Package ‘gstest’

April 30, 2025

Title (EN) 'G-Series' in 'R' | (FR) 'G-Séries' en 'R'

Version 3.0.0

Created April 30, 2025, at 3:42:27 PM EDT

Description (EN) Statistics Canada's generalized system devoted to time series benchmarking and reconciliation. The methods used in 'G-Series' essentially come from Dagum and Cholette (2006) <doi:10.1007/0-387-35439-5>. | (FR) Système généralisé de Statistique Canada consacré à l'étalonnage et à la réconciliation de séries chronologiques. Les méthodes utilisées dans 'G-Séries' proviennent essentiellement de Dagum et Cholette (2006) <doi:10.1007/0-387-35439-5>.

License GPL (>= 3)

URL https://ferlmic.github.io/gstest/en/, https://ferlmic.github.io/gstest/fr/

BugReports https://github.com/ferlmic/gstest/issues/

Email g-series@statcan.gc.ca

Depends R (>= 4.0)

Imports ggplot2, ggtext, graphics, grDevices, gridExtra, lifecycle, osqp, rlang (>= 1.1.0), stats, utils, xmpdf

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

LazyData true

Contents

`benchmarking` *Rétablir les contraintes temporelles*

Description

Réplication de la procédure BENCHMARKING de G-Séries 2.0 en SAS® (PROC BENCHMARKING). Voir la documentation de G-Séries 2.0 pour plus de détails (Statistique Canada 2016).

Cette fonction assure la cohérence entre les données de séries chronologiques d'une même variable cible mesurée à des fréquences différentes (ex., infra-annuellement et annuellement). L'étalonnage consiste à imposer le niveau de la série d'étalons (ex., données annuelles) tout en minimisant, autant que possible, les révisions au mouvement observé dans la série indicatrice (ex., données infra-annuelles). La fonction permet également l'étalonnage non contraignant où la série d'étalons peut également être révisée.

La fonction peut également être utilisée pour des sujets liés à l'étalonnage tels que la *distribution temporelle* (action réciproque de l'étalonnage : désagrégation de la série d'étalons en observations plus fréquentes), la *calendarisation* (cas spécial de distribution temporelle) et le *raccordement* (« *linking* » : connexion de différents segments de séries chronologiques en une série chronologique unique et cohérente).

Plusieurs séries peuvent être étalonnées en un seul appel de fonction.

Usage

```
benchmarking(
  series_df,
  benchmarks_df,
  rho,
  lambda,
  biasOption,
  bias = NA,
  tolV = 0.001,
  tolP = NA,
  warnNegResult = TRUE,
  tolN = -0.001,
  var = "value",
  with = NULL,
  by = NULL,
  verbose = FALSE,

  # Nouveau dans G-Séries 3.0
  constant = 0,
  negInput_option = 0,
  allCols = FALSE,
  quiet = FALSE
)
```

Arguments

`series_df`	(obligatoire) *Data frame*, ou objet compatible, qui contient les données de la (des) série(s) indicatrice(s) à étalonner. En plus de la (des) variable(s) contenant les données, spécifiée(s) avec l'argument `var`, le *data frame* doit aussi contenir deux variables numériques, `year` et `period`, identifiant les périodes des séries indicatrices.
`benchmarks_df`	(obligatoire) *Data frame*, ou objet compatible, qui contient les étalons. En plus de la (des) variable(s) contenant les données, spécifiée(s) avec l'argument `with`, le *data frame* doit aussi contenir quatre variables numériques, `startYear`, `startPeriod`, `endYear` et `endPeriod`, identifiant les périodes des séries indicatrices couvertes par chaque étalon.
`rho`	(obligatoire)

Nombre réel compris dans l'intervalle $[0, 1]$ qui spécifie la valeur du paramètre autorégressif ρ. Voir la section **Détails** pour plus d'informations sur l'effet du paramètre ρ.

`lambda` (obligatoire)

Nombre réel, avec des valeurs suggérées dans l'intervalle $[-3, 3]$, qui spécifie la valeur du paramètre du modèle d'ajustement λ. Les valeurs typiques sont `lambda = 0.0` pour un modèle additif et `lambda = 1.0` pour un modèle proportionnel.

`biasOption` (obligatoire)

Spécification de l'option d'estimation du biais :

- `1` : Ne pas estimer le biais. Le biais utilisé pour corriger la série indicatrice sera la valeur spécifiée avec l'argument `bias`.
- `2` : Estimer le biais, afficher le résultat, mais ne pas l'utiliser. Le biais utilisé pour corriger la série indicatrice sera la valeur spécifiée avec l'argument `bias`.
- `3` : Estimer le biais, afficher le résultat et utiliser le biais estimé pour corriger la série indicatrice. Toute valeur spécifiée avec l'argument `bias` sera ignorée.

L'argument `biasOption` n'est pas utilisé quand `rho = 1.0`. Voir la section **Détails** pour plus d'informations sur le biais.

`bias` (optionnel)

Nombre réel, ou `NA`, spécifiant la valeur du biais défini par l'utilisateur à utiliser pour la correction de la série indicatrice avant de procéder à l'étalonnage. Le biais est ajouté à la série indicatrice avec un modèle additif (argument `lambda = 0.0`) alors qu'il est multiplié dans le cas contraire (argument `lambda != 0.0`). Aucune correction de biais n'est appliquée lorsque `bias = NA`, ce qui équivaut à spécifier `bias = 0.0` lorsque `lambda = 0.0` et `bias = 1.0` dans le cas contraire. L'argument `bias` n'est pas utilisé lorsque `biasOption = 3` ou `rho = 1.0`. Voir la section **Détails** pour plus d'informations sur le biais.

La valeur par défaut est `bias = NA` (pas de biais défini par l'utilisateur).

`tolV`, `tolP` (optionnel)

Nombre réel non négatif, ou `NA`, spécifiant la tolérance, en valeur absolue ou en pourcentage, à utiliser pour la validation des étalons contraignants (coefficient d'altérabilité de 0.0) en sortie. Cette validation consiste à comparer la valeur des étalons contraignants en entrée à la valeur équivalente calculée à partir des données de la série étalonnée (sortie). Les arguments `tolV` et `tolP` ne peuvent pas être spécifiés tous les deux à la fois (l'un doit être spécifié tandis que l'autre doit être `NA`).

Exemple : pour une tolérance de 10 *unités*, spécifiez `tolV = 10, tolP = NA`; pour une tolérance de 1%, spécifiez `tolV = NA, tolP = 0.01`.

Les valeurs par défaut sont `tolV = 0.001` et `tolP = NA`.

`warnNegResult` (optionnel)

Argument logique (*logical*) spécifiant si un message d'avertissement doit être affiché lorsqu'une valeur négative créée par la fonction dans la série étalonnée (en sortie) est inférieure au seuil spécifié avec l'argument `tolN`.

La valeur par défaut est `warnNegResult = TRUE`.

`tolN` (optionnel)
Nombre réel négatif spécifiant le seuil pour l'identification des valeurs négatives. Une valeur est considérée négative lorsqu'elle est inférieure à ce seuil.
La valeur par défaut est `tolN = -0.001`.

