechnologicalProgress(...)
```

Arguments

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

See Also

gemCapitalAccumulation

Examples

```
#### a 2-by-2 example with labor-saving technological progress
tpr <- 0.03 # technological progress rate

dst.firm <- node_new(
  "prod",
  type = "SCES",
  es = 0.5, alpha = 1,
  beta = c(0.5, 0.5),
  "prod", "cc1"
)
node_set(dst.firm, "cc1",
         type = "Leontief", a = 1,
         "lab"
)

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

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

ge <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "prod",
  ts = TRUE,
  policy = list(
    function(time, A) {
      if (time >= 5) {
        node_set(A[[1]], "cc1",
                 a = (1 + tpr)^-(time - 4)
        )
      }
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 40,
  maxIteration = 1,
  z0 = c(200, 100),
  p0 = c(1, 1)
)

matplot(growth_rate(ge$ts.z), type = "o", pch = 20)
matplot(growth_rate(ge$ts.p), type = "o", pch = 20)

#### a 3-by-3 example with labor-saving technological progress
tpr <- 0.03 # technological progress rate

dst.manu <- node_new("manu",
                     type = "SCES", es = 0.5, alpha = 1,
                     beta = c(0.6, 0.4),
                     "manu", "cc1"
)
node_set(dst.manu, "cc1",
         type = "Leontief", a = 1,
         "lab"
)

dst.serv <- node_new("serv",
                     type = "SCES", es = 0.5, alpha = 1,
                     beta = c(0.4, 0.6),
                     "manu", "lab"
```

```
)

dst.consumer <- node_new("util",
                         type = "SCES", es = 0.5, alpha = 1,
                         beta = c(0.4, 0.6),
                         "manu", "serv"
)

dstl <- list(dst.manu, dst.serv, dst.consumer)

ge <- sdm2(
  A = dstl,
  B = matrix(c(
    1, 0, 0,
    0, 1, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = {
    S0Exg <- matrix(NA, 3, 3)
    S0Exg[3, 3] <- 100
    S0Exg
  },
  names.commodity = c("manu", "serv", "lab"),
  names.agent = c("manu", "serv", "consumer"),
  numeraire = c("manu"),
  ts = TRUE,
  policy = list(
    function(time, A) {
      if (time >= 5) {
        node_set(A[[1]], "cc1",
                 a = (1 + tpr)^-(time - 4)
        )
      }
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 40,
  maxIteration = 1,
  z0 = c(160, 60, 100),
  p0 = c(1, 1, 1)
)

matplot(ge$ts.z, type = "o", pch = 20)
matplot(growth_rate(ge$ts.z), type = "o", pch = 20)
matplot(growth_rate(ge$ts.p), type = "o", pch = 20)

#### a 3-by-3 example with labor-saving technological
#### progress and capital accumulation
dst.firm1 <- node_new(
  "prod",
  type = "CD",
  alpha = 2, beta = c(0.5, 0.5),
  "cap", "cc1"
```

```
)
node_set(dst.firm1, "cc1",
         type="Leontief", a=1,
         "lab")

dst.consumer <- dst.firm2 <- node_new(
  "util",
  type = "Leontief",
  a= 1,
  "prod"
)

ge <- sdm2(
  A = list(dst.firm1, dst.consumer, dst.firm2),
  B = matrix(c(
    1, 0, 0.5,
    0, 0, 1,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, NA, NA,
    NA, 100,NA
  ), 3, 3, TRUE),
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm1", "laborer","firm2"),
  numeraire = "prod",
  z0=c(400,200,400),
  policy = list(
    function(time, A) {
      if (time >= 5) {
        node_set(A[[1]],"cc1", a = (1 + 0.03)^-(time - 4))
      }
    },
    policyMarketClearingPrice
  ),
  maxIteration = 1,
  numberOfPeriods = 30,
  ts=TRUE
)

matplot(growth_rate(ge$ts.z), type="l")
```

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] # take commodity 1 as numeraire
  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(A, state) {
  wealth <- t(state$p) %*% state$S
  A[[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]
  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

#### an example with a LES (linear expenditure system) utility function
demand_LES <- function(p, w,
                       gamma = c(0.1, 0.2),
                       beta = c(0.4, 0.6)) {
  d <- c()
  discretionary.income <- w - sum(p * gamma)
  for (k in seq_along(gamma)) {
    d[k] <- gamma[k] + beta[k] * discretionary.income / p[k]
  }

  d
}

A <- function(state) {
  a1 <- CD_A(1, rbind(1 / 3, 2 / 3), state$p)
```

```
  a2 <- demand_LES(state$p, state$w[2])
  cbind(a1, a2)
}

sdm2(
  A = A,
  B = matrix(0, 2, 2),
  S0Exg = matrix(c(
    3, 4,
    7, 0
  ), 2, 2, TRUE),
  names.commodity = c("fish", "banana"),
  names.agent = c("Annie", "Ben"),
  numeraire = "banana"
)
```

`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 (see the first reference)
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 (see the second reference)
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
```

`gemSecurityPricing` *Compute Security Market Equilibria for Some Simple Cases*

Description

Compute the equilibrium of a security (i.e. asset) market by the function sdm2 and by computing marginal utility of securities (see Sharpe, 2008).

Usage

```
gemSecurityPricing(
  S = diag(2),
  UP = diag(nrow(S)),
  uf = NULL,
  muf = NULL,
  ratio_adjust_coef = 0.05,
  numeraire = 1,
  ...
)
```

Arguments

`S`	an n-by-m supply matrix of securities.
`UP`	a unit (security) payoff k-by-n matrix.
`uf`	a utility function or a utility function list.
`muf`	a marginal utility function or a marginal utility function list.
`ratio_adjust_coef`	a scalar indicating the adjustment velocity of demand structure.
`numeraire`	the index of the numeraire commodity.
`...`	arguments to be passed to the function sdm2.

Value

A general equilibrium containing a value marginal utility matrix (VMU).

References

Danthine, J. P., Donaldson, J. (2005, ISBN: 9780123693808) Intermediate Financial Theory. Elsevier Academic Press.

Sharpe, William F. (2008, ISBN: 9780691138503) Investors and Markets: Portfolio Choices, Asset Prices, and Investment Advice. Princeton University Press.

Wang Jiang (2006, ISBN: 9787300073477) Financial Economics. Beijing: China Renmin University Press. (In Chinese)

Xu Gao (2018, ISBN: 9787300258232) Twenty-five Lectures on Financial Economics. Beijing: China Renmin University Press. (In Chinese)

https://web.stanford.edu/~wfsharpe/apsim/index.html

See Also

gemSecurityPricingExample.

Examples

```
gemSecurityPricing(muf = function(x) 1 / x)

gemSecurityPricing(
  S = cbind(c(1, 0), c(0, 2)),
  muf = function(x) 1 / x
)

gemSecurityPricing(
  UP = cbind(c(1, 0), c(0, 2)),
  muf = function(x) 1 / x
)

#### an example of Danthine and Donaldson (2005, section 8.3).
ge <- gemSecurityPricing(
  S = matrix(c(
    10, 5,
    1, 4,
    2, 6
  ), 3, 2, TRUE),
  uf = function(x) 0.5 * x[1] + 0.9 * (1 / 3 * log(x[2]) + 2 / 3 * log(x[3]))
)

ge$p

#### an example of Sharpe (2008, chapter 2, case 1)
secy1 <- c(1, 0, 0, 0, 0)
secy2 <- c(0, 1, 1, 1, 1)
secy3 <- c(0, 5, 3, 8, 4) - 3 * secy2
secy4 <- c(0, 3, 5, 4, 8) - 3 * secy2
# unit security payoff matrix
UP <- cbind(secy1, secy2, secy3, secy4)
```

```
prob <- c(0.15, 0.25, 0.25, 0.35)
wt <- prop.table(c(1, 0.96 * prob)) # weights

geSharpe1 <- gemSecurityPricing(
  S = matrix(c(
    49, 49,
    30, 30,
    10, 0,
    0, 10
  ), 4, 2, TRUE),
  UP = UP,
  uf = list(
    function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / 1.5),
    function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / 2.5)
  )
)
geSharpe1$p
geSharpe1$p[3:4] + 3 * geSharpe1$p[2]

## an example of Sharpe (2008, chapter 3, case 2)
geSharpe2 <- gemSecurityPricing(
  S = matrix(c(
    49, 49, 98, 98,
    30, 30, 60, 60,
    10, 0, 20, 0,
    0, 10, 0, 20
  ), 4, 4, TRUE),
  UP = UP,
  uf = list(
    function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / 1.5),
    function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / 2.5),
    function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / 1.5),
    function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / 2.5)
  )
)

geSharpe2$p
geSharpe2$p[3:4] + 3 * geSharpe2$p[2]
geSharpe2$D

## an example of Sharpe (2008, chapter 3, case 3)
geSharpe3 <- gemSecurityPricing(UP,
  uf = function(x) (x - x^2 / 400) %*% wt,
  S = matrix(c(
    49, 98,
    30, 60,
    5, 10,
    5, 10
  ), 4, 2, TRUE)
)
geSharpe3$p
geSharpe3$p[3:4] + 3 * geSharpe3$p[2]
```

```
# the same as above
geSharpe3b <- gemSecurityPricing(
  S = matrix(c(
    49, 98,
    30, 60,
    5, 10,
    5, 10
  ), 4, 2, TRUE),
  UP = UP,
  muf = function(x) (1 - x / 200) * wt
)

geSharpe3b$p
geSharpe3b$p[3:4] + 3 * geSharpe3b$p[2]

## an example of Sharpe (2008, chapter 3, case 4)
geSharpe4 <- gemSecurityPricing(
  S = matrix(c(
    49, 98,
    30, 60,
    5, 10,
    5, 10
  ), 4, 2, TRUE),
  UP,
  muf = function(x) abs((x - 20)^(-1)) * wt,
  maxIteration = 100,
  numberOfPeriods = 300,
  ts = TRUE
)

geSharpe4$p
geSharpe4$p[3:4] + 3 * geSharpe4$p[2]

## an example of Sharpe (2008, chapter 6, case 14)
prob1 <- c(0.15, 0.26, 0.31, 0.28)
wt1 <- prop.table(c(1, 0.96 * prob1))
prob2 <- c(0.08, 0.23, 0.28, 0.41)
wt2 <- prop.table(c(1, 0.96 * prob2))

uf1 <- function(x) CES(alpha = 1, beta = wt1, x = x, es = 1 / 1.5)
uf2 <- function(x) CES(alpha = 1, beta = wt2, x = x, es = 1 / 2.5)
geSharpe14 <- gemSecurityPricing(
  S = matrix(c(
    49, 49,
    30, 30,
    10, 0,
    0, 10
  ), 4, 2, TRUE),
  UP = UP,
  uf = list(uf1,uf2)
)

geSharpe14$D
```

```
geSharpe14$p
geSharpe14$p[3:4] + 3 * geSharpe14$p[2]
mu <- marginal_utility(geSharpe14$Payoff, diag(5),uf=list(uf1,uf2))
mu[,1]/mu[1,1]
mu[,2]/mu[1,2]

#### an example of Wang (2006, example 10.1, P146)
geWang <- gemSecurityPricing(
  S = matrix(c(
    1, 0,
    0, 2,
    0, 1
  ), 3, 2, TRUE),
  muf = list(
    function(x) 1 / x * c(0.5, 0.25, 0.25),
    function(x) 1 / sqrt(x) * c(0.5, 0.25, 0.25)
  )
)

geWang$p # c(1, (1 + sqrt(17)) / 16)

# the same as above
geWang.b <- gemSecurityPricing(
  S = matrix(c(
    1, 0,
    0, 2,
    0, 1
  ), 3, 2, TRUE),
  uf = list(
    function(x) log(x) %*% c(0.5, 0.25, 0.25),
    function(x) 2 * sqrt(x) %*% c(0.5, 0.25, 0.25)
  )
)

geWang.b$p

#### an example of Xu (2018, section 10.4, P151)
wt <- c(1, 0.5, 0.5)
ge <- gemSecurityPricing(
  S = matrix(c(
    1, 0,
    0, 0.5,
    0, 2
  ), 3, 2, TRUE),
  uf = list(
    function(x) CRRA(x, gamma = 1, prob = wt)$u,
    function(x) CRRA(x, gamma = 0.5, prob = wt)$u
  )
)

ge$p # c(1, (1 + sqrt(5)) / 4, (1 + sqrt(17)) / 16)

#### an example of incomplete market
```

```
ge <- gemSecurityPricing(
  UP = cbind(c(1, 1), c(2, 1)),
  uf = list(
    function(x) sum(log(x)) / 2,
    function(x) sum(sqrt(x))
  ),
  ratio_adjust_coef = 0.1,
  priceAdjustmentVelocity = 0.05,
  policy = makePolicyMeanValue(span = 100),
  maxIteration = 1,
  numberOfPeriods = 2000,
)

ge$p

## the same as above
ge.b <- gemSecurityPricing(
  UP = cbind(c(1, 1), c(2, 1)),
  muf = list(
    function(x) 1 / x * c(0.5, 0.5),
    function(x) 1 / sqrt(x) * c(0.5, 0.5)
  ),
  ratio_adjust_coef = 0.1,
  priceAdjustmentVelocity = 0.05,
  policy = makePolicyMeanValue(span = 100),
  maxIteration = 1,
  numberOfPeriods = 2000,
  ts = TRUE
)

ge.b$p
matplot(ge.b$ts.p, type = "l")

#### an example with outside position.
asset1 <- c(1, 0, 0)
asset2 <- c(0, 1, 1)

# unit (asset) payoff matrix
UP <- cbind(asset1, asset2)
wt <- c(0.5, 0.25, 0.25) # weights

uf1 <- function(x) prod((x + c(0, 0, 2))^wt)
uf2 <- function(x) prod(x^wt)

ge <- gemSecurityPricing(
  S = matrix(c(
    1, 1,
    0, 2
  ), 2, 2, TRUE),
  UP = UP,
  uf = list(uf1, uf2),
  numeraire = 1
)
```

```
ge$p
ge$z
uf1(ge$Payoff[,1])
uf2(ge$Payoff[,2])
```

---

gemSecurityPricingExample *Some Examples of Security Pricing*

Description

These examples illustrate how to find the equilibrium of a security (i.e. asset) market by the function sdm2 and by computing marginal utility of securities (see Sharpe, 2008).

Usage

```
gemSecurityPricingExample(...)
```

Arguments

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

Value

A general equilibrium.

References

Danthine, J. P., Donaldson, J. (2005, ISBN: 9780123693808) Intermediate Financial Theory. Elsevier Academic Press.

Sharpe, William F. (2008, ISBN: 9780691138503) Investors and Markets: Portfolio Choices, Asset Prices, and Investment Advice. Princeton University Press.

Xu Gao (2018, ISBN: 9787300258232) Twenty-five Lectures on Financial Economics. Beijing: China Renmin University Press. (In Chinese)

https://web.stanford.edu/~wfsharpe/apsim/index.html

See Also

gemSecurityPricing.

Examples

```
#### an example of Danthine and Donaldson (2005, section 8.3).
uf <- function(x) 0.5 * x[1] + 0.9 * (1 / 3 * log(x[2]) + 2 / 3 * log(x[3]))

ge <- sdm2(
  A = function(state) {
    VMU <- marginal_utility(state$last.A %*% dg(state$last.z), diag(3), uf, state$p)
    Ratio <- sweep(VMU, 2, colMeans(VMU), "/")

    A <- state$last.A * Ratio
    prop.table(A, 2)
  },
  B = matrix(0, 3, 2),
  S0Exg = matrix(c(
    10, 5,
    1, 4,
    2, 6
  ), 3, 2, TRUE),
  names.commodity = c("secy1", "secy2", "secy3"),
  names.agent = c("agt1", "agt2"),
  numeraire = "secy1",
  ts = TRUE
)

ge$p

#### an example of Sharpe (2008, chapter 2)
secy1 <- c(1, 0, 0, 0, 0)
secy2 <- c(0, 1, 1, 1, 1)
secy3 <- c(0, 5, 3, 8, 4) - 3 * secy2
secy4 <- c(0, 3, 5, 4, 8) - 3 * secy2
# unit (security) payoff matrix
UP <- cbind(secy1, secy2, secy3, secy4)

prob <- c(0.15, 0.25, 0.25, 0.35)
wt <- prop.table(c(1, 0.96 * prob)) # weights

gamma.agt1 <- 1.5
gamma.agt2 <- 2.5

ge <- sdm2(
  A = function(state) {
    Payoff <- UP %*% (state$last.A %*% dg(state$last.z))

    VMU <- marginal_utility(Payoff, UP, list(
      function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / gamma.agt1),
      function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / gamma.agt2)
    ), price = state$p)
    Ratio <- sweep(VMU, 2, colMeans(VMU), "/")

    A <- state$last.A * ratio_adjust(Ratio, coef = 0.05, method = "linear")
```

```
    A <- prop.table(A, 2)
  },
  B = matrix(0, 4, 2),
  S0Exg = matrix(c(
    49, 49,
    30, 30,
    10, 0,
    0, 10
  ), 4, 2, TRUE),
  names.commodity = c("secy1", "secy2", "secy3", "secy4"),
  names.agent = c("agt1", "agt2"),
  numeraire = "secy1"
)

ge$p
ge$p[3:4] + 3 * ge$p[2]

#### an example of Xu (2018, section 10.4, P151)
secy1 <- c(1, 0, 0)
secy2 <- c(0, 1, 0)
secy3 <- c(0, 0, 1)
prob <- c(0.5, 0.5)
wt <- c(1, prob)
UP <- cbind(secy1, secy2, secy3)

gamma.agt1 <- 1
gamma.agt2 <- 0.5

ge <- sdm2(
  A = function(state) {
    Payoff <- UP %*% (state$last.A %*% dg(state$last.z))

    VMU <- marginal_utility(Payoff, UP, list(
      # Here CRRA(...)$u, CRRA(...)$CE and CES functions are interexchangeable.
      function(x) CRRA(x, gamma = gamma.agt1, p = wt)$u,
      function(x) CES(alpha = 1, beta = wt, x = x, es = 1 / gamma.agt2)
    ), state$p)
    Ratio <- sweep(VMU, 2, colMeans(VMU), "/")

    A <- state$last.A * Ratio
    prop.table(A, 2)
  },
  B = matrix(0, 3, 2),
  S0Exg = matrix(c(
    1, 0,
    0, 0.5,
    0, 2
  ), 3, 2, TRUE),
  names.commodity = c("secy1", "secy2", "secy3"),
  names.agent = c("agt1", "agt2"),
  numeraire = "secy1",
  maxIteration = 1,
```

```
  ts = TRUE
)

ge$p #c(1, (1 + sqrt(5)) / 4, (1 + sqrt(17)) / 16)

## the same as above.
dst.agt1 <- node_new("util",
  type = "CD", alpha = 1, beta = c(0.5, 0.25, 0.25),
  "secy1", "secy2", "secy3"
)

dst.agt2 <- node_new("util",
  type = "CES", alpha = 1, beta = c(2, 1, 1), sigma = 0.5,
  "secy1", "secy2", "secy3"
)

ge <- sdm2(
  A = list(dst.agt1, dst.agt2),
  B = matrix(0, 3, 2),
  S0Exg = matrix(c(
    1, 0,
    0, 0.5,
    0, 2
  ), 3, 2, TRUE),
  names.commodity = c("secy1", "secy2", "secy3"),
  names.agent = c("agt1", "agt2"),
  numeraire = "secy1",
  maxIteration = 1,
  ts = TRUE
)

ge$p

#### an example with production.
asset1 <- c(1, 0, 0, 0, 0, 0)
asset2 <- c(0, 1, 0, 0, 0, 0)
asset3 <- c(0, 0, 1, 3, 1, 2)
asset4 <- c(0, 0, 4, 2, 6, 2)
asset5 <- c(0, 0, 1, 0, 2, 0)

# unit (asset) payoff matrix
UP <- cbind(asset1, asset2, asset3, asset4, asset5)

muf1 <- function(x) 1 / x
muf2 <- function(x) 1 / x * c(0.4, 0.1, 0.2, 0.05, 0.2, 0.05)

ge <- sdm2(
  A = function(state) {
    Payoff <- UP %*% (state$last.A[, 1:2] %*% dg(state$last.z[1:2]))

    VMU <- marginal_utility(Payoff, UP, muf = list(muf1, muf2), price = state$p)
    Ratio <- sweep(VMU, 2, colMeans(VMU), "/")
```

```
    A <- state$last.A[, 1:2] * ratio_adjust(Ratio, coef = 0.15, method = "linear")
    A <- prop.table(A, 2)

    a.firm <- CD_A(alpha = 4, Beta = c(0.5, 0.5, 0, 0, 0), state$p)
    A <- cbind(A, a.firm)
  },
  B = matrix(c(
    0, 0, 0,
    0, 0, 0,
    0, 0, 0,
    0, 0, 0,
    0, 0, 1
  ), 5, 3, TRUE),
  S0Exg = matrix(c(
    1, 1, NA,
    1, 2, NA,
    1, NA, NA,
    NA, 1, NA,
    NA, NA, NA
  ), 5, 3, TRUE),
  names.commodity = c("asset1", "asset2", "asset3", "asset4", "asset5"),
  names.agent = c("consumer1", "consumer2", "firm"),
  numeraire = "asset1"
)

ge$p
ge$z

#### an example with demand structure trees.
asset1 <- c(1, 0, 0, 0, 0)
asset2 <- c(0, 1, 3, 1, 2)
asset3 <- c(0, 2, 1, 3, 1)

# unit payoff matrix
UP <- cbind(asset1, asset2, asset3)

dst.consumer1 <- node_new("util",
                          type = "CES", es = 0.5, alpha = 1, beta = c(0.5, 0.5),
                          "x1", "u2"
)
node_set(dst.consumer1, "u2",
         type = "CES", es = 0.8, alpha = 1, beta = c(0.6, 0.4),
         "u2.1", "u2.2"
)
node_set(dst.consumer1, "u2.1",
         type = "CES", es = 1, alpha = 1, beta = c(0.8, 0.2),
         "x2", "x3"
)
node_set(dst.consumer1, "u2.2",
         type = "CES", es = 1, alpha = 1, beta = c(0.8, 0.2),
         "x4", "x5"
)
```

```
dst.consumer2 <- node_new("util",
                          type = "CES", es = 0.5, alpha = 1, beta = c(0.5, 0.5),
                          "x1", "u2"
)
node_set(dst.consumer2, "u2",
         type = "CES", es = 0.8, alpha = 1, beta = c(0.6, 0.4),
         "u2.1", "u2.2"
)
node_set(dst.consumer2, "u2.1",
         type = "CES", es = 1, alpha = 1, beta = c(0.2, 0.8),
         "x2", "x3"
)
node_set(dst.consumer2, "u2.2",
         type = "CES", es = 1, alpha = 1, beta = c(0.2, 0.8),
         "x4", "x5"
)

uf1 <- function(x) {
  names(x) <- paste0("x", seq_along(x))
  output(dst.consumer1, x)
}

uf2 <- function(x) {
  names(x) <- paste0("x", seq_along(x))
  output(dst.consumer2, x)
}

ge <- gemSecurityPricing(
  S = matrix(c(
    3, 3,
    1, 0,
    0, 2
  ), 3, 2, TRUE),
  UP = UP,
  uf = list(uf1, uf2)
)

ge$p
ge$z
```

`gemstEndogenousProductionFunction`
An Endogenous Production Function

Description

This is an example of the market-clearing path with an endogenous production function. The parameter of the production function will change with the output level.

To deal with locally or globally increasing returns to scale, we can simply use an endogenous CES-type production function instead of a production function with a more complex form.

Usage

```
gemstEndogenousProductionFunction(...)
```

Arguments

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

Examples

```
dst.firm <- node_new(
  "output",
  type = "CD", alpha = 5, beta = c(0.5, 0.5),
  "prod", "lab"
)

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

ge <- sdm2(
  A = list(dst.firm, dst.consumer),
  B = matrix(c(
    1, 0,
    0, 1
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 1
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "lab",
  z0 = c(1, 1),
  p0 = c(1, 1),
  ts = TRUE,
  policy = list(
    function(A, state) {
      A[[1]]$alpha <- 5 * state$last.z[1]^0.1
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 40,
  maxIteration = 1
)

matplot(ge$ts.z, type = "o", pch = 20)
```

gemstEndogenousUtilityFunction
Some General Equilibrium Models with Endogenous Utility Function

Description

Some examples of the market-clearing path with an endogenous utility function. The parameters of the utility function will change with the utility level.

To deal with non-homothetic preferences, we can simply use an endogenous CES-type utility function instead of a utility function with a more complex form.

Usage

```
gemstEndogenousUtilityFunction(...)
```

Arguments

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

Examples

```
#### a 2-by-2 example
dst.firm <- node_new(
  "output",
  type = "CD", alpha = 5, beta = c(0.5, 0.5),
  "prod", "lab"
)

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "prod", "lab"
)

ge <- sdm2(
  A = list(dst.firm, dst.consumer),
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 1
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "lab",
  z0 = c(0.01, 1),
```

```
  p0 = c(1, 1),
  ts = TRUE,
  policy = list(
    function(A, state) {
      util <- state$last.z[2]
      beta2 <- 0.95 * plogis(util, location = 2, scale = 2)
      A[[2]]$beta <- c(1 - beta2, beta2)
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 20,
  maxIteration = 1
)

matplot(ge$ts.z, type = "o", pch = 20)
ge$z
dst.consumer$beta

#### a 3-by-3 example with 100 laborers
#### Assume that each laborer desires to consume one unit of
#### corn per period, regardless of her level of utility.
dst.firm.corn <- node_new(
  "corn",
  type = "CD", alpha = 1, beta = c(0.5, 0.5),
  "iron", "lab"
)

dst.firm.iron <- node_new(
  "iron",
  type = "CD", alpha = 5, beta = c(0.5, 0.5),
  "iron", "lab"
)

dst.consumer <- node_new(
  "util",
  type = "Leontief",
  a = c(0.5, 0.5),
  "corn", "iron"
)

ge <- sdm2(
  A = list(dst.firm.corn, dst.firm.iron, dst.consumer),
  B = matrix(c(
    1, 0, 0,
    0, 1, 0,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, NA, NA,
    NA, NA, 100
  ), 3, 3, TRUE),
  names.commodity = c("corn", "iron", "lab"),
```