`var` (optionnel)
Vecteur (longueur minimale de 1) de chaînes de caractères spécifiant le(s) nom(s) de variable(s) du *data frame* des séries indicatrices (argument `series_df`) contenant les valeurs et (optionnellement) les coefficients d'altérabilité définis par l'utilisateur de la (des) série(s) à étalonner. Ces variables doivent être numériques.
La syntaxe est `var = c("serie1 </ alt_ser1>", "serie2 </ alt_ser2>", ...)`. Des coefficients d'altérabilité par défaut de 1.0 sont utilisés lorsqu'une variable de coefficients d'altérabilité définie par l'utilisateur n'est pas spécifiée à côté d'une variable de série indicatrice. Voir la section **Détails** pour plus d'informations sur les coefficients d'altérabilité.
Exemple : `var = "value / alter"` étalonnerait la variable `value` du *data frame* des séries indicatrices avec les coefficients d'altérabilité contenus dans la variable `alter` tandis que `var = c("value / alter", "value2")` étalonnerait en plus la variable `value2` avec des coefficients d'altérabilité par défaut de 1.0.
La valeur par défaut est `var = "value"` (étalonner la variable `value` avec des coefficients d'altérabilité par défaut de 1.0).

`with` (optionnel)
Vecteur (même longueur que l'argument `var`) de chaînes de caractères, ou `NULL`, spécifiant le(s) nom(s) de variable(s) du *data frame* des étalons (argument `benchmarks_df`) contenant les valeurs et (optionnellement) les coefficients d'altérabilité définis par l'utilisateur des étalons. Ces variables doivent être numériques. La spécification de `with = NULL` entraîne l'utilisation de variable(s) d'étalons correspondant à la (aux) variable(s) spécifiée(s) avec l'argument `var` sans coefficients d'altérabilité d'étalons définis par l'utilisateur (c'est à dire des coefficients d'altérabilité par défaut de 0.0 correspondant à des étalons contraignants).
La syntaxe est `with = NULL` ou `with = c("bmk1 </ alt_bmk1>", "bmk2 </ alt_bmk2>", ...)`. Des coefficients d'altérabilité par défaut de 0.0 (étalons contraignants) sont utilisés lorsqu'une variable de coefficients d'altérabilité définie par l'utilisateur n'est pas spécifiée à côté d'une variable d'étalon. Voir la section **Détails** pour plus d'informations sur les coefficients d'altérabilité.
Exemple : `with = "val_bmk"` utiliserait la variable `val_bmk` du *data frame* des étalons avec les coefficients d'altérabilité par défaut de 0.0 pour étalonner la série indicatrice tandis que `with = c("val_bmk", "val_bmk2 / alt_bmk2")` étalonnerait en plus une deuxième série indicatrice en utilisant la variable d'étalons `val_bmk2` avec les coefficients d'altérabilité d'étalons contenus dans la variable `alt_bmk2`.
La valeur par défaut est `with = NULL` (même(s) variable(s) d'étalons que l'argument `var` avec des coefficients d'altérabilité d'étalons par défaut de 0.0).

`by` (optionnel)
Vecteur (longueur minimale de 1) de chaînes de caractères, ou `NULL`, spécifiant le(s) nom(s) de variable(s) dans les *data frames* d'entrée (arguments `series_df` et `benchmarks_df`) à utiliser pour former des groupes (pour le traitement « groupes-BY ») et permettre l'étalonnage de plusieurs séries en un seul appel de fonction.

Les variables groupes-BY peuvent être numériques ou caractères (facteurs ou non), doivent être présentes dans les deux *data frames* d'entrée et apparaîtront dans les trois *data frames* de sortie (voir la section **Valeur de retour**). Le traitement groupes-BY n'est pas implémenté lorsque `by = NULL`. Voir « Étalonnage de plusieurs séries » dans la section **Détails** pour plus d'informations.

La valeur par défaut est `by = NULL` (pas de traitement groupes-BY).

`verbose` (optionnel)

Argument logique (*logical*) spécifiant si les informations sur les étapes intermédiaires avec le temps d'exécution (temps réel et non le temps CPU) doivent être affichées. Notez que spécifier l'argument `quiet = TRUE` annulerait l'argument `verbose`.

La valeur par défaut est `verbose = FALSE`.

`constant` (optionnel)

Nombre réel qui spécifie une valeur à ajouter temporairement à la fois à la (aux) série(s) indicatrice(s) et aux étalons avant de résoudre les problèmes d'étalonnage proportionnels (`lambda != 0.0`). La constante temporaire est enlevée de la série étalonnée finale en sortie. Par exemple, la spécification d'une (petite) constante permettrait l'étalonnage proportionnel avec `rho = 1` (étalonnage de Denton proportionnel) sur avec des séries indicatrices qui comprennent des valeurs de 0. Sinon, l'étalonnage proportionnel avec des valeurs de 0 pour la série indicatrice n'est possible que lorsque `rho < 1`. Spécifier une constante avec l'étalonnage additif (`lambda = 0.0`) n'a pas d'impact sur les données étalonnées résultantes. Les variables de données dans le *data frame* de sortie **graphTable** incluent la constante, correspondant au problème d'étalonnage effectivement résolu par la fonction.

La valeur par défaut est `constant = 0` (pas de constante additive temporaire).

`negInput_option`

(optionnel)

Traitement des valeurs négatives dans les données d'entrée pour l'étalonnage proportionnel (`lambda != 0.0`) :

- `0` : Ne pas autoriser les valeurs négatives pour l'étalonnage proportionnel. Un message d'erreur est affiché en présence de valeurs négatives dans les séries indicatrices ou les étalons d'entrée et des valeurs manquantes (`NA`) sont renvoyées pour les séries étalonnées. Ceci correspond au comportement de G-Séries 2.0.
- `1` : Autoriser les valeurs négatives pour l'étalonnage proportionnel mais avec l'affichage d'un message d'avertissement.
- `2` : Autoriser les valeurs négatives pour l'étalonnage proportionnel sans afficher de message.

La valeur par défaut est `negInput_option = 0` (ne pas autoriser les valeurs négatives pour l'étalonnage proportionnel).

`allCols` (optionnel)

Argument logique (*logical*) spécifiant si toutes les variables du *data frame* des séries indicatrices (argument `series_df`), autres que `year` et `period`, déterminent l'ensemble des séries à étalonner. Les valeurs spécifiées avec les arguments `var` et `with` sont ignorées lorsque `allCols = TRUE`, ce qui implique

automatiquement des coefficients d'altérabilité par défaut, et des variables avec les mêmes noms que les séries indicatrices doivent exister dans le *data frame* des étalons (argument `benchmarks_df`).

La valeur par défaut est `allCols = FALSE`.

`quiet` (optionnel)

Argument logique (*logical*) spécifiant s'il faut ou non afficher uniquement les informations essentielles telles que les messages d'avertissements, les messages d'erreurs et les informations sur les variables (séries) ou les groupes-BY lorsque plusieurs séries sont étalonnées en un seul appel à la fonction. Nous vous déconseillons d'*envelopper* votre appel à `benchmarking()` avec `suppressMessages()` afin de supprimer l'affichage des informations sur les variables (séries) ou les groupes-BY lors du traitement de plusieurs séries, car cela compliquerait le dépannage en cas de problèmes avec des séries individuelles. Notez que la spécification de `quiet = TRUE` annulera également l'argument `verbose`.

La valeur par défaut est `quiet = FALSE`.