```
  names.agent = c("firm.corn", "firm.iron", "consumer"),
  numeraire = "lab",
  ts = TRUE,
  policy = list(
    function(A, state) {
      last.util <- state$last.z[3] / 100 # the previous utility level of each laborer
      a1 <- min(1 / last.util, 1)
      A[[3]]$a <- c(a1, 1 - a1)
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 40,
  maxIteration = 1
)

matplot(ge$ts.z, type = "o", pch = 20)
ge$z
ge$A
ge$D

#### a 4-by-4 example with 100 homogeneous laborers
dst.agri <- node_new(
  "output",
  type = "SCES",
  es = 0.5, alpha = 2, beta = c(0.2, 0.8),
  "manu", "lab"
)

dst.manu <- node_new(
  "output",
  type = "SCES",
  es = 0.5, alpha = 3, beta = c(0.6, 0.4),
  "manu", "lab"
)

dst.serv <- node_new(
  "output",
  type = "SCES",
  es = 0.5, alpha = 2, beta = c(0.4, 0.6),
  "manu", "lab"
)

dst.consumer <- node_new(
  "util",
  type = "CD", alpha = 1, beta = c(0.6, 0.3, 0.1),
  "agri", "manu", "serv"
)

ge <- sdm2(
  A = list(dst.agri, dst.manu, dst.serv, dst.consumer),
  B = diag(c(1, 1, 1, 0)),
  S0Exg = {
    tmp <- matrix(NA, 4, 4)
```

```
    tmp[4, 4] <- 100
    tmp
  },
  names.commodity = c("agri", "manu", "serv", "lab"),
  names.agent = c("agri", "manu", "serv", "consumer"),
  numeraire = "lab",
  z0 = c(1, 1, 1, 0),
  ts = TRUE,
  policy = list(
    function(A, state) {
      util <- state$last.z[4] / 100
      beta1 <- structural_function(util, c(1, 6), 0.6, 0.1)
      beta3 <- structural_function(util, c(1, 6), 0.1, 0.5)
      beta2 <- 1 - beta1 - beta3
      A[[4]]$beta <- c(beta1, beta2, beta3)
    },
    policyMarketClearingPrice
  ),
  numberOfPeriods = 20,
  maxIteration = 1
)

matplot(ge$ts.z, type = "o", pch = 20)
ge$z
dst.consumer$beta
```

`gemstStructuralMultipleEquilibria_2_2`
Structural Multiple Equilibria and Structural Transition Policy

---

Description

Some examples of structural multiple equilibria and structural transition policy. In these examples it is assumed that the firm has a structural production function (see Li, Fu, 2020), e.g.

structural_function(last.output, c(0.3, 0.4), 1, 2) * x1^0.35 * x2^0.65

wherein last.output is the output of the firm in the previous period.

Usage

```
gemstStructuralMultipleEquilibria_2_2(...)
```

Arguments

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

References

Li Wu, Fu Caihui (2020) A Simulation Study on the Economic Structure Transition Policy. Journal of Shanghai University (Social Sciences). 37(2), pp: 33-45. (In Chinese)

Examples

```
dst.firm <- node_new("output",
                     type = "CD", alpha = 1,
                     beta = c(0.35, 0.65),
                     "prod", "lab"
)

dst.consumer <- node_new("utility",
                         type = "CD", alpha = 1,
                         beta = c(0.4, 0.6),
                         "prod", "lab"
)

policy.technology <- function(time, state, A) {
  # state$last.z[1] is the previous output.
  A[[1]]$alpha <- structural_function(state$last.z[1], c(0.3, 0.4), 1, 2)
}

policy.tax <- function(time, state) {
  if ((time >= 15) && state$last.z[1] < 0.4) {
    state$S[2, 2] <- 0.8
    state$S[2, 1] <- 0.2
  } else {
    state$S[2, 2] <- 1
    state$S[2, 1] <- 0
  }

  state
}

f <- function(z0 = c(0.1, 1),
              policy = list(
                policy.technology,
                policyMarketClearingPrice
              )) {
  ge <- sdm2(
    A = list(dst.firm, dst.consumer),
    B = matrix(c(
      1, 0,
      0, 0
    ), 2, 2, TRUE),
    S0Exg = matrix(c(
      NA, NA,
      NA, 1
    ), 2, 2, TRUE),
    names.commodity = c("prod", "lab"),
```

```
    names.agent = c("firm", "consumer"),
    numeraire = "lab",
    z0 = z0,
    p0 = c(1, 1),
    maxIteration = 1,
    numberOfPeriods = 30,
    policy = policy,
    ts = TRUE
  )

  matplot(ge$ts.z, type = "o", pch = 20)
  ge
}

geLow <- f()
geLow$z

geHigh <- f(z0 = c(0.5, 1))
geHigh$z

f(policy = list(
  policy.technology,
  policy.tax,
  policyMarketClearingPrice
))

#### structural transition: disequilibrium path and
## market-clearing path (instantaneous equilibrium path)
dst.firm <- node_new("output",
                     type = "CD", alpha = 5,
                     beta = c(0.5, 0.5),
                     "prod", "lab"
)

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

policy.technology <- function(time, state, A) {
  # state$last.z[1] is last output.
  A[[1]]$alpha <- structural_function(state$last.z[1], c(15, 20), 5, 15)
  return(NULL)
}

policy.tax <- function(time, state) {
  if ((time >= 100) && (time <= 109)) {
    state$S[2, 2] <- 0.6
    state$S[2, 1] <- 0.4
  } else {
    state$S[2, 2] <- 1
    state$S[2, 1] <- 0
  }
```

```
  state
}

f <- function(z0 = c(1, 1),
              p0 = c(1, 1),
              policy = policy.technology) {
  ge <- sdm2(
    A = list(dst.firm, dst.consumer),
    B = matrix(c(
      1, 0,
      0, 1
    ), 2, 2, TRUE),
    S0Exg = matrix(c(
      NA, NA,
      NA, 1
    ), 2, 2, TRUE),
    names.commodity = c("prod", "lab"),
    names.agent = c("firm", "consumer"),
    numeraire = "lab",
    z0 = z0,
    p0 = p0,
    maxIteration = 1,
    numberOfPeriods = 200,
    policy = policy,
    priceAdjustmentVelocity = 0.4,
    ts = TRUE
  )

  matplot(ge$ts.z, type = "l", pch = 20)
  ge
}

geLow <- f()
geLow$z

geHigh <- f(z0 = c(18, 1), p0 = c(1, 9))
geHigh$z

## structural transition: disequilibrium path
f(policy = list(
  policy.technology,
  policy.tax
))$z


## structural transition: market-clearing path
f(policy = list(
  policy.technology,
  policy.tax,
  policyMarketClearingPrice
))$z
```

```
## structural transition through foreign aid
policy.foreign_aid <- function(time, state) {
  if ((time >= 100) && (time <= 109)) {
    state$S[2, 2] <- 3
  } else {
    state$S[2, 2] <- 1
  }

  state
}

f(policy = list(
  function(time, state, A) { # technology policy
    # state$last.z[1] is last output.
    A[[1]]$alpha <- structural_function(state$last.z[1], c(30, 35), 5, 15)
  },
  policy.foreign_aid
))
```

---

gemTax_3_3 *Some General Equilibrium Models with Tax*

Description

Some general equilibrium models with tax.

Usage

```
gemTax_3_3(...)
```

Arguments

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

References

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

Examples

```
#### turnover tax
dst.firm <- node_new("prod",
                     type = "FIN",
                     rate = c(1, tax.rate = 0.25),
                     "cc1", "tax"
)
node_set(dst.firm, "cc1",
```

```
          type = "CD",
          alpha = 2, beta = c(0.5, 0.5),
          "prod", "lab"
)

dst.laborer <- dst.government <-
  node_new("util",
    type = "Leontief",
    a = 1,
    "prod"
  )

ge.TT <- sdm2(
  A = list(dst.firm, dst.laborer, dst.government),
  B = diag(c(1, 0, 0)),
  S0Exg = {
    S0Exg <- matrix(NA, 3, 3)
    S0Exg[2, 2] <- 100
    S0Exg[3, 3] <- 100
    S0Exg
  },
  names.commodity = c("prod", "lab", "tax"),
  names.agent = c("firm", "laborer", "government"),
  numeraire = "prod"
)

ge.TT$p
ge.TT$z
ge.TT$D
ge.TT$S

#### product tax
dst.taxed.prod <- node_new("taxed.prod",
                           type = "FIN",
                           rate = c(1, tax.rate = 0.25),
                           "prod", "tax"
)

dst.firm <- node_new("prod",
                     type = "CD",
                     alpha = 2, beta = c(0.5, 0.5),
                     dst.taxed.prod, "lab"
)

dst.laborer <- dst.government <-
  node_new("util",
    type = "Leontief",
    a = 1,
    dst.taxed.prod
  )

ge.PT <- sdm2(
  A = list(dst.firm, dst.laborer, dst.government),
```

```
  B = diag(c(1, 0, 0)),
  S0Exg = {
    S0Exg <- matrix(NA, 3, 3)
    S0Exg[2, 2] <- 100
    S0Exg[3, 3] <- 100
    S0Exg
  },
  names.commodity = c("prod", "lab", "tax"),
  names.agent = c("firm", "laborer", "government"),
  numeraire = "prod"
)

ge.PT$p
ge.PT$z
ge.PT$D
ge.PT$S

#### consumption tax
dst.firm <- node_new("output",
                     type = "CD", alpha = 2,
                     beta = c(0.5, 0.5),
                     "prod", "lab"
)

dst.laborer <- node_new("util",
                        type = "FIN",
                        rate = c(1, consumption.tax.rate = 1/3),
                        "prod", "tax"
)

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

ge.CT <- sdm2(
  A = list(dst.firm, dst.laborer, dst.government),
  B = diag(c(1, 0, 0)),
  S0Exg = {
    S0Exg <- matrix(NA, 3, 3)
    S0Exg[2, 2] <- 100
    S0Exg[3, 3] <- 100
    S0Exg
  },
  names.commodity = c("prod", "lab", "tax"),
  names.agent = c("firm", "laborer", "government"),
  numeraire = "prod"
)

ge.CT$p
ge.CT$z
ge.CT$D
```

```
ge.CT$S

#### value added tax
dst.firm <- node_new("output",
  type = "CD", alpha = 2,
  beta = c(0.5, 0.5),
  "prod", "taxed.lab"
)
node_set(dst.firm, "taxed.lab",
  type = "FIN",
  rate = c(1, vat.rate = 1/3),
  "lab", "tax"
)

dst.laborer <- dst.government <-
  node_new("util",
    type = "Leontief",
    a = 1,
    "prod"
  )

ge.VAT <- sdm2(
  A = list(dst.firm, dst.laborer, dst.government),
  B = diag(c(1, 0, 0)),
  S0Exg = {
    S0Exg <- matrix(NA, 3, 3)
    S0Exg[2, 2] <- S0Exg[3, 3] <- 100
    S0Exg
  },
  names.commodity = c("prod", "lab", "tax"),
  names.agent = c("firm", "laborer", "government"),
  numeraire = "prod"
)

ge.VAT$p
ge.VAT$z
ge.VAT$D
ge.VAT$S

#### income tax
income.tax.rate <- 1 / 4

dst.firm <- node_new("output",
                     type = "CD",
                     alpha = 2, beta = c(0.5, 0.5),
                     "prod", "lab"
)

dst.laborer <- dst.government <-
  node_new("util",
    type = "Leontief",
    a = 1,
    "prod"
```

```
  )

ge.IT <- sdm2(
  A = list(dst.firm, dst.laborer, dst.government),
  B = diag(c(1, 0),2,3),
  S0Exg = {
    S0Exg <- matrix(NA, 2, 3)
    S0Exg[2, 2] <- 100 * (1 - income.tax.rate)
    S0Exg[2, 3] <- 100 * income.tax.rate
    S0Exg
  },
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "laborer", "government"),
  numeraire = "prod"
)

ge.IT$p
ge.IT$z
ge.IT$D
ge.IT$S

#### turnover tax (Li, 2019, example 4.11)
dst.firm <- node_new("output",
                     type = "FIN",
                     rate = c(1, turnover.tax.rate = 1),
                     "cc1", "tax"
)
node_set(dst.firm, "cc1",
         type = "CD",
         alpha = 1, beta = c(0.5, 0.5),
         "prod", "lab"
)

dst.laborer1 <- node_new("util",
                          type = "CD",
                          alpha = 1, beta = c(0.5, 0.5),
                          "prod", "lab"
)

dst.laborer2 <- node_new("utility",
                          type = "Leontief",
                          a = 1,
                          "prod"
)

ge.TT2 <- sdm2(
  A = list(dst.firm, dst.laborer1, dst.laborer2),
  B = diag(c(1, 0, 0)),
  S0Exg = {
    S0Exg <- matrix(NA, 3, 3)
    S0Exg[2, 2] <- S0Exg[2, 3] <-
      S0Exg[3, 3] <- 100
    S0Exg
```

```
  },
  names.commodity = c("prod", "lab", "tax"),
  names.agent = c("firm", "laborer1", "laborer2"),
  numeraire = "lab"
)

ge.TT2$p
ge.TT2$z
```

---

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, A, 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(A[[1]], "taxed.lab")$rate[2]
    current.tax.rate <- last.tax.rate / adjustment.ratio
  } else {
    current.tax.rate <- 0.1
  }
  node_set(A[[1]], "taxed.lab", rate = c(1, current.tax.rate))
  node_set(A[[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]
```

---

gemTax_5_5 *A General Equilibrium Model with Tax*

Description

A general equilibrium model with tax. The model contains 5 types of commodities (i.e. prod1, prod2, labor, capital and tax payment receipts) and 5 agents (i.e. firm1, firm2, laborer, capitalist and government).