Details

Lorsque $\rho < 1$, cette fonction renvoie la solution des moindres carrés généralisés d'un cas particulier du modèle général d'étalonnage basé sur la régression proposé par Dagum et Cholette (2006). Le modèle, sous forme matricielle, est le suivant :

$$\begin{bmatrix} s^\dagger \\ a \end{bmatrix} = \begin{bmatrix} I \\ J \end{bmatrix} \theta + \begin{bmatrix} e \\ \varepsilon \end{bmatrix}$$

où

- a est le vecteur de longueur M des étalons.
- $s^\dagger = \begin{cases} s + b & \text{si } \lambda = 0 \\ s \cdot b & \text{sinon} \end{cases}$ est le vecteur de longueur T des valeurs de la série indicatrice corrigée pour le biais, s désignant la série indicatrice initiale (d'entrée).
- b est le bias, qui est spécifié avec l'argument `bias` lorsque `bias_option != 3` ou, lorsque `bias_option = 3`, est estimé par $\hat{b} = \begin{cases} \frac{{1_M}^{\mathrm{T}}(a - Js)}{{1_M}^{\mathrm{T}} J 1_T} & \text{si } \lambda = 0 \\ \frac{{1_M}^{\mathrm{T}} a}{{1_M}^{\mathrm{T}} Js} & \text{sinon} \end{cases}$, où $1_X = (1, ..., 1)^{\mathrm{T}}$ est un vecteur de 1 de longueur X.
- J est la matrice $M \times T$ des contraintes d'agrégation temporelles avec les éléments $j_{m,t} = \begin{cases} 1 & \text{si l'étalon } m \text{ couvre la période } t \\ 0 & \text{sinon} \end{cases}$.
- θ est le vecteur des valeurs de la série finale (étalonnée).
- $e \sim (0, V_e)$ est le vecteur des erreurs de mesure de $s^\dagger$ avec matrice de covariance $V_e = C \Omega_e C$.
- $C = \mathrm{diag}\left(\sqrt{c_{s^\dagger}} \left| s^\dagger \right|^\lambda\right)$ où $c_{s^\dagger}$ est le vecteur des coefficients d'altérabilité de $s^\dagger$, en définissant $0^0 = 1$.
- Ω_e est une matrice $T \times T$ avec les éléments $\omega_{e_{i,j}} = \rho^{|i-j|}$ représentant l'autocorrelation d'un processus AR(1), en définissant encore $0^0 = 1$.
- $\varepsilon \sim (0, V_\varepsilon)$ est le vecteur des erreurs de mesure des étalons a avec matrice de covariance $V_\varepsilon = \mathrm{diag}\left(c_a a\right)$ où c_a est le vecteur des coefficients d'altérabilité des étalons a.

La solution des moindres carrés généralisés est la suivante :

$$\hat{\theta} = s^{\dagger} + V_e J^{\mathrm{T}} \left(J V_e J^{\mathrm{T}} + V_{\varepsilon} \right)^{+} \left(a - J s^{\dagger} \right)$$

où A^{+} désigne l'inverse de Moore-Penrose de la matrice A.

Lorsque $\rho = 1$, la fonction renvoie la solution de la méthode de Denton (modifiée) :

$$\hat{\theta} = s + W \left(a - J s \right)$$

où

- W est la matrice du coin supérieur droit du produit matriciel suivant

$$\begin{bmatrix} D^{+}\Delta^{\mathrm{T}}\Delta D^{+} & J^{\mathrm{T}} \\ J & 0 \end{bmatrix}^{+} \begin{bmatrix} D^{+}\Delta^{\mathrm{T}}\Delta D^{+} & 0 \\ J & I_M \end{bmatrix} = \begin{bmatrix} I_T & W \\ 0 & W_{\nu} \end{bmatrix}$$

- $D = \mathrm{diag}\left(|s|^{\lambda} \right)$, en définissant $0^0 = 1$. Notez que D correspond à C avec $c_{s^{\dagger}} = 1.0$ et sans correction de biais (arguments `bias_option = 1` et `bias = NA`).
- Δ est une matrice $T - 1 \times T$ avec les éléments $\delta_{i,j} = \begin{cases} -1 & \text{si } i = j \\ 1 & \text{si } j = i + 1 \\ 0 & \text{sinon} \end{cases}$.
- W_{ν} est une matrice $M \times M$ associée aux multiplicateurs de Lagrange du problème de minimisation correspondant exprimé comme suit :

$$\begin{aligned} &\underset{\theta}{\text{minimiser}} && \sum_{t \geq 2} \left[\frac{(s_t - \theta_t)}{|s_t|^{\lambda}} - \frac{(s_{t-1} - \theta_{t-1})}{|s_{t-1}|^{\lambda}} \right]^2 \\ &\text{sous contrainte(s)} && a = J\theta \end{aligned}$$

Voir Quenneville et al. (2006) et Dagum and Cholette (2006) pour les détails.

Paramètre autorégressif ρ et le *biais*:

Le paramètre ρ (argument `rho`) est associé au changement entre la série indicatrice (d'entrée) et la série étalonnée (de sortie) pour deux périodes consécutives et est souvent appelé *paramètre de préservation du mouvement*. Plus la valeur de ρ est grande, plus les mouvements d'une période à l'autre de la série indicatrice sont préservés dans la série étalonnée. Avec $\rho = 0$, la préservation des mouvements d'une période à l'autre n'est pas appliquée et les ajustements d'étalonnage qui en résultent ne sont pas lisses, comme dans le cas du prorata ($\rho = 0$ et $\lambda = 0.5$) où les ajustements prennent la forme d'une *fonction en escalier*. À l'autre extrémité du spectre on trouve $\rho = 1$, appelé *étalonnage de Denton*, où la préservation du mouvement d'une période à l'autre est maximisée, ce qui se traduit par l'ensemble le plus lisse possible d'ajustements d'étalonnage disponibles avec la fonction.

Le *biais* représente l'écart attendu entre les étalons et la série indicatrice. Il peut être utilisé pour pré-ajuster la série indicatrice afin de réduire, en moyenne, les écarts entre les deux sources de données. La correction du biais, qui est spécifiée avec les arguments `biasOption` et `bias`, peut être particulièrement utile pour les périodes non couvertes par les étalons lorsque $\rho < 1$. Dans ce contexte, le paramètre ρ dicte la vitesse à laquelle les ajustements d'étalonnage projetés convergent vers le biais (ou convergent vers *aucun ajustement* sans correction du biais) pour les périodes non couvertes par un étalon. Plus la valeur de ρ est petite, plus la convergence vers le

biais est rapide, avec convergence immédiate lorsque $\rho = 0$ et aucune convergence (l'ajustement de la dernière période couverte par un étalon est répété indéfiniment) lorsque $\rho = 1$ (étalonnage de Denton). En fait, les arguments `biasOption` et `bias` ne sont pas utilisés lorsque $\rho = 1$ puisque la correction du biais n'a pas d'impact sur les résultats de l'étalonnage de Denton. La valeur suggérée pour ρ est 0.9 pour les indicateurs mensuels et $0.9^3 = 0.729$ pour les indicateurs trimestriels, ce qui représente un compromis raisonnable entre maximiser la préservation du mouvement et réduire les révisions à mesure que de nouveaux étalons deviendront disponibles à l'avenir (*problème d'actualité* de l'étalonnage). En pratique, il convient de noter que l'étalonnage de Denton pourrait être *approximé* avec le modèle basé sur la régression en utilisant une valeur de ρ inférieure à, mais très proche de 1.0 (par exemple, $\rho = 0.999$). Voir Dagum et Cholette (2006) pour une discussion complète sur ce sujet.

Coefficients d'altérabilité:

Les coefficients d'altérabilité $c_{s^\dagger}$ et c_a représentent conceptuellement les erreurs de mesure associées aux valeurs de la série indicatrice (corrigée pour le biais) $s^\dagger$ et des étalons a respectivement. Il s'agit de nombres réels non négatifs qui, en pratique, spécifient l'ampleur de la modification permise d'une valeur initiale par rapport aux autres valeurs. Un coefficient d'altérabilité de 0.0 définit une valeur fixe (contraignante), tandis qu'un coefficient d'altérabilité supérieur à 0.0 définit une valeur libre (non contraignante). L'augmentation du coefficient d'altérabilité d'une valeur initiale entraîne davantage de changements pour cette valeur dans la solution d'étalonnage et, inversement, moins de changements lorsque l'on diminue le coefficient d'altérabilité. Les coefficients d'altérabilité par défaut sont 0.0 pour les étalons (contraignants) et 1.0 pour les valeurs de la série indicatrice (non contraignantes). Remarques importantes :

- Avec une valeur de $\rho = 1$ (argument `rho = 1`, associé à l'étalonnage de Denton), seuls les coefficients d'altérabilité par défaut (0.0 pour un étalon et 1.0 pour une valeur de série indicatrice) sont valides. La spécification de variables de coefficients d'altérabilité définies par l'utilisateur n'est donc pas autorisée. Si de telles variables sont spécifiées (voir les arguments `var` et `with`), la fonction les ignore et affiche un message d'avertissement dans la console.
- Les coefficients d'altérabilité $c_{s^\dagger}$ entrent en jeu après que la série indicatrice ait été corrigée pour le biais, lorsqu'applicable ($c_{s^\dagger}$ est associé à $s^\dagger$ et non à s). Cela signifie que la spécification d'un coefficient d'altérabilité de 0.0 pour une valeur de série indicatrice donnée **ne se traduira pas** par une valeur inchangée après étalonnage **avec correction du biais** (voir les arguments `biasOption` et `bias`).

Les étalons non contraignants, le cas échéant, peuvent être récupérés (calculés) à partir de la série étalonnée (voir le *data frame* de sortie **series** dans la section **Valeur de retour**). Le *data frame* de sortie **benchmarks** contient toujours les étalons fournis dans le *data frame* d'entrée des étalons (argument `benchmarks_df`).

Étalonnage de plusieurs séries:

Plusieurs séries peuvent être étalonnées en un seul appel à `benchmarking()`, en spécifiant `allCols = TRUE`, en spécifiant (manuellement) plusieurs variables avec l'argument `var` (et l'argument `with`) ou avec le traitement groupes-BY (argument `by != NULL`). Une distinction importante est que toutes les séries indicatrices spécifiées avec `allCols = TRUE` ou avec l'argument `var` (et les étalons avec l'argument `with`) doivent avoir la même longueur, c'est-à-dire le même ensemble de périodes et et le même ensemble (nombre) d'étalons. L'étalonnage de séries de longueurs différentes (différents ensembles de périodes) ou avec différents ensembles (nombres) d'étalons doit être effectué avec un traitement groupes-BY sur des données empilées pour les *data frames* d'entrée de séries

indicatrices et d'étalons (voir les fonctions utilitaires `stack_tsDF()` et `stack_bmkDF()`). Les arguments `by` et `var` peuvent être combinés afin d'implémenter le traitement groupes-BY pour des séries multiples comme illustré par l'*Exemple 2* dans la section **Exemples**. Alors que l'utilisation de variables multiples avec 'argument `var` (ou `allCols = TRUE`) sans traitement groupes-BY (argument `by = NULL`) est légèrement plus efficace (plus rapide), une approche groupes-BY avec une seule variable de série est généralement recommandée car elle est plus générale (fonctionne dans tous les contextes). Cette dernière est illustrée par l'*Exemple 3* dans la section **Exemples**. Les variables BY spécifiées avec l'argument `by` apparaissent dans les trois *data frames* de sortie.

Arguments `constant` et `negInput_option`:

Ces arguments permettent d'étendre l'utilisation de l'étalonnage proportionnel à un plus grand nombre de problèmes. Leurs valeurs par défaut correspondent au comportement de G-Séries 2.0 (SAS® PROC BENCHMARKING) pour lequel des options équivalentes ne sont pas définies. Bien que l'étalonnage proportionnel ne soit pas nécessairement l'approche la plus appropriée (l'étalonnage additif pourrait être plus indiqué) lorsque les valeurs de la série indicatrice approchent de 0 (ratios d'une période à l'autre instables) ou « traversent la ligne de 0 » et peuvent donc passer de positives à négatives et vice-versa (ratios d'une période à l'autre difficiles à interpréter), ces cas ne sont pas invalides d'un point de vue mathématique (le problème d'étalonnage proportionnel associé peut être résolu). Il est toutefois fortement recommandé d'analyser et de valider soigneusement les données étalonnées obtenues dans ces situations pour s'assurer qu'elles correspondent à des solutions raisonnables et interprétables.

Traitement des valeurs manquantes (`NA`):

- Si une valeur manquante apparaît dans l'une des variables du *data frame* d'entrée des étalons (autre que les variables BY), les enregistrements avec les valeurs manquantes sont laissés de côté, un message d'avertissement est affiché et la fonction s'exécute.
- Si une valeur manquante apparaît dans les variables `year` ou `period` du *data frame* d'entrée des séries indicatrices et que des variables BY sont spécifiées, le groupe-BY correspondant est ignoré, un message d'avertissement s'affiche et la fonction passe au groupe-BY suivant. Si aucune variable BY n'est spécifiée, un message d'avertissement s'affiche et aucun traitement n'est effectué.
- Si une valeur manquante apparaît dans l'une des variables des données de série du *data frame* d'entrée des séries indicatrices et que des variables BY sont spécifiées, le groupe-BY correspondant est ignoré, un message d'avertissement est affiché et la fonction passe au groupe-BY suivant. Si aucune variable BY n'est spécifiée, la série indicatrice concernée n'est pas traitée, un message d'avertissement est affiché et la fonction passe à la série indicatrice suivante (le cas échéant).

Value

La fonction renvoie une liste de trois *data frames* :

- **series** : *data frame* contenant les données étalonnées (sortie principale de la fonction). Les variables BY spécifiées avec l'argument `by` sont incluses dans le *data frame* mais pas les variables de coefficient d'altérabilité spécifiées avec l'argument `var`.
- **benchmarks** : copie du *data frame* d'entrée des étalons (à l'exclusion des étalons non valides, le cas échéant). Les variables BY spécifiées avec l'argument `by` sont incluses dans le *data frame* mais pas les variables de coefficient d'altérabilité spécifiées avec l'argument `with`.

- **graphTable** : *data frame* contenant des données supplémentaires utiles pour produire des tableaux et des graphiques analytiques (voir la fonction `plot_graphTable()`). Il contient les variables suivantes en plus des variables BY spécifiées avec l'argument `by` :
  - `varSeries` : Nom de la variable de la série indicatrice
  - `varBenchmarks` : Nom de la variable des étalons
  - `altSeries` : Nom de la variable des coefficients d'altérabilité définis par l'utilisateur pour la série indicatrice
  - `altSeriesValue` : Coefficients d'altérabilité de la série indicatrice
  - `altbenchmarks` : Nom de la variable des coefficients d'altérabilité définis par l'utilisateur pour les étalons
  - `altBenchmarksValue` : Coefficients d'altérabilité des étalons
  - `t` : Identificateur de la période de la série indicatrice (1 à T)
  - `m` : Identificateur des périodes de couverture de l'étalon (1 à M)
  - `year` : Année civile du point de données
  - `period` : Valeur de la période (du cycle) du point de données (1 à `periodicity`)
  - `constant` : Constante additive temporaire (argument `constant`)
  - `rho` : Paramètre autorégressif ρ (argument `rho`)
  - `lambda` : Paramètre du modèle d'ajustement λ (argument `lambda`)
  - `bias` : Ajustement du biais (par défaut, défini par l'utilisateur ou biais estimé selon les arguments `biasOption` et `bias`)
  - `periodicity` : Le nombre maximum de périodes dans une année (par exemple 4 pour une série indicatrice trimestrielle)
  - `date` : Chaîne de caractères combinant les valeurs des variables `year` et `period`
  - `subAnnual` : Valeurs de la série indicatrice
  - `benchmarked` : Valeurs de la série étalonnée
  - `avgBenchmark` : Valeurs des étalons divisées par le nombre de périodes de couverture
  - `avgSubAnnual` : Valeurs moyennes de la série indicatrice (variable `subAnnual`) pour les périodes couvertes par les étalons
  - `subAnnualCorrected` : Valeurs de la série indicatrice corrigée pour le biais
  - `benchmarkedSubAnnualRatio` : Différence ($\lambda = 0$) ou ratio ($\lambda \neq 0$) des valeurs des variables `benchmarked` et `subAnnual`
  - `avgBenchmarkSubAnnualRatio` : Différence ($\lambda = 0$) ou ratio ($\lambda \neq 0$) des valeurs des variables `avgBenchmark` et `avgSubAnnual`
  - `growthRateSubAnnual` : Différence ($\lambda = 0$) ou différence relative ($\lambda \neq 0$) d'une période à l'autre des valeurs de la série indicatrice (variable `subAnnual`)
  - `growthRateBenchmarked` : Différence ($\lambda = 0$) ou différence relative ($\lambda \neq 0$) d'une période à l'autre des valeurs de la série étalonnée (variable `benchmarked`)