Usage

```
gemTax_5_5(...)
```

Arguments

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

Examples

```
tax.rate.cap1 <- 0.25
tax.rate.lab1 <- 0.25

tax.rate.cap2 <- 0.25
tax.rate.lab2 <- 0.25

es.prod <- 0.5
es.cap.lab <- 0.5

beta.firm1 <- c(0.2, 0.8)
beta.firm2 <- c(0.8, 0.2)

beta.laborer <- c(0.5, 0.5)
beta.capitalist <- c(0.5, 0.5)
beta.government <- c(0.8, 0.2)
```

```
dst.firm1 <- node_new("prod",
                      type = "SCES",
                      alpha = 1, beta = beta.firm1, es = es.cap.lab,
                      "cc1", "cc2"
)
node_set(dst.firm1, "cc1",
         type = "FIN",
         rate = c(1, tax.rate = tax.rate.lab1),
         "lab", "tax"
)
node_set(dst.firm1, "cc2",
         type = "FIN",
         rate = c(1, tax.rate = tax.rate.cap1),
         "cap", "tax"
)

node_plot(dst.firm1, TRUE)

dst.firm2 <- node_new("prod",
                      type = "SCES",
                      alpha = 1, beta = beta.firm2, es = es.cap.lab,
                      "cc1", "cc2"
)
node_set(dst.firm2, "cc1",
         type = "FIN",
         rate = c(1, tax.rate = tax.rate.lab2),
         "lab", "tax"
)
node_set(dst.firm2, "cc2",
         type = "FIN",
         rate = c(1, tax.rate = tax.rate.cap2),
         "cap", "tax"
)

dst.laborer <- node_new("util",
                        type = "SCES",
                        alpha = 1, beta = beta.laborer, es = es.prod,
                        "prod1", "prod2"
)

dst.capitalist <- node_new("util",
                           type = "SCES",
                           alpha = 1, beta = beta.capitalist, es = es.prod,
                           "prod1", "prod2"
)

dst.government <- node_new("util",
                           type = "SCES",
                           alpha = 1, beta = beta.government, es = es.prod,
                           "prod1", "prod2"
)

ge <- sdm2(
```

```
  A = list(dst.firm1, dst.firm2, dst.laborer, dst.capitalist, dst.government),
  B = diag(c(1, 1, 0, 0, 0)),
  S0Exg = {
    S0Exg <- matrix(NA, 5, 5)
    S0Exg[3, 3] <- 100
    S0Exg[4, 4] <- 100
    S0Exg[5, 5] <- 100
    S0Exg
  },
  names.commodity = c("prod1", "prod2", "lab", "cap", "tax"),
  names.agent = c("firm1", "firm2", "laborer", "capitalist", "government"),
  numeraire = "lab"
)

ge$p
ge$z
ge$D
ge$S
addmargins(ge$DV)
addmargins(ge$SV)
```

`gemTax_QuasilinearPreference_4_4` *A General Equilibrium Model with Tax and Quasilinear Utility Functions.*

Description

This model is essentially a pure exchange economy. The model contains 4 types of commodities (i.e. corn, iron, taxed iron and tax payment receipts) and 4 agents (i.e. consumer 1, consumer 2, a firm and the government). Consumer 1 has corn and the utility function is x1 + beta1 * (alpha1 * x3 - 0.5 * x3^2) wherein x1 is corn and x3 is taxed iron. Consumer 2 has iron and the utility function is x1 + beta2 * (alpha2 * x2 - 0.5 * x2^2) wherein x1 is corn and x2 is iron. Consumer 1 (i.e. the iron demander) wants to buy iron from consumer 2 (i.e. the iron supplier) and the government will tax the transaction. The firm (i.e. a tax agency) inputs iron and tax payment receipts (similar to tax stamp) to output taxed iron, and due to government taxation requirements consumer 1 have to buy taxed iron from the firm and consumer 2 have to sell iron through the firm. Government has tax payment receipts and the utility function is x1.

Usage

```
gemTax_QuasilinearPreference_4_4(...)
```

Arguments

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

Examples

```
tax.rate <- 1
iron.supply <- 100

beta1 <- 0.05
alpha1 <- 3 / beta1 + 60

beta2 <- 0.05
alpha2 <- iron.supply - 60 + (3 / beta2)

demand_consumer1 <- function(p, w) {
  d <- rbind(0, 0)
  w <- w / p[1]
  p <- p / p[1] # take corn as numeraire
  d[2] <- max(alpha1 - p[2] / beta1, 0)
  if (d[2] * p[2] > w) {
    d[2] <- w / p[2]
    d[1] <- 0
  } else {
    d[1] <- w - d[2] * p[2]
  }

  d
}

demand_consumer2 <- function(p, w) {
  d <- rbind(0, 0)
  w <- w / p[1]
  p <- p / p[1]
  d[2] <- max(alpha2 - p[2] / beta2, 0)
  if (d[2] * p[2] > w) {
    d[2] <- w / p[2]
    d[1] <- 0
  } else {
    d[1] <- w - d[2] * p[2]
  }

  d
}

ge <- sdm2(
  A = function(state) {
    a1 <- demand_consumer1(c(state$p[1], state$p[3]), state$w[1])
    a1 <- c(a1[1], 0, a1[2], 0)

    a2 <- demand_consumer2(state$p[1:2], state$w[2])
    a2 <- c(a2, 0, 0)

    a.firm <- c(0, 1, 0, tax.rate * state$p[2] / state$p[4])

    a.gov <- c(1, 0, 0, 0)
```

```
    cbind(a1, a2, a.firm, a.gov)
  },
  B = matrix(c(
    0, 0, 0, 0,
    0, 0, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 0
  ), 4, 4, TRUE),
  S0Exg = matrix(c(
    1000, NA, NA, NA,
    NA, iron.supply, NA, NA,
    NA, NA, NA, NA,
    NA, NA, NA, 1
  ), 4, 4, TRUE),
  names.commodity = c("corn", "iron", "taxed.iron", "tax"),
  names.agent = c("consumer1", "consumer2", "firm", "gov"),
  numeraire = "corn",
  priceAdjustmentVelocity = 0.05
)

ge$p
ge$D
ge$S
addmargins(ge$DV)
addmargins(ge$SV)

ge.x <- ge$D[3, 1]
ge.pl <- ge$p[2]
ge.ph <- ge$p[3]
plot(function(x) (alpha1 - x) * beta1, 0, alpha1,
     xlim = c(0, 100), ylim = c(0, 6), xlab = "iron", ylab = "price"
)
curve((alpha2 - iron.supply + x) * beta2, 0,
      alpha1,
      add = TRUE
)
grid()
points(ge.x, ge.ph, col = "red", pch = 20) # pch=8
points(ge.x, ge.pl, col = "red", pch = 20)

polygon(c(0, ge.x, ge.x, 0), c(ge.ph, ge.ph, ge.pl, ge.pl))
segments(0, 3, x1 = 60, y1 = 3, col = "red")
text(c(0, ge.x, ge.x, 0) + 3, c(ge.ph + 0.3, ge.ph + 0.3,
  ge.pl - 0.3, ge.pl - 0.3), c("A", "B", "C", "D"))
text(c(3, ge.x + 3, 60), 3.3, c("E", "F", "G"))

u.consumer1 <- function(x) x[1] + beta1 * (alpha1 * x[2] - 0.5 * x[2]^2)
u.consumer2 <- function(x) x[1] + beta2 * (alpha2 * x[2] - 0.5 * x[2]^2)

u.consumer1(ge$D[c(1, 3), 1]) + u.consumer2(ge$D[c(1:2), 2]) + ge$z[4]
# The value above is 1430 when the tax rate is 0.
```

gemTax_VAT_IncomeTax_5_4

A General Equilibrium Model with Value-added Tax and Income Tax

Description

A general equilibrium model with value-added tax and income tax.

Usage

```
gemTax_VAT_IncomeTax_5_4(...)
```

Arguments

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

Examples

```
vat.rate <- 0.2 # value-added tax rate
income.tax.rate <- 0.2

dst.manu <- node_new("output",
                     type = "FIN",
                     rate = c(5 / 6, vat.rate),
                     "cc1", "vat"
)
node_set(dst.manu, "cc1",
         type = "SCES",
         es = 0.5, alpha = 1,
         beta = c(0.25, 0.75),
         "lab", "cap"
)

dst.serv <- node_new("output",
                     type = "FIN",
                     rate = c(5 / 6, vat.rate),
                     "cc1", "vat"
)
node_set(dst.serv, "cc1",
         type = "SCES",
         es = 0.5, alpha = 1,
         beta = c(0.75, 0.25),
         "lab", "cap"
)

dst.household <- node_new("util",
                          type = "SCES",
                          es = 0.5, alpha = 1,
                          beta = c(0.2, 0.8),
```

```
                          "manu", "serv"
)

dst.government <- node_new("util",
                          type = "SCES",
                          es = 0.5, alpha = 1,
                          beta = c(0.5, 0.5),
                          "manu", "serv"
)

dstl <- list(dst.manu, dst.serv, dst.household, dst.government)

ge <- sdm2(
  A = dstl,
  B = diag(c(1, 1, 0, 0), 5, 4),
  S0Exg = matrix(c(
    NA, NA, NA, NA,
    NA, NA, NA, NA,
    NA, NA, 360 * (1 - income.tax.rate), 360 * income.tax.rate,
    NA, NA, 240 * (1 - income.tax.rate), 240 * income.tax.rate,
    NA, NA, NA, 120
  ), 5, 4, TRUE),
  names.commodity = c("manu", "serv", "lab", "cap", "vat"),
  names.agent = c("manu", "serv", "household", "government"),
  numeraire = "lab"
)

ge$p
ge$z
addmargins(ge$DV)
addmargins(ge$SV)

## VAT rate reduction
dst.manu$rate <- dst.serv$rate <- c(5 / 6, vat.rate = 0.1)

ge.new <- sdm2(
  A = dstl,
  B = diag(c(1, 1, 0, 0), 5, 4),
  S0Exg = matrix(c(
    NA, NA, NA, NA,
    NA, NA, NA, NA,
    NA, NA, 360 * (1 - income.tax.rate), 360 * income.tax.rate,
    NA, NA, 240 * (1 - income.tax.rate), 240 * income.tax.rate,
    NA, NA, NA, 120
  ), 5, 4, TRUE),
  names.commodity = c("manu", "serv", "lab", "cap", "vat"),
  names.agent = c("manu", "serv", "household", "government"),
  numeraire = "lab"
)

ge.new$p
ge.new$z
addmargins(ge.new$DV)
```

```
addmargins(ge.new$SV)
```

---

gemTemporaryEquilibriumPath

Some Examples of Temporary Equilibrium Paths

Description

Some examples of temporary equilibrium paths with sticky decisions of a firm, that is, the firm adjusts its technology sluggishly when the prices change.

Let's make a simple distinction between market clearing paths, instantaneous equilibrium paths, and temporary equilibrium paths. Under the assumption of (complete) rationality, economic agents will make decisions that are most beneficial to them based on the information they have. If the information does not change, then the decision will not change. The market clearing path (with finite or infinite length) under the assumption of complete rationality is the instantaneous equilibrium path.

However, under the assumption of bounded rationality, the decisions made by economic agents may not be optimal. They may follow some simple rules-of-thumb and may just adjust the previous decisions sluggishly according to the changes in information even though they can adjust flexibly, so that the new decisions are better than the old ones under the new information. Hence the current decision is not necessarily the optimal decision. Even if the information does not change, it is still possible for agents to make further improvements to this decision in the next period. It can also be said that in this case, the decision maker's decision is sticky, that is, it only makes limited improvements to the previous decision based on new information, rather than directly adjusting to the optimal decision. A temporary equilibrium path is a market clearing path (with finite or infinite length) in which the economic agents can have complete rationality or bounded rationality. The instantaneous equilibrium path is a special case of the the temporary equilibrium path.

Usage

```
gemTemporaryEquilibriumPath(...)
```

Arguments

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

References

Grandmont, J.M. (1977). Temporary General Equilibrium Theory. Econometrica 45, 535-572.