Notes :

- Le *data frame* de sortie **benchmarks** contient toujours les étalons originaux fournis dans le *data frame* d'entrée des étalons. Les étalons modifiés non contraignants, le cas échéant, peuvent être récupérés (calculés) à partir du *data frame* de sortie **series**.

- La fonction renvoie un objet NULL si une erreur se produit avant que le traitement des données ne puisse commencer. Dans le cas contraire, si l'exécution est suffisamment avancée pour que le traitement des données puisse commencer, alors un objet incomplet sera renvoyé en cas d'erreur (par exemple, un *data frame* de sortie **series** avec des valeurs NA pour les données étalonnées).
- La fonction renvoie des objets « data.frame » qui peuvent être explicitement convertis en d'autres types d'objets avec la fonction `as*()` appropriée (ex., `tibble::as_tibble()` convertirait n'importe lequel d'entre eux en tibble).

References

Dagum, E. B. et P. Cholette (2006). **Benchmarking, Temporal Distribution and Reconciliation Methods of Time Series**. Springer-Verlag, New York, Lecture Notes in Statistics, Vol. 186

Fortier, S. et B. Quenneville (2007). « Theory and Application of Benchmarking in Business Surveys ». **Proceedings of the Third International Conference on Establishment Surveys (ICES-III)**. Montréal, juin 2007.

Latendresse, E., M. Djona et S. Fortier (2007). « Benchmarking Sub-Annual Series to Annual Totals – From Concepts to SAS® Procedure and Enterprise Guide® Custom Task ». **Proceedings of the SAS® Global Forum 2007 Conference**. Cary, NC: SAS Institute Inc.

Quenneville, B., S. Fortier, Z.-G. Chen et E. Latendresse (2006). « Recent Developments in Benchmarking to Annual Totals in X-12-ARIMA and at Statistics Canada ». **Proceedings of the Eurostat Conference on Seasonality, Seasonal Adjustment and Their Implications for Short-Term Analysis and Forecasting**. Luxembourg, mai 2006.

Quenneville, B., P. Cholette, S. Fortier et J. Bérubé (2010). « Benchmarking Sub-Annual Indicator Series to Annual Control Totals (Forillon v1.04.001) ». **Document interne**. Statistique Canada, Ottawa, Canada.

Quenneville, B. et S. Fortier (2012). « Restoring Accounting Constraints in Time Series – Methods and Software for a Statistical Agency ». **Economic Time Series: Modeling and Seasonality**. Chapman & Hall, New York.

Statistique Canada (2012). **Théorie et application de l'étalonnage (Code du cours 0436)**. Statistique Canada, Ottawa, Canada.

Statistique Canada (2016). « La procédure BENCHMARKING ». **Guide de l'utilisateur de G-Séries 2.0**. Statistique Canada, Ottawa, Canada.

See Also

`stock_benchmarking()` `plot_graphTable()` bench_graphs `plot_benchAdj()` `gs.gInv_MP()` aliases

Examples

```
# Définir le répertoire de travail (pour les fichiers graphiques PDF)
rep_ini <- getwd()
setwd(tempdir())


###########
# Exemple 1 : Cas simple d'étalonnage d'une série trimestrielle à des valeurs annuelles
```

```
# Série indicatrice trimestrielle
mes_ind1 <- ts_to_tsDF(ts(c(1.9, 2.4, 3.1, 2.2, 2.0, 2.6, 3.4, 2.4, 2.3),
                          start = c(2015, 1),
                          frequency = 4))
mes_ind1

# Étalons annuels pour données trimestrielles
mes_eta1 <- ts_to_bmkDF(ts(c(10.3, 10.2),
                           start = 2015,
                           frequency = 1),
                        ind_frequency = 4)
mes_eta1

# Étalonnage avec...
#   - valeur de `rho` recommandée pour des séries trimestrielles (`rho = 0.729`)
#   - modèle proportionnel (`lambda = 1`)
#   - correction de la série indicatrice pour le biais avec estimation du biais
#     (`biasOption = 3`)
res_eta1 <- benchmarking(mes_ind1,
                         mes_eta1,
                         rho = 0.729,
                         lambda = 1,
                         biasOption = 3)

# Générerer les graphiques d'étalonnage
plot_graphTable(res_eta1$graphTable, "Graphs_ex1.pdf")


############
# Exemple 2 : Étalonnage de deux séries trimestrielles à des valeurs annuelles,
#             avec groupes-BY et coef. d'altérabilité définis par l'utilisateur.

# Données sur les ventes (mêmes ventes pour les groupes A et B; seuls les coef.
# d'alté. pour les ventes de camionnettes diffèrent)
ventes_tri <- ts(matrix(c(# Voitures
                          1851, 2436, 3115, 2205, 1987, 2635, 3435, 2361, 2183, 2822,
                          3664, 2550, 2342, 3001, 3779, 2538, 2363, 3090, 3807, 2631,
                          2601, 3063, 3961, 2774, 2476, 3083, 3864, 2773, 2489, 3082,
                          # Camionnettes
                          1900, 2200, 3000, 2000, 1900, 2500, 3800, 2500, 2100, 3100,
                          3650, 2950, 3300, 4000, 3290, 2600, 2010, 3600, 3500, 2100,
                          2050, 3500, 4290, 2800, 2770, 3080, 3100, 2800, 3100, 2860),
                        ncol = 2),
                 start = c(2011, 1),
                 frequency = 4,
                 names = c("voitures", "camionnettes"))

ventes_ann <- ts(matrix(c(# Voitures
                          10324, 10200, 10582, 11097, 11582, 11092,
                          # Camionnettes
                          12000, 10400, 11550, 11400, 14500, 16000),
                        ncol = 2),
```

```
                start = 2011,
                frequency = 1,
                names = c("voitures", "camionnettes"))

# Séries indicatrices trimestrielles (avec les coef. d'alté. par défaut pour l'instant)
mes_ind2 <- rbind(cbind(data.frame(groupe = rep("A", nrow(ventes_tri)),
                                   alt_cam = rep(1, nrow(ventes_tri))),
                        ts_to_tsDF(ventes_tri)),
                  cbind(data.frame(groupe = rep("B", nrow(ventes_tri)),
                                   alt_cam = rep(1, nrow(ventes_tri))),
                        ts_to_tsDF(ventes_tri)))

# Ventes contraignantes de camionnettes (coef. d'alté. = 0) pour 2012 T1 et T2
# dans le groupe A (lignes 5 et 6)
mes_ind2$alt_cam[c(5,6)] <- 0
head(mes_ind2, n = 10)
tail(mes_ind2)

# Étalons annuels pour données trimestrielles (sans coef. d'alté.)
mes_eta2 <- rbind(cbind(data.frame(groupe = rep("A", nrow(ventes_ann))),
                        ts_to_bmkDF(ventes_ann, ind_frequency = 4)),
                  cbind(data.frame(groupe = rep("B", nrow(ventes_ann))),
                        ts_to_bmkDF(ventes_ann, ind_frequency = 4)))
mes_eta2

# Étalonnage avec...
#   - valeur de `rho` recommandée pour des séries trimestrielles (`rho = 0.729`)
#   - modèle proportionnel (`lambda = 1`)
#   - sans correction du biais (`biasOption = 1` et `bias` non spécifié)
#   - `quiet = TRUE` afin d'éviter l'affichage de l'en-tête de la fonction
res_eta2 <- benchmarking(mes_ind2,
                         mes_eta2,
                         rho = 0.729,
                         lambda = 1,
                         biasOption = 1,
                         var = c("voitures", "camionnettes / alt_cam"),
                         with = c("voitures", "camionnettes"),
                         by = "groupe",
                         quiet = TRUE)

# Générerer les graphiques d'étalonnage
plot_graphTable(res_eta2$graphTable, "Graphs_ex2.pdf")

# Vérifier la valeur des ventes de camionnettes pour 2012 T1 et T2
# dans le groupe A (valeurs fixes)
all.equal(mes_ind2$camionnettes[c(5,6)], res_eta2$series$camionnettes[c(5,6)])


###########
# Exemple 3 : identique à l'exemple 2, mais en étalonnant les 4 séries
#             en tant que groupes-BY (4 groupes-BY au lieu de 2)

ventes_tri2 <- ts.union(A = ventes_tri, B = ventes_tri)
```

```
mes_ind3 <- stack_tsDF(ts_to_tsDF(ventes_tri2))
mes_ind3$alter <- 1
mes_ind3$alter[mes_ind3$series == "A.camionnettes"
               & mes_ind3$year == 2012 & mes_ind3$period <= 2] <- 0
head(mes_ind3)
tail(mes_ind3)

ventes_ann2 <- ts.union(A = ventes_ann, B = ventes_ann)
mes_eta3 <- stack_bmkDF(ts_to_bmkDF(ventes_ann2, ind_frequency = 4))
head(mes_eta3)
tail(mes_eta3)

res_eta3 <- benchmarking(mes_ind3,
                         mes_eta3,
                         rho = 0.729,
                         lambda = 1,
                         biasOption = 1,
                         var = "value / alter",
                         with = "value",
                         by = "series",
                         quiet = TRUE)

# Générerer les graphiques d'étalonnage
plot_graphTable(res_eta3$graphTable, "Graphs_ex3.pdf")

# Convertir le « data frame » `res_eta3$series` en un objet « mts »
ventes_tri2_eta <- tsDF_to_ts(unstack_tsDF(res_eta3$series), frequency = 4)

# Afficher les 10 premières observations
ts(ventes_tri2_eta[1:10, ], start = start(ventes_tri2), deltat = deltat(ventes_tri2))

# Vérifier la valeur des ventes de camionnettes pour 2012 T1 et T2
# dans le groupe A (valeurs fixes)
all.equal(window(ventes_tri2[, "A.camionnettes"], start = c(2012, 1), end = c(2012, 2)),
        window(ventes_tri2_eta[, "A.camionnettes"], start = c(2012, 1), end = c(2012, 2)))


# Réinitialiser le répertoire de travail à son emplacement initial
setwd(rep_ini)
```

---

bench_graphs *Générer un graphique d'étalonnage*

Description

Fonctions utilisées à l'interne par `plot_graphTable()` pour générer les graphiques d'étalonnage dans un fichier PDF :

- `ori_plot()`: Échelle originale (argument `ori_plot_flag = TRUE` de `plot_graphTable()`)
- `adj_plot()`: Échelle d'ajustement (argument `adj_plot_flag = TRUE` de `plot_graphTable()`)

- GR_plot(): Taux de croissance (argument GR_plot_flag = TRUE de plot_graphTable())
- GR_table(): Tableau des taux de croissance (argument GR_table_flag = TRUE de plot_graphTable())

Lorsque ces fonctions sont appelées directement, le *data frame* **graphTable** (argument graphTable) ne devrait contenir qu'une **série unique** et le graphique est généré dans le périphérique de graphiques courant (actif).

Usage

```
ori_plot(
  graphTable,
  title_str = "Original Scale",
  subtitle_str = NULL,
  mth_gap = NULL,
  points_set = NULL,
  pt_sz = 2,
  display_ggplot = TRUE,
  .setup = TRUE
)

adj_plot(
  graphTable,
  title_str = "Adjustment Scale",
  subtitle_str = NULL,
  mth_gap = NULL,
  full_set = NULL,
  pt_sz = 2,
  display_ggplot = TRUE,
  .setup = TRUE
)

GR_plot(
  graphTable,
  title_str = "Growth Rates",
  subtitle_str = NULL,
  factor = NULL,
  type_chars = NULL,
  periodicity = NULL,
  display_ggplot = TRUE,
  .setup = TRUE
)

GR_table(
  graphTable,
  title_str = "Growth Rates Table",
  subtitle_str = NULL,
  factor = NULL,
  type_chars = NULL,
  display_ggplot = TRUE,
```

```
  .setup = TRUE
)
```

Arguments

`graphTable`	(obligatoire) *Data frame*, ou objet compatible, correspondant au *data frame* de sortie **graphTable** de la fonction d'étalonnage.
`title_str`, `subtitle_str`	(optionnel) Chaînes de caractères spécifiant les titre et sous-titre du graphique. `subtitle_str` est construit automatiquement à partir du contenu du *data frame* `graphTable` lorsque `NULL` et contient le nom *data frame* `graphTable` sur la 2ème ligne et les paramètres d'étalonnage sur la 3ème ligne. La spécification de chaînes vides (`""`) supprimerait les titres. L'utilisation de syntaxe Markdown et HTML simple est permise (ex., pour l'affichage de caractères gras, italiques ou en couleur) grâce à l'utilisation à l'interne de la librairie ggtext (voir `help(package = "ggtext")`). **Les valeurs par défaut** sont `subtitle_str = NULL` et un titre propre à chaque fonction pour `title_str` (voir **Utilisation**).
`mth_gap`	(optionnel) Nombre de mois entre deux périodes consécutives (ex., 1 pour des données mensuelles, 3 pour des données trimestrielles, etc.). Basé sur le contenu du *data frame* `graphTable` lorsque `NULL` (calculé comme `12 / graphTable$periodicity[1]`). **La valeur par défaut** est `mth_gap = NULL`.
`points_set`, `full_set`	(optionnel) Vecteur de chaînes de caractères des éléments (variables du *data frame* `graphTable`) à inclure dans le graphique. Automatiquement construit lorsque `NULL`. Voir `plot_graphTable()` pour la liste des variables utilisées (par défaut) par chaque type de graphique. **Les valeurs par défaut** sont `points_set = NULL` et `full_set = NULL`.
`pt_sz`	(optionnel) Taille du pictogramme (symbole) des points de données pour ggplot2. **La valeur par défaut** est `pt_sz = 2`.
`display_ggplot`	(optionnel) Argument logique (*logical*) indiquant si l'object ggplot doit être affiché dans le périphérique de graphiques courant (actif). **La valeur par défaut** est `display_ggplot = TRUE`.
`.setup`	(optionnel) Argument logique indiquant si les étapes de configuration doivent être exécutées ou non. Doit être `TRUE` lorsque la fonction est appelée directement (c.-à-d., hors du contexte de `plot_graphTable()`). **La valeur par défaut** est `.setup = TRUE`.