See Also

policyMarketClearingPrice

Examples

```
f <- function(stickiness.firm = 0) {
  dst.firm <- node_new("output",
    type = "Leontief", a = c(1 - stickiness.firm, stickiness.firm),
    "cc1", "cc2"
  )
  node_set(dst.firm, "cc1",
    type = "CD", alpha = 5,
    beta = c(0.5, 0.5),
    "prod", "lab"
  )
  node_set(dst.firm, "cc2",
    type = "CD", alpha = 5,
    beta = c(0.5, 0.5),
    "prod", "lab"
  )

  dst.consumer <- node_new("utility",
    type = "CD", alpha = 1,
    beta = c(0.5, 0.5),
    "prod", "lab"
  )

  ge <- sdm2(
    A = list(dst.firm, dst.consumer),
    B = diag(c(1, 0)),
    S0Exg = {
      S0Exg <- matrix(NA, 2, 2)
      S0Exg[2, 2] <- 100
      S0Exg
    },
    names.commodity = c("prod", "lab"),
    names.agent = c("firm", "consumer"),
    numeraire = "lab",
    maxIteration = 1,
    numberOfPeriods = 20,
    policy = list(
      function(time, A, state) {
        if (time > 1) {
          node_set(A[[1]], "cc2",
            type = "Leontief", a = state$last.A[, 1]
          )
        }
      },
      policyMarketClearingPrice
    ),
    ts = TRUE
  )

  print(ge$p)
  print(ge$z)
```

```
  par(mfrow = c(1, 2))
  matplot(ge$ts.p, type = "l")
  matplot(ge$ts.z, type = "l")
}

f()
f(stickiness.firm = 0.8)
```

---

gemTwoCountryForeignExchangeRate_6_6
Example 7.6 (Foreign Exchange Rate) in Li (2019)

Description

This is Example 7.6 in Li (2019), which illustrates foreign exchange rates.

Usage

```
gemTwoCountryForeignExchangeRate_6_6(...)
```

Arguments

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

Examples

```
dst.firm1 <- node_new("output",
  type = "FIN", rate = c(1, interest.rate = 0.1),
  "cc1", "money1"
)
node_set(dst.firm1, "cc1",
  type = "CD", alpha = 1,
  beta = c(0.5, 0.5),
  "iron", "lab1"
)

dst.firm2 <- Clone(dst.firm1)
node_replace(dst.firm2, "money1", "money2")
node_replace(dst.firm2, "lab1", "lab2")
node_plot(dst.firm2)

dst.laborer1 <- node_new("util",
  type = "FIN", rate = c(1, interest.rate = 0.1),
  "cc1", "money1"
)
node_set(dst.laborer1, "cc1",
  type = "Leontief", a = 1,
```

```
    "wheat"
  )

  dst.money.owner1 <- Clone(dst.laborer1)

  dst.laborer2 <- Clone(dst.laborer1)
  node_replace(dst.laborer2, "money1", "money2")

  dst.money.owner2 <- Clone(dst.laborer2)

  ge <- sdm2(
    A = list(
      dst.firm1, dst.laborer1, dst.money.owner1,
      dst.firm2, dst.laborer2, dst.money.owner2
    ),
    B = diag(c(1, 0, 0, 1, 0, 0)),
    S0Exg = {
      tmp <- matrix(NA, 6, 6)
      tmp[2, 2] <- 100
      tmp[3, 3] <- 600
      tmp[5, 5] <- 100
      tmp[6, 6] <- 100
      tmp
    },
    names.commodity = c(
      "wheat", "lab1", "money1",
      "iron", "lab2", "money2"
    ),
    names.agent = c(
      "firm1", "laborer1", "money.owner1",
      "firm2", "laborer2", "money.owner2"
    ),
    numeraire = c("money1" = 0.1) # interest.rate
  )

  ge$p[6] / ge$p[3] # foreign exchange rate
```

---

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 <- sdm2(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 <- sdm2(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 <- sdm2(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 <- sdm2(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 <- sdm2(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 <- sdm2(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 <- sdm2(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 <- sdm2(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"]
```

---

gemTwoCountryPureExchange_Bond

Some Examples of Two-Country Pure Exchange Economy with Bond

Description

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

Usage

```
gemTwoCountryPureExchange_Bond(...)
```

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 <- 1
# the amount of outbound investment corresponding to each unit of consumption
outbound.investment.rate <- 0.1

dst.consumption <- node_new("consumption",
```

```
  type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = es.DFProd,
  "prod.CHN", "prod.USA"
)
node_set(dst.consumption, "prod.CHN",
  type = "Leontief", a = 1 / technology.level.CHN,
  "lab.CHN"
)
node_set(dst.consumption, "prod.USA",
  type = "Leontief", a = 1,
  "lab.USA"
)
dst.CHN <- node_new("CHN",
  type = "FIN", rate = c(1, outbound.investment.rate),
  dst.consumption, "bond.USA"
)
node_plot(dst.CHN)

dst.USA <- Clone(dst.consumption)

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

ge <- sdm2(dstl,
  names.commodity = c("lab.CHN", "lab.USA", "bond.USA"),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 3, 2, TRUE),
  S0Exg = matrix(c(
    100, 0,
    0, 100,
    0, 100
  ), 3, 2, TRUE),
  numeraire = "lab.CHN"
)
ge$p[2]

#### Add currencies to the example above.
es.DFProd <- 0.8
technology.level.CHN <- 1
outbound.investment.rate <- 0.1
interest.rate <- 1e-4

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("CHN",
  type = "FIN",
  rate = c(
    1, outbound.investment.rate,
    outbound.investment.rate * interest.rate
  ),
  "consumption", "bond.USA", "money.USA"
)
node_set(dst.CHN, "consumption",
  type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = es.DFProd,
  prod_money.CHN, prod_money.USA
)

node_plot(dst.CHN)

dst.USA <- Clone(node_set(dst.CHN, "consumption"))
node_plot(dst.USA)

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

ge <- sdm2(dstl,
  names.commodity = c(
    "lab.CHN", "money.CHN",
    "lab.USA", "money.USA",
    "bond.USA"
  ),
  names.agent = c("CHN", "USA"),
  B = matrix(0, 5, 2, TRUE),
  S0Exg = matrix(c(
    100, 0,
    100, 0,
    0, 100,
    0, 100,
    0, 100
  ), 5, 2, TRUE),
  numeraire = c("money.CHN" = interest.rate)
)

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

---

gemTwoCountry_Bond_7_4 *An Example of Two-Country Economy with Bond*

Description

A general equilibrium example of two-country economy with bond.

Usage

```
gemTwoCountry_Bond_7_4(...)
```

Arguments

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

Value

A general equilibrium.

See Also

gemTwoCountry_Tariff_9_5

Examples

```
es.DFProd <- 0.8 # substitution elasticity between domestic and foreign products
es.CL <- 0.8 # substitution elasticity between capital and labor

dst.firm.CHN <- node_new("output",
  type = "SCES", alpha = 1, beta = c(0.78, 0.22), es = es.CL,
  "lab.CHN", "cap.CHN"
)

dst.household.CHN <- node_new("util",
  type = "FIN", rate = c(1, 0.028),
  "cc1", "bond.ROW"
) # 0.1 is the amount of outbound investment corresponding to
# each unit of cc1 (i.e. composite commodity 1).

node_set(dst.household.CHN, "cc1",
  type = "SCES", alpha = 1, beta = c(0.93, 0.07), es = es.DFProd,
  "prod.CHN", "prod.ROW"
)

node_plot(dst.household.CHN)

dst.firm.ROW <- node_new("output",
  type = "SCES", alpha = 1, beta = c(0.75, 0.25), es = es.CL,
  "lab.ROW", "cap.ROW"
)

dst.household.ROW <- node_new("util",
  type = "SCES", alpha = 1, beta = c(0.02, 0.98), es = es.DFProd,
  "prod.CHN", "prod.ROW"
)
```

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

ge <- sdm2(dstl,
  names.commodity = c(
    "prod.CHN", "lab.CHN", "cap.CHN",
    "prod.ROW", "lab.ROW", "cap.ROW", "bond.ROW"
  ),
  names.agent = c(
    "firm.CHN", "household.CHN",
    "firm.ROW", "household.ROW"
  ),
  B = {
    tmp <- matrix(0, 7, 4, TRUE)
    tmp[1, 1] <- tmp[4, 3] <- 1
    tmp
  },
  S0Exg = {
    tmp <- matrix(NA, 7, 4, TRUE)
    tmp[2, 2] <- 53 # the supply of lab.CHN
    tmp[3, 2] <- 15 # the supply of cap.CHN
    tmp[5, 4] <- 240 # the supply of lab.ROW
    tmp[6, 4] <- 77 # the supply of cap.ROW
    tmp[7, 4] <- 2 # the supply of bond.ROW
    tmp
  },
  numeraire = "lab.CHN"
)

ge$p
ge$z

# Determine the parameters in the
# example based on an input-output table.
IT <- matrix(c(
  0, 61.44, 0, 6.498,
  53, 0, 0, 0,
  14.94, 0, 0, 0,
  0, 4.647, 0, 320,
  0, 0, 242.9, 0,
  0, 0, 81.74, 0,
  0, 1.85, 0, 0
), 7, 4, TRUE)

OT <- matrix(c(
  67.94, 0, 0, 0,
  0, 53, 0, 0,
  0, 14.94, 0, 0,
  0, 0, 324.64, 0,
  0, 0, 0, 242.9,
  0, 0, 0, 81.74,
  0, 0, 0, 1.85
), 7, 4, TRUE)
```

```
dimnames(IT) <- dimnames(OT) <- list(
  c("prod.CHN", "lab.CHN", "cap.CHN", "prod.ROW", "lab.ROW", "cap.ROW", "bond.ROW"),
  c("firm.CHN", "household.CHN", "firm.ROW", "household.ROW")
)

es.DFProd <- 0.8 # substitution elasticity between domestic and foreign products
es.CL <- 0.8 # substitution elasticity between capital and labor

dst.firm.CHN <- node_new("output",
                         type = "SCES",
                         alpha = OT["prod.CHN", "firm.CHN"] /
                           sum(IT[c("lab.CHN", "cap.CHN"), "firm.CHN"]),
                         beta = prop.table(IT[c("lab.CHN", "cap.CHN"), "firm.CHN"]),
                         es = es.CL,
                         "lab.CHN", "cap.CHN"
)

# the amount of outbound investment corresponding to
# each unit of composite commodity 1 used by household.
outbound.investment.rate <- IT["bond.ROW", "household.CHN"] /
  sum(IT[c("prod.CHN", "prod.ROW"), "household.CHN"])

dst.household.CHN <- node_new("util",
                             type = "FIN",
                             rate = c(1, outbound.investment.rate),
                             "cc1", "bond.ROW"
)

node_set(dst.household.CHN, "cc1",
         type = "SCES", alpha = 1,
         beta = prop.table(IT[c("prod.CHN", "prod.ROW"), "household.CHN"]),
         es = es.DFProd,
         "prod.CHN", "prod.ROW"
)

dst.firm.ROW <- node_new("output",
                         type = "SCES", alpha = 1,
                         beta = prop.table(IT[c("lab.ROW", "cap.ROW"), "firm.ROW"]),
                         es = es.CL,
                         "lab.ROW", "cap.ROW"
)

dst.household.ROW <- node_new("util",
                             type = "SCES", alpha = 1,
                       beta = prop.table(IT[c("prod.CHN", "prod.ROW"), "household.ROW"]),
                             es = es.DFProd,
                             "prod.CHN", "prod.ROW"
)

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

ge <- sdm2(dstl,
```

```
    names.commodity = c(
      "prod.CHN", "lab.CHN", "cap.CHN",
      "prod.ROW", "lab.ROW", "cap.ROW", "bond.ROW"
    ),
    names.agent = c(
      "firm.CHN", "household.CHN",
      "firm.ROW", "household.ROW"
    ),
    B = {
      tmp <- matrix(0, 7, 4, TRUE)
      tmp[1, 1] <- tmp[4, 3] <- 1
      tmp
    },
    S0Exg = {
      tmp <- matrix(NA, 7, 4, TRUE)
      tmp[2, 2] <- OT["lab.CHN", "household.CHN"]
      tmp[3, 2] <- OT["cap.CHN", "household.CHN"]
      tmp[5, 4] <- OT["lab.ROW", "household.ROW"]
      tmp[6, 4] <- OT["cap.ROW", "household.ROW"]
      tmp[7, 4] <- OT["bond.ROW", "household.ROW"]
      tmp
    },
    numeraire = "lab.CHN"
  )

  ge$p
  ge$z
```

---

gemTwoCountry_RealExchangeRateIndex_7_4
Calculating Real Exchange Rate Index

Description

Some examples of calculating the real exchange rate index in a two-country economy.

Usage

```
gemTwoCountry_RealExchangeRateIndex_7_4(...)
```

Arguments

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

Value

A real exchange rate index.