`factor`, `type_chars`
: (optionnel)
Facteur de taux de croissance (1 ou 100) et suffixe de l'étiquette des valeurs (« » ou « (%) ») selon le paramètre du modèle d'ajustement λ. Basé sur le contenu du *data frame* `graphTable` lorsque `NULL` (basé sur `graphTable$lambda[1]`).
Les valeurs par défaut sont `factor` = `NULL` et `type_chars` = `NULL`.

`periodicity`
: (optionnel)
Le nombre de périodes dans une année. Basé sur le contenu du *data frame* `graphTable` lorsque `NULL` (défini comme `graphTable$periodicity[1]`).
La valeur par défaut est `periodicity` = `NULL`.

Details

Voir `plot_graphTable()` pour une description détaillée des quatre graphiques d'étalonnage associés à ces fonctions individuelles. Ces graphiques sont optimisés pour un format de papier Lettre US en orientation paysage, c.-à-d., 11po de large (27.9cm, 1056px avec 96 PPP) et 8.5po de haut (21.6cm, 816px avec 96 PPP). Gardez cela à l'esprit lorsque vous visualisez ou enregistrez des graphiques générés par des appels à ces fonctions individuelles (c.-à-d., hors du contexte de `plot_graphTable()`). Notez également que `GR_plot()` et `GR_table()` génèrent souvent plus d'un graphique (plus d'une *page*), à moins de réduire le nombre de périodes fournies en entrée dans le *data frame* `graphTable` (ex., en subdivisant le *data frame* par plages d'années civiles).

Value

En plus d'afficher le(s) graphique(s) correspondant(s) dans le périphérique de graphiques actif (sauf si `display_ggplot` = `FALSE`), chaque fonction renvoie également de manière invisible une liste contenant les objets ggplot générés. Notes :

- `ori_plot()` et `adj_plot()` génèrent un seul objet ggplot (un seul graphique) alors que `GR_plot()` et `GR_table()` génèrent souvent plusieurs objets ggplot (plusieurs graphiques).
- Les objets ggplot renvoyés peuvent être affichés *manuellement* avec `print()`, auquel cas il est suggéré d'apporter les mises à jour suivantes au thème ggplot2 (modifications utilisés à l'interne lorsque `display_ggplot` = `TRUE`) :

```
ggplot2::theme_update(
  plot.title = ggtext::element_markdown(hjust = 0.5),
  plot.subtitle = ggtext::element_markdown(hjust = 0.5),
  legend.position = "bottom",
 plot.margin = ggplot2::margin(t = 1.5, r = 1.5, b = 1.5, l = 1.5, unit = "cm"))
```

See Also

`plot_graphTable()` `plot_benchAdj()` `benchmarking()` `stock_benchmarking()`

Examples

```
# Désactiver la création du périphérique de graphiques pour la page de référence HTML
# du site web (non petinent dans ce contexte)
creer_grDev <- !(identical(Sys.getenv("IN_PKGDOWN"), "true"))
```

```
# Série chronologique trimestrielle initiale (série indicatrice à étalonner)
sc_tri <- ts(c(1.9, 2.4, 3.1, 2.2, 2.0, 2.6, 3.4, 2.4, 2.3),
             start = c(2015, 1), frequency = 4)

# Série chronologique annuelle (étalons)
sc_ann <- ts(c(10.3, 10.2), start = 2015, frequency = 1)


# Étalonnage proportionnel
res_eta <- benchmarking(ts_to_tsDF(sc_tri),
                        ts_to_bmkDF(sc_ann, ind_frequency = 4),
                        rho = 0.729, lambda = 1, biasOption = 3,
                        quiet = TRUE)


# Ouvrir un nouveau périphérique de graphiques de 11po de large et 8.5po de haut
# (format de papier Lettre US en orientation paysage)
if (creer_grDev) {
  dev.new(width = 11, height = 8.5, unit = "in", noRStudioGD = TRUE)
}

# Générer les graphiques d'étalonnage
ori_plot(res_eta$graphTable)
adj_plot(res_eta$graphTable)
GR_plot(res_eta$graphTable)
GR_table(res_eta$graphTable)


# Simuler l'étalonnage de plusieurs séries (3 séries de stocks)

sc_tri2 <- ts.union(ser1 = sc_tri, ser2 = sc_tri * 100, ser3 = sc_tri * 10)
sc_ann2 <- ts.union(ser1 = sc_ann, ser2 = sc_ann * 100, ser3 = sc_ann * 10)

# Avec l'argument `allCols = TRUE` (séries identifiées avec la colonne `varSeries`)
res_eta2 <- benchmarking(ts_to_tsDF(sc_tri2),
                         ts_to_bmkDF(sc_ann2, ind_frequency = 4),
                         rho = 0.729, lambda = 1, biasOption = 3,
                         allCols = TRUE,
                         quiet = TRUE)

# Graphiques « Échelle originale » et « Échelle d'ajustement » pour la 2ième série (ser2)
res_ser2 <- res_eta2$graphTable[res_eta2$graphTable$varSeries == "ser2", ]
ori_plot(res_ser2)
adj_plot(res_ser2)

# Avec l'argument `by = "series"` (séries identifiées avec la colonne `series`)
res_eta3 <- benchmarking(stack_tsDF(ts_to_tsDF(sc_tri2)),
                         stack_bmkDF(ts_to_bmkDF(sc_ann2, ind_frequency = 4)),
                         rho = 0.729, lambda = 1, biasOption = 3,
                         by = "series",
                         quiet = TRUE)
```

```
# Graphique des taux de croissance pour le 3ième séries (ser3)
res_ser3 <- res_eta3$graphTable[res_eta3$graphTable$series == "ser3", ]
GR_plot(res_ser3)


# Fermer le périphérique de graphiques
if (creer_grDev) {
  dev.off()
}
```

`build_balancing_problem` *Construire les éléments de base des problèmes d'équilibrage.*

Description

Cette fonction est utilisée à l'interne par `tsbalancing()` pour construire les éléments de base des problèmes d'équilibrage. Elle peut également être utile pour dériver manuellement les séries indirectes associées aux contraintes d'équilibrage d'égalité (en dehors du contexte de `tsbalancing()`).

Usage

```
build_balancing_problem(
  in_ts,
  problem_specs_df,
  in_ts_name = deparse1(substitute(in_ts)),
  ts_freq = stats::frequency(in_ts),
  periods = gs.time2str(in_ts),
  n_per = nrow(as.matrix(in_ts)),
  specs_df_name = deparse1(substitute(problem_specs_df)),
  temporal_grp_periodicity = 1,
  alter_pos = 1,
  alter_neg = 1,
  alter_mix = 1,
  lower_bound = -Inf,
  upper_bound = Inf,
  validation_only = FALSE
)
```

Arguments

`in_ts` (obligatoire)

Objet de type série chronologique (« ts » ou « mts »), ou objet compatible, qui contient les données des séries chronologiques à réconcilier. Il s'agit des données d'entrée (solutions initiales) des problèmes d'équilibrage (« *balancing* »).

`problem_specs_df`

(obligatoire)

Data frame des spécifications du problème d'équilibrage. En utilisant un format clairsemé (épars) inspiré de la procédure LP de SAS/OR® (SAS Institute 2015), il ne contient que les informations pertinentes telles que les coefficients non nuls des contraintes d'équilibrage ainsi que les coefficients d'altérabilité et les bornes inférieures/supérieures à utiliser au lieu des valeurs par défaut (c.-à-d., les valeurs qui auraient la priorité sur celles définies avec les arguments `alter_pos`, `alter_neg`, `alter_mix`, `alter_temporal`, `lower_bound` et `upper_bound`).