See Also

gemTwoCountry_Bond_7_4

Examples

```
# es.DFProd is the substitution elasticity between domestic and foreign products.
makeDstl <- function(es.DFProd = 0.5,
                     alpha.firm.CHN = 1,
                     beta.household.CHN = c(0.75, 0.25),
                     outbound.investment.rate = 0.25) {
  es.CL <- 0.8 # substitution elasticity between capital and labor

  dst.firm.CHN <- node_new("output",
    type = "SCES", alpha = alpha.firm.CHN, beta = c(0.75, 0.25), es = es.CL,
    "lab.CHN", "cap.CHN"
  )

  dst.household.CHN <- node_new("util",
    type = "FIN", rate = c(1, outbound.investment.rate),
    "cc1", "bond.ROW"
  ) # 0.1 is the amount of foreign investment corresponding to
  # each unit of cc1 (i.e. composite commodity 1).

  node_set(dst.household.CHN, "cc1",
    type = "SCES", alpha = 1, beta = beta.household.CHN, es = es.DFProd,
    "prod.CHN", "prod.ROW"
  )

  node_plot(dst.household.CHN)

  dst.firm.ROW <- node_new("output",
    type = "SCES", alpha = 1, beta = c(0.5, 0.5), es = es.CL,
    "lab.ROW", "cap.ROW"
  )

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

  list(dst.firm.CHN, dst.household.CHN, dst.firm.ROW, dst.household.ROW)
}

dstl <- makeDstl()

SExg <- {
  tmp <- matrix(NA, 7, 4, TRUE)
  tmp[2, 2] <- 30 # the supply of lab.CHN
  tmp[3, 2] <- 10 # the supply of cap.CHN
  tmp[5, 4] <- 156 # the supply of lab.ROW
  tmp[6, 4] <- 156 # the supply of cap.ROW
```

```
  tmp[7, 4] <- 8 # the supply of bond.ROW
  tmp
}

f <- function(A = dstl,
              S0Exg = SExg) {
  sdm2(
    A = A,
    names.commodity = c(
      "prod.CHN", "lab.CHN", "cap.CHN",
      "prod.ROW", "lab.ROW", "cap.ROW", "bond.ROW"
    ),
    names.agent = c(
      "firm.CHN", "household.CHN",
      "firm.ROW", "household.ROW"
    ),
    B = {
      tmp <- matrix(0, 7, 4, TRUE)
      tmp[1, 1] <- tmp[4, 3] <- 1
      tmp
    },
    S0Exg = S0Exg,
    numeraire = "lab.CHN"
  )
}

ge.benchmark <- f()

## real exchange rate index
reri <- function(ge.new, ge.benchmark) {
  weight.CHN <- rowSums(ge.benchmark$SV)[c("prod.CHN", "lab.CHN", "cap.CHN")]
  weight.ROW <- rowSums(ge.benchmark$SV)[c("prod.ROW", "lab.ROW", "cap.ROW")]

  weighted.mean(ge.new$p[c("prod.ROW", "lab.ROW", "cap.ROW")], weight.ROW) /
    weighted.mean(ge.new$p[c("prod.CHN", "lab.CHN", "cap.CHN")], weight.CHN)
}


## technology progress in CHN
reri(f(A = makeDstl(es.DFProd = 5, alpha.firm.CHN = 2)), ge.benchmark)
reri(f(A = makeDstl(es.DFProd = 0.5, alpha.firm.CHN = 2)), ge.benchmark)

## labor supply change in CHN
SExg.LSC <- SExg
SExg.LSC[2, 2] <- SExg.LSC[2, 2] * 2

reri(f(A = makeDstl(es.DFProd = 5), S0Exg = SExg.LSC), ge.benchmark)

reri(f(S0Exg = SExg.LSC), ge.benchmark)

## capital accumulation in CHN
SExg.CA <- SExg
SExg.CA[3, 2] <- SExg.CA[3, 2] * 3
```

```
reri(f(A = makeDstl(es.DFProd = 5), S0Exg = SExg.CA), ge.benchmark)

reri(f(S0Exg = SExg.CA), ge.benchmark)

## preference change in China
reri(f(A = makeDstl(es.DFProd = 5, beta.household.CHN = c(0.5, 0.5))), ge.benchmark)
reri(f(A = makeDstl(beta.household.CHN = c(0.5, 0.5))), ge.benchmark)

## outbound-investment-rate change in China
reri(f(A = makeDstl(es.DFProd = 5, outbound.investment.rate = 0.1)), ge.benchmark)
reri(f(A = makeDstl(outbound.investment.rate = 0.1)), ge.benchmark)
```

---

gemTwoCountry_Tariff_9_5 *An Example of Two-Country Economy with Tariff*

Description

A general equilibrium example of two-country economy with tariff.

Usage

```
gemTwoCountry_Tariff_9_5(...)
```

Arguments

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

Value

A general equilibrium.

See Also

gemTwoCountry_Bond_7_4

Examples

```
es.DFProd <- 0.8 # substitution elasticity between domestic and foreign products
es.CL <- 0.8 # substitution elasticity between capital and labor

dst.firm.CHN <- node_new("output",
  type = "SCES", alpha = 1, beta = c(0.78, 0.22), es = es.CL,
  "lab.CHN", "cap.CHN"
)

dst.household.CHN <- node_new("util",
```

```
  type = "FIN", rate = c(1, outbound.investment.rate = 0.028),
  "cc1", "bond.ROW"
)

node_set(dst.household.CHN, "cc1",
  type = "SCES", alpha = 1, beta = c(0.93, 0.07), es = es.DFProd,
  "prod.CHN", "imported.prod.CHN"
)

node_plot(dst.household.CHN)

dst.foreign.trade.CHN <- node_new("imported.product",
  type = "FIN",
  rate = c(1, 0.016),
  "prod.ROW", "tariff.CHN"
)

dst.firm.ROW <- node_new("output",
  type = "SCES", alpha = 1, beta = c(0.75, 0.25), es = es.CL,
  "lab.ROW", "cap.ROW"
)

dst.household.ROW <- node_new("util",
  type = "SCES", alpha = 1, beta = c(0.02, 0.98), es = es.DFProd,
  "prod.CHN", "prod.ROW"
)

dstl <- list(
  dst.firm.CHN, dst.household.CHN, dst.foreign.trade.CHN,
  dst.firm.ROW, dst.household.ROW
)

ge <- sdm2(dstl,
  names.commodity = c(
    "prod.CHN", "lab.CHN", "cap.CHN", "imported.prod.CHN", "tariff.CHN",
    "prod.ROW", "lab.ROW", "cap.ROW", "bond.ROW"
  ),
  names.agent = c(
    "firm.CHN", "household.CHN", "foreign.trade.CHN",
    "firm.ROW", "household.ROW"
  ),
  B = {
    tmp <- matrix(0, 9, 5, TRUE)
    tmp[1, 1] <- tmp[6, 4] <- 1
    tmp[4, 3] <- 1
    tmp
  },
  S0Exg = {
    tmp <- matrix(NA, 9, 5, TRUE)
    tmp[2, 2] <- 53 # the supply of lab.CHN
    tmp[3, 2] <- 15 # the supply of cap.CHN
    tmp[5, 2] <- 0.29 # the supply of tariff.CHN
    tmp[7, 5] <- 240 # the supply of lab.ROW
```

```
    tmp[8, 5] <- 77 # the supply of cap.ROW
    tmp[9, 5] <- 2 # the supply of bond.ROW
    tmp
  },
  numeraire = "lab.CHN"
)

ge$p
ge$z
```

---

gemTwoIndustries_4_3 *A 4-by-3 Economy with Two Industries*

Description

A 4-by-3 economy with two industries.

Usage

```
gemTwoIndustries_4_3(...)
```

Arguments

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

References

Ju, Jiandong, Justin Yifu Lin, Qing Liu, Kang Shi (2020) Structural Changes and the Real Exchange Rate Dynamics. Journal of International Money and Finance, Vol. 107, pp: 102192.

Examples

```
dst.manu <- node_new("output",
                     type = "SCES", es = 1, alpha = 1,
                     beta = c(0.2, 0.8),
                     "lab", "cap"
)

dst.serv <- node_new("output",
                     type = "SCES", es = 1, alpha = 1,
                     beta = c(0.8, 0.2),
                     "lab", "cap"
)

dst.consumer <- node_new("util",
                         type = "SCES", es = 1, alpha = 1,
                         beta = c(0.5, 0.5),
                         "manu", "serv"
```

```
)

dstl <- list(dst.manu, dst.serv, dst.consumer)

S0Exg <- matrix(NA, 4, 3)
S0Exg[3:4, 3] <- c(100, 100)

f <- function(dstl, S0Exg) {
  sdm2(
    A = dstl,
    B = matrix(c(
      1, 0, 0,
      0, 1, 0,
      0, 0, 0,
      0, 0, 0
    ), 4, 3, TRUE),
    S0Exg = S0Exg,
    names.commodity = c("manu", "serv", "lab", "cap"),
    names.agent = c("manu", "serv", "consumer"),
    numeraire = c("manu")
  )
}

ge <- f(dstl = dstl, S0Exg = S0Exg)

ge$D
ge$p

##
dstl2 <- lapply(dstl, Clone)
dstl2[[1]]$alpha <- 2

ge <- f(dstl = dstl2, S0Exg = S0Exg)
ge$D
ge$p

##
S0Exg2 <- S0Exg
S0Exg2[3, 3] <- 200 # labor supply
ge <- f(dstl = dstl, S0Exg = S0Exg2)
ge$D
ge$p

##
S0Exg3 <- S0Exg
S0Exg3[4, 3] <- 200 # capital supply
ge <- f(dstl = dstl, S0Exg = S0Exg3)
ge$D
ge$p

##
dstl3 <- lapply(dstl, Clone)
dstl3[[3]]$beta <- c(0.2, 0.8)
```

```
ge <- f(dstl = dstl3, S0Exg = S0Exg)
ge$D
ge$p

## exogenous wage rate
S0Exg4 <- S0Exg
S0Exg4[3, 3] <- 1000 # labor supply

ge <- sdm2(
  A = dstl2,
  B = matrix(c(
    1, 0, 0,
    0, 1, 0,
    0, 0, 0,
    0, 0, 0
  ), 4, 3, TRUE),
  S0Exg = S0Exg4,
  names.commodity = c("manu", "serv", "lab", "cap"),
  names.agent = c("manu", "serv", "consumer"),
  numeraire = c("manu"),
  pExg = c(1, NA, 1, NA),
  maxIteration = 1,
  ts = TRUE
)

matplot(ge$ts.z, type = "l")
matplot(ge$ts.q, type = "l")
tail(ge$ts.q)
ge$p
```

---

gem_2_2 *Some Simple 2-by-2 General Equilibrium Models*

Description

Some simple 2-by-2 general equilibrium models with a firm and a laborer.

Usage

```
gem_2_2(...)
```

Arguments

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

References

http://www.econ.ucla.edu/riley/MAE/Course/SolvingForTheWE.pdf

Examples

```
####
ge.Leontief <- sdm2(
  A = matrix(c(
    0.5, 1,
    0.5, 0
  ), 2, 2, TRUE),
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "laborer"),
  numeraire = "prod"
)

ge.Leontief$p
ge.Leontief$z
ge.Leontief$D
ge.Leontief$S

## the same as above
ge2.Leontief <- sdm2(
  A = list(
    dst.firm = node_new(
      "output",
      type = "Leontief",
      a = c(0.5, 0.5),
      "prod", "lab"
    ),
    dst.consumer = node_new(
      "util",
      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),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "laborer"),
  numeraire = "prod"
```

```
)

ge2.Leontief$p
ge2.Leontief$z
ge2.Leontief$D
ge2.Leontief$S

####
ge.CD <- sdm2(
  A = function(state) {
    ## the vector of demand coefficients of the firm
    a1 <- CD_A(alpha = 2, Beta = c(0.5, 0.5), state$p)
    ## the vector of demand coefficients of the laborer
    a2 <- c(1, 0)
    cbind(a1, a2)
  },
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "laborer"),
  numeraire = "prod"
)

ge.CD$p
ge.CD$z
ge.CD$D
ge.CD$S

## the same as above
ge2.CD <- sdm2(
  A = list(
    dst.firm = node_new(
      "output",
      type = "CD", alpha = 2, beta = c(0.5, 0.5),
      "prod", "lab"
    ),
    dst.consumer = node_new(
      "util",
      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),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "laborer"),
  numeraire = "prod"
)

ge2.CD$p
ge2.CD$z
ge2.CD$D
ge2.CD$S

####
ge.SCES <- sdm2(
  A = function(state) {
    a1 <- SCES_A(es = 0.5, alpha = 1, Beta = c(0.5, 0.5), p = state$p)
    a2 <- c(1, 0)
    cbind(a1, a2)
  },
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "laborer"),
  numeraire = "prod"
)

ge.SCES$p
ge.SCES$z
ge.SCES$D
ge.SCES$S

## the same as above
ge2.SCES <- sdm2(
  A = list(
    dst.firm = node_new(
      "output",
      type = "SCES",
      es = 0.5, alpha = 1, beta = c(0.5, 0.5),
      "prod", "lab"
    ),
    dst.consumer = node_new(
      "util",
      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),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "laborer"),
  numeraire = "prod"
)

ge2.SCES$p
ge2.SCES$z
ge2.SCES$D
ge2.SCES$S

#### Example 1 in the ucla reference
ge3.SCES <- sdm2(
  A = function(state) {
    a.firm <- c(0, 0.25)
    a.consumer <- SCES_A(es = 0.5, alpha = 1, Beta = c(0.5, 0.5), p = state$p)
    cbind(a.firm, a.consumer)
  },
  B = matrix(c(
    1, 0,
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 30
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "laborer"),
  numeraire = "prod"
)

ge3.SCES$p
ge3.SCES$z
ge3.SCES$D
ge3.SCES$S
```

gem_3_2 *Some Simple 3-by-2 General Equilibrium Models*

Description

Some simple 3-by-2 general equilibrium models with a firm and a consumer.

Usage

```
gem_3_2(...)
```

Arguments

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

References

http://www.econ.ucla.edu/riley/MAE/Course/SolvingForTheWE.pdf

Examples

```
ge.CD <- sdm2(
  A = function(state) {
    ## the vector of demand coefficients of the firm
    a1 <- CD_A(alpha = 2, Beta = c(0, 0.5, 0.5), state$p)
    ## the vector of demand coefficients of the consumer
    a2 <- c(1, 0, 0)
    cbind(a1, a2)
  },
  B = matrix(c(
    1, 0,
    0, 0,
    0, 0
  ), 3, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100,
    NA, 100
  ), 3, 2, TRUE),
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "prod"
)

ge.CD$p
ge.CD$z
ge.CD$D
ge.CD$S

#### Example 2 in the ucla reference
## By introducing a new factor of production (called land here)
## a firm with diminishing returns to scale can be converted into
## a firm with constant returns to scale.
ge2.CD <- sdm2(
  A = function(state) {
    a.firm <- CD_A(alpha = 6, Beta = c(0, 0.5, 0.5), state$p)
    a.consumer <- CD_A(alpha = 1, Beta = c(0.2, 0.8, 0), state$p)
    cbind(a.firm, a.consumer)
  },
```

```
  B = matrix(c(
    1, 0,
    0, 0,
    0, 0
  ), 3, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 81,
    NA, 1
  ), 3, 2, TRUE),
  names.commodity = c("prod", "lab", "land"),
  names.agent = c("firm", "consumer"),
  numeraire = "prod"
)

ge2.CD$p
ge2.CD$z
ge2.CD$D
ge2.CD$S

####
ge.SCES <- sdm2(
  A = function(state) {
    a1 <- SCES_A(es = 0.5, alpha = 1, Beta = c(0, 0.5, 0.5), p = state$p)
    a2 <- c(1, 0, 0)
    cbind(a1, a2)
  },
  B = matrix(c(
    1, 0,
    0, 0,
    0, 0
  ), 3, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100,
    NA, 100
  ), 3, 2, TRUE),
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "prod"
)

ge.SCES$p
ge.SCES$z
ge.SCES$D
ge.SCES$S

####
ge2.SCES <- sdm2(
  A = function(state) {
    a1 <- SCES_A(es = 0.5, alpha = 1, Beta = c(0.2, 0.4, 0.4), p = state$p)
    a2 <- c(1, 0, 0)
    cbind(a1, a2)
```

```
  },
  B = matrix(c(
    1, 0,
    0, 0,
    0, 0
  ), 3, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100,
    NA, 100
  ), 3, 2, TRUE),
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "prod"
)

ge2.SCES$p
ge2.SCES$z
ge2.SCES$D
ge2.SCES$S

#### nested production function
dst.firm <- node_new(
  "prod",
  type = "Leontief",
  a = c(0.2, 0.8),
  "prod", "cc1"
)
node_set(dst.firm, "cc1",
         type = "SCES",
         es = 0.5, alpha = 1, beta = c(0.5, 0.5),
         "cap", "lab"
)

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

ge3.SCES <- sdm2(
  A = list(dst.firm, dst.consumer),
  B = matrix(c(
    1, 0,
    0, 0,
    0, 0
  ), 3, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 100,
    NA, 100
  ), 3, 2, TRUE),
  names.commodity = c("prod", "cap", "lab"),
```

```
  names.agent = c("firm", "consumer"),
  numeraire = "prod"
)

ge3.SCES$p
ge3.SCES$z
ge3.SCES$D
ge3.SCES$S
```

---

gem_3_3 *Some Simple 3-by-3 General Equilibrium Models*

Description

Some simple 3-by-3 general equilibrium models with two firms and a consumer.

Usage

```
gem_3_3(...)
```

Arguments

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

See Also

gemCapitalAccumulation

Examples

```
####
ge <- sdm2(
  A = function(state) {
    ## the vector of demand coefficients of the firm
    a.firm.corn <- CD_A(alpha = 1, Beta = c(0, 0.5, 0.5), p = state$p)
    a.firm.iron <- CD_A(alpha = 2, Beta = c(0, 0.5, 0.5), p = state$p)
    ## the vector of demand coefficients of the consumer
    a.consumer <- CD_A(alpha = 1, Beta = c(0.5, 0.5, 0), p = state$p)
    cbind(a.firm.corn, a.firm.iron, a.consumer)
  },
  B = diag(c(1, 1), 3),
  S0Exg = {
    tmp <- matrix(NA, 3, 3)
    tmp[3, 3] <- 100
    tmp
  },
  names.commodity = c("corn", "iron", "lab"),
  names.agent = c("firm.corn", "firm.iron", "consumer"),
```

```
  numeraire = "lab"
)

ge$p
ge$z
ge$D
ge$S

####
ge <- sdm2(
  A = function(state) {
    ## the vector of demand coefficients of the firm
    a.firm.corn <-
      SCES_A(es = 1,
             alpha = 1,
             Beta = c(0, 0.5, 0.5),
             p = state$p)
    a.firm.iron <-
      SCES_A(es = 1,
             alpha = 2,
             Beta = c(0, 0.5, 0.5),
             p = state$p)
    ## the vector of demand coefficients of the consumer
    a.consumer <- CD_A(alpha = 1, Beta = c(0.5, 0.5, 0), p = state$p)
    cbind(a.firm.corn, a.firm.iron, a.consumer)
  },
  B = diag(c(1, 1), 3),
  S0Exg = {
    tmp <- matrix(NA, 3, 3)
    tmp[3, 3] <- 100
    tmp
  },
  names.commodity = c("corn", "iron", "lab"),
  names.agent = c("firm.corn", "firm.iron", "consumer"),
  numeraire = "lab"
)

ge$p
ge$z
ge$D
ge$S

#### a general equilibrium model containing a production firm
#### and a capital-goods-leasing firm
ge <- sdm2(
  A = function(state) {
    a.firm1 <- CD_A(alpha = 2, Beta = c(0, 0.5, 0.5), state$p)
    a.consumer <- c(1, 0, 0)
    a.firm2 <- c(1, 0, 0)
    cbind(a.firm1, a.consumer, a.firm2)
  },
  B = matrix(c(
    1, 0, 0.5,
```

```
    0, 0, 1,
    0, 0, 0
  ), 3, 3, TRUE),
  S0Exg = matrix(c(
    NA, NA, NA,
    NA, NA, NA,
    NA, 100,NA
  ), 3, 3, TRUE),
  names.commodity = c("prod", "cap", "lab"),
  names.agent = c("firm1", "consumer","firm2"),
  numeraire = "prod",
  priceAdjustmentVelocity = 0.05
)
ge$p
ge$z
ge$D
```

---

gem_3_4 *Some Simple 3-by-4 General Equilibrium Models*

Description

Some simple 3-by-4 general equilibrium models with two firms and two consumer.

Usage

```
gem_3_4(...)
```

Arguments

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

Examples

```
####
ge <- sdm2(
  A = function(state) {
    a.firm.corn <- CD_A(alpha = 1, Beta = c(0, 0.5, 0.5), state$p)
    a.firm.iron <- CD_A(alpha = 2, Beta = c(0, 0.5, 0.5), state$p)
    a.consumer1 <- c(1, 0, 0)
    a.consumer2 <- CD_A(alpha = 1, Beta = c(0.5, 0.5, 0), state$p)

    cbind(a.firm.corn, a.firm.iron, a.consumer1, a.consumer2)
  },
  B = diag(c(1, 1, 0), 3, 4),
  S0Exg = {
    tmp <- matrix(NA, 3, 4)
    tmp[3, 3:4] <- 100
```

```
    tmp
  },
  names.commodity = c("corn", "iron", "lab"),
  names.agent = c("firm.corn", "firm.iron", "consumer1", "consumer2"),
  numeraire = "lab"
)

ge$p
ge$z
ge$D
ge$S

####
ge <- sdm2(
  A = function(state) {
    a.firm.corn <-
      SCES_A(
        es = 1,
        alpha = 1,
        Beta = c(0, 0.5, 0.5),
        p = state$p
      )
    a.firm.iron <-
      SCES_A(
        es = 1,
        alpha = 2,
        Beta = c(0, 0.5, 0.5),
        p = state$p
      )
    a.consumer1 <- c(1, 0, 0)
    a.consumer2 <- CD_A(alpha = 1, Beta = c(0.5, 0.5, 0), state$p)

    cbind(a.firm.corn, a.firm.iron, a.consumer1, a.consumer2)
  },
  B = diag(c(1, 1, 0), 3, 4),
  S0Exg = {
    tmp <- matrix(NA, 3, 4)
    tmp[3, 3:4] <- 100
    tmp
  },
  names.commodity = c("corn", "iron", "lab"),
  names.agent = c("firm.corn", "firm.iron", "consumer1", "consumer2"),
  numeraire = "lab"
)

ge$p
ge$z
ge$D
ge$S
```

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 = TRUE)
```

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)
```

`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)

#### Heron's method for finding square roots
x <- 1
f <- function(x, n) (x + n / x) / 2
iterate(x, f, 10, n = 5)

#### Find a root of the equation x^3-x-1==0.
x <- 1.5
f <- function(x) (x + 1)^(1 / 3)
```

```
iterate(x, f, 10)

####
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)
```

`makeCountercyclicalProductTax`
Make a Countercyclical Product Tax Policy Function

Description

This function returns a countercyclical product tax policy function to accelerate convergence when calculating general equilibrium. In some cases this tax policy with variable tax rates can stabilize the economy (see Li, 2019, section 9.4.5.4) . When a firm's output is higher than the average output in previous periods, a tax is imposed on the firm to reduce the output of the product. Tax revenue will be used for implicit public spending. The way of taxation is to directly deduct a part of the supply of the firm.

Usage

```
makeCountercyclicalProductTax(agent = 1, time.win = c(100, Inf), span = 50)
```

Arguments

`agent`	a vector specifying the indices or names of firms to be taxed.
`time.win`	the time window vector, i.e. a 2-vector specifying the start time and end time of policy implementation.
`span`	a positive integer which indicates the number of periods when calculating the average output.

Value

A countercyclical product tax policy function.

See Also

CGE::Example9.10.policy.tax

Examples

```
ge <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = 1000
)

ge <- gemCanonicalDynamicMacroeconomic_4_3(
  numberOfPeriods = 1000,
  policy = makeCountercyclicalProductTax(time.win = c(500, Inf))
)
```

---

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)
  )
)
```

---

makePolicyMeanValue *Make a Mean Value Policy Function*

Description

This function returns a mean value policy function with a given span to accelerate convergence when calculating general equilibrium. We can observe the number of periods included in the economic cycle of the time series, and then set the number of periods as the parameter (i.e. span) of this function. See policyMeanValue

Usage

```
makePolicyMeanValue(span = 200)
```

Arguments

span	a positive integer. When the time index is an integer multiple of span, the mean value policy sets the current prices and supplies to the averages of the previous span-1 periods.

Value

A mean value policy function.

See Also

policyMeanValue gemDualLinearProgramming.

Examples

```
## See the function gemDualLinearProgramming.
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
  }

  ge <- sdm2(
    A = A, B = B, S0Exg = S0Exg,
    maxIteration = 1,
    numberOfPeriods = 1000,
    ts = TRUE
  )
  matplot(ge$ts.q, type = "l")

  ge2 <- sdm2(
    A = A, B = B, S0Exg = S0Exg,
    maxIteration = 1,
    numberOfPeriods = 1000,
    ts = TRUE,
    policy = makePolicyMeanValue(150)
  )
  matplot(ge2$ts.q, type = "l")
```

`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$ts.z, type = "o", pch = 20)
matplot(ge$ts.p, type = "o", 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$ts.z, type = "o", pch = 20)
matplot(ge$ts.p, type = "o", 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$ts.z, type = "o", pch = 20)
matplot(ge$ts.p, type = "o", pch = 20)
```

`marginal_utility` *Marginal Utility*

Description

If the argument price is null, this function computes the (delta) marginal utility. By default, delta is set to 1e-10. Otherwise this function computes the (delta) value marginal utility. For a utility function U(x), two vector x, y and a scalar price, the marginal utility is (U(x + delta * y) - U(x)) / delta, and the value marginal utility is (U(x + delta * y / price) - U(x)) / delta. For a marginal utility function M(x), three vector x, y, wt and a scalar price, the marginal utility is sum(M(x) * y * wt), and the value marginal utility is sum(M(x) * y * wt / price).

Usage

```
marginal_utility(x, y, uf, price = NULL, delta = 1e-10, muf = NULL)
```

Arguments

x	a numeric k-by-m matrix or a numeric vector (i.e. a k-by-1 matrix).

y	a numeric k-by-n matrix or a numeric vector (i.e. a k-by-1 matrix).
uf	a utility function or a list consisting of m utility functions.
price	NULL or a numeric n-vector consisting of prices of each column of y.
delta	a scalar.
muf	a marginal utility function or a list consisting of m marginal utility functions. A marginal utility function is the gradient of a utility function. When the components in the marginal utility vector are the same, a marginal utility function may calculate only one of the components.

Value

A n-by-m marginal utility matrix. Its (i,j)-th element corresponds to the i-th column of y and the j-th column of x.

References

Sharpe, William F. (2008, ISBN: 9780691138503) Investors and Markets: Portfolio Choices, Asset Prices, and Investment Advice. Princeton University Press.

Examples

```
marginal_utility(1:2, cbind(1:2, 1:1), AMV)
marginal_utility(1:2, cbind(1:2, 1:1), AMV, delta = 100)
marginal_utility(cbind(1:2, 3:4), cbind(1:2, 1:1), AMV)
marginal_utility(
  cbind(1:2, 3:4), cbind(1:2, 1:1),
  list(AMV, function(x) AMV(x, gamma = 0.5))
)

####
wt <- 1:2
uf <- function(x) (x - x^2 / 400) %*% wt
muf <- function(x) (1 - 1 / 200 * x) * wt
marginal_utility(1:2, cbind(1:2, 1:1), uf)
marginal_utility(1:2, cbind(1:2, 1:1), muf = muf)

####
marginal_utility(
  1:2, cbind(1:2, 1:1),
  function(x, gamma = 1, p = rep(1, length(x))) CRRA(x, gamma, p)$CE
)
marginal_utility(1:2, cbind(1:2, 1:1), function(x) sqrt(prod(x)))

gamma <- 0.8
wt <- c(0.25, 0.75)
marginal_utility(
  1:2, cbind(1:2, 1:1),
  function(x) CRRA(x, gamma = gamma, prob = wt)$CE
)
## the same as above. CRRA and CES utility funcitons are essentially the same.
```

```
es <- 1 / gamma
beta <- wt^es
marginal_utility(
  1:2, cbind(1:2, 1:1),
  function(x) CES(x = x, sigma = 1 - 1 / es, alpha = 1, beta = wt)
)

prop.table(marginal_utility(
  1:2, cbind(1:2, 1:1),
  function(x) CRRA(x, gamma = gamma, prob = wt)$CE
))
prop.table(marginal_utility(
  1:2, cbind(1:2, 1:1),
  function(x) CRRA(x, gamma = gamma, prob = wt)$u
))
```

---

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_aggregate *Aggregate Some Rows and Columns of a Matrix*

Description

Aggregate some rows and columns of a matrix to obtain a matrix with smaller dimensions. This function can be used for aggregating some rows and columns of an input-output table.

Usage

```
matrix_aggregate(
  M,
  row.index = NULL,
  col.index = NULL,
  row.name = NULL,
  col.name = NULL
)
```

Arguments

M	a numeric matrix without NA.
row.index	a numeric vector or a list of numeric vectors indicating the index numbers of rows to be aggregated. The default value is is NULL.
col.index	a numeric vector or a list of numeric vectors indicating the index numbers of columns to be aggregated. The default value is is NULL.
row.name	a character vector or a list of character vectors indicating the names of rows to be aggregated. The default value is NULL. If row.index or col.index is not NULL, row.name and col.name will be ignored.
col.name	a character vector or a list of character vectors indicating the names of columns to be aggregated. The default value is NULL.

Examples

```
M <- matrix(1:16,4,4,TRUE)
colnames(M) <- paste0("c",1:4)
rownames(M) <- paste0("r",1:4)
addmargins(M)

M2 <- matrix_aggregate(M, list(c(1,3),c(2, 4)), 2:3)
addmargins(M2)
```

```
M3 <- matrix_aggregate(M, row.name = list(c("r1","r3"),c("r2","r4")), col.name = c("c2","c3"))
addmargins(M3)
```

---

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]])
```

MDCES_demand *Modified Displaced CES Demand Function*

Description

Compute the modified displaced CES demand function. Firstly, the (unmodified) DCES demand vector and the (unmodified) utility level are computed under the given income and prices. Secondly, the modified beta and es are computed under the unmodified utility level. Finally, the DCES demand vector (namely the modified DCES demand vector) and the utility level (namely the modified DCES utility) are computed under the modified beta, the modified es, the given income and prices.

Usage

```
MDCES_demand(es, beta, xi, w, p, betaMod = NULL, esMod = NULL, detail = FALSE)
```

Arguments

es	the elasticity of substitution.
beta	a n-vector consisting of the marginal expenditure share coefficients (Fullerton, 1989).
xi	a n-vector. Each element of xi parameterizes whether the particular good is a necessity for the household (Acemoglu, 2009, page 152). For example, xi[i] > 0 may mean that the household needs to consume at least a certain amount of good i to survive.
w	a scalar indicating the income.
p	a n-vector indicating the prices.
betaMod	a function with the unmodified utility level u.unmod as the argument.
esMod	a function with the unmodified utility level u.unmod as the argument.
detail	If detail==FALSE, the modified demand vector is returned. If detail==TRUE, the returned vector consists of the modified demand vector, the modified utility, the modified es, the modified beta, the unmodified utility and the unmodified demand vector.

References

Acemoglu, D. (2009, ISBN: 9780691132921) Introduction to Modern Economic Growth. Princeton University Press.

Fullerton, D. (1989) Notes on Displaced CES Functional Forms. Available at: https://works.bepress.com/don_fullerton/39/

Examples

```
MDCES_demand(
  es = 1.7, beta = c(0.9, 0.1), xi = c(12, 48),
  w = 24, p = c(1, 1 / 400),
```

```
  betaMod = function(u.unmod) {
    beta2 <- min(0.1, 10 / u.unmod)
    c(1 - beta2, beta2)
  },
  detail = TRUE
)

#### An example of computing the daily
#### labor supply at various wage rates.
result <- c()

for (real.wage in 4:400) {
  x <- MDCES_demand(
    es = 1.7, beta = c(0.9, 0.1),
    xi = c(12, 48), w = 24,
    p = c(1, 1 / real.wage),
    betaMod = function(u.unmod) {
      beta2 <- min(0.1, 10 / u.unmod)
      c(1 - beta2, beta2)
    },
    detail = TRUE
  )

  lab.supply <- unname(24 - x[1])
  result <- rbind(
    result,
    c(real.wage, lab.supply, x)
  )
}

plot(result[, 1:2],
  type = "o", pch = 20,
  xlab = "hourly real wage",
  ylab = "daily labor supply"
)

#### A 2-by-2 general equilibrium model
#### with a MDCES demand function
ge <- sdm2(
  A = function(state) {
    a.firm <- CD_A(alpha = 5, Beta = c(0.5, 0.5), state$p)
    a.consumer <-
      MDCES_demand(
        es = 1, beta = c(0.5, 0.5), xi = c(0, 0), w = state$w[2], p = state$p,
        betaMod = function(u.unmod) {
          beta2 <- 0.95 * plogis(u.unmod, location = 2, scale = 2)
          c(1 - beta2, beta2)
        }
      )
    cbind(a.firm, a.consumer)
  },
  B = matrix(c(
    1, 0,
```

```
    0, 0
  ), 2, 2, TRUE),
  S0Exg = matrix(c(
    NA, NA,
    NA, 1
  ), 2, 2, TRUE),
  names.commodity = c("prod", "lab"),
  names.agent = c("firm", "consumer"),
  numeraire = "lab"
)

ge$z
ge$D
MDCES_demand(
  es = 1, beta = c(0.5, 0.5), xi = c(0, 0),
  w = 1, p = ge$p,
  betaMod = function(u.unmod) {
    beta2 <- 0.95 * plogis(u.unmod, location = 2, scale = 2)
    c(1 - beta2, beta2)
  }
)
```

---

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 and the package data.tree.

As the package data.tree says:

"One of most important things to note about data.tree is that it exhibits reference semantics. In a nutshell, this means that you can modify your tree along the way, without having to reassign it to a variable after each modification. By and large, this is a rather exceptional behavior in R, where value-semantics is king most of the time."

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 (or some child nodes) will be created with the value as the name (or names).

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)

# the same as above
dst <- node_new(
  "firm",
  c("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_plot(dst)
node_plot(dst, TRUE)
```

---

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.

Usage

```
node_plot(node, param = FALSE, ...)
```

Arguments

node	a tree (i.e. a Node object).
param	If TRUE, those parameters such as alpha, beta, es etc. will be shown.
...	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 (or some child nodes) will be created with the value as the name (or names).

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)

# the same as above
dst <- node_set(
  "firm", NA,
  c("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)
```

`output` *Compute the Utility of a Consumer or the Output of a Firm by the Demand Structural Tree*

Description

Given a demand structural tree and an input vector, this function computes the utility of a consumer or the output of a firm. If the demand structural tree has a FUNC-type node, the node should has an attribute named fun that is a function computing the output.

Usage

```
output(node, input)
```

Arguments

node	a demand structural tree.
input	an input vector with names of commodities.

Value

A scalar.

Examples

```
dst <- node_new("output",
                type = "SCES", es = 0, alpha = 1, beta = c(0.5, 0.5),
                "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(0.5, 0.5),
         "labor", "capital"
)

node_plot(dst, TRUE)

p <- c(wheat = 1, iron = 3, labor = 2, capital = 4)
x <-demand_coefficient(dst, p)
output(dst, 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, temporary equilibrium path). 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, A, state)
```

Arguments

`time`	the current time.
`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.
`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)

Grandmont, J.M. (1977). Temporary General Equilibrium Theory. Econometrica 45, 535-572.

See Also

CGE::iep and `sdm2`, `gemTemporaryEquilibriumPath`. 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$ts.z, type = "o", pch = 20)
matplot(ge$ts.p, type = "o", 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$ts.z, type = "o", pch = 20)
matplot(ge$ts.p, type = "o", 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, A) {
  A[[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$ts.z[x[1], 1], type = "o", pch = 20)
plot(x, ge$ts.z[x, 2] / ge$ts.z[x[1], 2], type = "o", pch = 20)
plot(x, c[x] / c[x[1]], type = "o", 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$ts.z, type = "o", pch = 20)
matplot(ge$ts.p, type = "o", 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 averages 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

`makePolicyMeanValue` `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 numeric vector or a positive numeric n-by-m matrix.
coef	a positive number, a positive numeric vector or a positive numeric n-by-m matrix.
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 or a matrix with dimensions the same as the argument ratio.

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")

X <- replicate(3, x)
Y <- ratio_adjust(X, c(0.8, 1, 1.2))
matplot(x, Y, type = "l")
```

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 of the 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

```
beta <- c(0.6, 0.4)
SCES(alpha = 1, beta = beta, x = beta, 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. n is the number of commodity types. m is the number of economic agents. The argument state is a list consisting of time (the current time), p (the current price vector), last.z (the output and utility vector of the previous period), w (the current wealth vector) and last.A (the demand coefficient matrix the previous period).

`B` an n-by-m matrix containing of the output coefficients (i.e. yield coefficients) of producers. Each producer produces one or more commodities. The output of each producer is equal to its activity level multiplied by the output coefficients. Columns corresponding to consumers are usually zeros. 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 relative tolerance condition.
- `maxIteration` — the maximum number of (outer) iterations. 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 (i.e. the number of inner iterations) in each (outer) 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.
  - A - the same as the parameter A of sdm2. When A is a demand structure tree list, it needs not be returned after it is adjusted.
  - 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$last.z is the output and utility vector of the previous period. state$B is the current supply coefficient matrix. state$last.A is the demand coefficient matrix of the previous period. state$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 relative 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 temporary equilibrium path (alias market clearing path, instantaneous equilibrium path) is computed by the function iep. In this package, the temporary 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, A) {
  if (time >= 200) {
    A[[1]]$a <- c(0.5, 0.8)
  } else {
    A[[1]]$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, A) {
  if (time >= 200) {
    A[[1]]$a[2] <- A[[1]]$a[2] * 0.999
  } else {
    A[[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, A) {
  A[[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, 0)
)
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, 0)
)
matplot(ge.high.level$ts.z, type = "l")

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

    state$current.policy.data <- c(time, A[[1]]$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, 0, 0),
    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, 0, 0),
  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 kind of kinked (piecewise) 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")

####
f <- function(theta) {
  p <- c(1, 1)
  structural_function(
    theta,
    c(15, 20),
    function(p) CD_A(alpha = 5, Beta = c(0.6, 0.4), p),
    function(p) CD_A(alpha = 15, Beta = c(0.3, 0.7), p),
    p
  )
}

tmp <- sapply(1:25, f)
matplot(t(tmp), type = "l")
```

var.p — *Population Variance and Population Standard Deviation*

Description

The function var.p computes a population variance. The function sd.p computes a population standard deviation.

Usage

```
var.p(x, wt = rep(1, length(x)), na.rm = FALSE)

sd.p(x, wt = rep(1, length(x)), na.rm = FALSE)
```

Arguments

`x`	a numeric vector.
`wt`	a numeric vector of weights. By default all elements of x are given the same weight.
`na.rm`	a logical value indicating whether NA values should be stripped before the computation proceeds.

Functions

- `var.p`: Population variance.
- `sd.p`: Population standard deviation.

Examples

```
var.p(1:6)

var.p(x = 1:5, wt = 6:10)
var.p(x = rep(1:5, 6:10))

sd.p(x = 1:5, wt = 6:10)
sd.p(x = rep(1:5, 6:10))
```

Index