Les informations sont fournies à l'aide de quatre variables obligatoires (`type`, `col`, `row` et `coef`) et d'une variable facultative (`timeVal`). Un enregistrement (une rangée) dans le *data frame* des spécifications du problème définit soit une étiquette pour l'un des sept types d'éléments du problème d'équilibrage avec les colonnes `type` et `row` (voir *Enregistrements de définition d'étiquette* ci-dessous) ou bien spécifie des coefficients (valeurs numériques) pour ces éléments du problème d'équilibrage avec les variables `col`, `row`, `coef` et `timeVal` (voir *Enregistrements de spécification d'information* ci-dessous).

- **Enregistrements de définition d'étiquette** (`type` n'est pas manquant (n'est pas `NA`))
  - `type` (car) : mot-clé réservé identifiant le type d'élément du problème en cours de définition :
    - `EQ` : contrainte d'équilibrage d'égalité ($=$)
    - `LE` : contrainte d'équilibrage d'inégalité de type inférieure ou égale ($\leq$)
    - `GE` : contrainte d'équilibrage d'inégalité de type supérieure ou égale ($\geq$)
    - `lowerBd` : borne inférieure des valeurs de période
    - `upperBd` : borne supérieure des valeurs de période
    - `alter` : coefficient d'altérabilité des valeurs de période
    - `alterTmp` : coefficient d'altérabilité des totaux temporels
  - `row` (car) : étiquette à associer à l'élément du problème (*mot-clé* `type`)
  - *toutes les autres variables ne sont pas pertinentes et devraient contenir des données manquantes (valeurs* `NA`*)*

- **Enregistrements de spécification d'information** (`type` est manquant (est `NA`))
  - `type` (car) : non applicable (`NA`)
  - `col` (car) : nom de la série ou mot réservé `_rhs_` pour spécifier la valeur du côté droit (*RHS* pour **R***ight-***H***and* **S***ide*) d'une contrainte d'équilibrage.
  - `row` (car) : étiquette de l'élément du problème.
  - `coef` (num) : valeur de l'élément du problème :
    - coefficient de la série dans la contrainte d'équilibrage ou valeur *RHS*
    - borne inférieure ou supérieure des valeurs de période de la série

    * coefficient d'altérabilité des valeurs de période ou des totaux temporels de la série
  - `timeVal` (num) : valeur de temps optionnelle pour restreindre l'application des bornes ou coefficients d'altérabilité des séries à une période (ou groupe temporel) spécifique. Elle correspond à la valeur de temps, telle que renvoyée par `stats::time()`, pour une période (observation) donnée des séries chronologiques d'entrée (argument `in_ts`) et correspond conceptuellement à $année + (période - 1)/fréquence$.

Notez que les chaînes de caractères vides (`""` ou `''`) pour les variables de type caractère sont interprétées comme manquantes (`NA`) par la fonction. La variable `row` identifie les éléments du problème d'équilibrage et est la variable clé qui fait le lien entre les deux types d'enregistrements. La même étiquette (`row`) ne peut être associée à plus d'un type d'éléments du problème (`type`) et plusieurs étiquettes (`row`) ne peuvent pas être définies pour un même type d'éléments du problème donné (`type`), à l'exception des contraintes d'équilibrage (valeurs `"EQ"`, `"LE"` et `"GE"` de la colonne `type`). Voici certaines caractéristiques conviviales du *data frame* des spécifications du problème :

- L'ordre des enregistrements (rangées) n'est pas important.
- Les valeurs des variables de type caractère (`type`, `row` et `col`) ne sont pas sensibles à la casse (ex., les chaînes de caractères `"Constraint 1"` et `"CONSTRAINT 1"` pour la variable `row` seraient considérées comme une même étiquette d'élément du problème), sauf lorsque `col` est utilisé pour spécifier un nom de série (une colonne de l'objet d'entrée de type série chronologique) où **la sensibilité à la casse est appliquée**.
- Les noms des variables du *data frame* des spécifications du problème ne sont pas non plus sensibles à la casse (ex., `type`, `Type` ou `TYPE` sont tous des noms de variable valides) et `time_val` est un nom de variable accepté (au lieu de `timeVal`).

Enfin, le tableau suivant dresse la liste des alias valides (acceptés) pour les *mots-clés* `type` (type d'éléments du problème) :

Mot-clé	**Alias**
`EQ`	`==`, `=`
`LE`	`<=`, `<`
`GE`	`>=`, `>`
`lowerBd`	`lowerBound`, `lowerBnd`, + *mêmes termes avec '_', '.' ou ' ' entre les mots*
`upperBd`	`upperBound`, `upperBnd`, + *mêmes termes avec '_', '.' ou ' ' entre les mots*
`alterTmp`	`alterTemporal`, `alterTemp`, + *mêmes termes avec '_', '.' ou ' ' entre les mots*

L'examen des **Exemples** devrait aider à conceptualiser le *data frame* des spécifications du problème d'équilibrage.

`in_ts_name` (optional)

Chaîne de caractères contenant la valeur de l'argument `in_ts`.

La valeur par défaut est `in_ts_name = deparse1(substitute(in_ts))`.

`ts_freq` (optional)

Fréquence de l'object the type série chronologique (argument `in_ts`).
La valeur par défaut est `ts_freq = stats::frequency(in_ts)`.

`periods` (optional)
Vecteur de chaînes de caractères décrivant les périodes de l'object the type série chronologique (argument `in_ts`).
La valeur par défaut est `periods = gs.time2str(in_ts)`.

`n_per` (optional)
Nombre de périodes de l'object the type série chronologique (argument `in_ts`).
La valeur par défaut est `n_per = nrow(as.matrix(in_ts))`.

`specs_df_name` (optional)
Chaîne de caractères contenant la valeur de l'argument `problem_specs_df`.
La valeur par défaut est `specs_df_name = deparse1(substitute(problem_specs_df))`.

`temporal_grp_periodicity`
(optionnel)
Nombre entier positif définissant le nombre de périodes dans les groupes temporels pour lesquels les totaux doivent être préservés. Par exemple, spécifiez `temporal_grp_periodicity = 3` avec des séries chronologiques mensuelles pour la préservation des totaux trimestriels et `temporal_grp_periodicity = 12` (ou `temporal_grp_periodicity = frequency(in_ts)`) pour la préservation des totaux annuels. Spécifier `temporal_grp_periodicity = 1` (*défaut*) correspond à un traitement période par période sans préservation des totaux temporels.
La valeur par défaut est `temporal_grp_periodicity = 1` (traitement période par période sans préservation des totaux temporels).

`alter_pos` (optionnel)
Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut associé aux valeurs des séries chronologiques avec des coefficients **positifs** dans toutes les contraintes d'équilibrage dans lesquelles elles sont impliquées (ex., les séries composantes dans les problèmes de ratissage (« *raking* ») de tables d'agrégation). Les coefficients d'altérabilité fournis dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.
La valeur par défaut est `alter_pos = 1.0` (valeurs non contraignantes).

`alter_neg` (optionnel)
Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut associé aux valeurs des séries chronologiques avec des coefficients **négatifs** dans toutes les contraintes d'équilibrage dans lesquelles elles sont impliquées (ex., les séries de total de marge dans les problèmes de ratissage (« *raking* ») de tables d'agrégation). Les coefficients d'altérabilité fournis dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.
La valeur par défaut est `alter_neg = 1.0` (valeurs non contraignantes).

`alter_mix` (optionnel)
Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut associé aux valeurs des séries chronologiques avec un mélange de coefficients **positifs et négatifs** dans les contraintes d'équilibrage dans lesquelles elles sont impliquées. Les coefficients d'altérabilité fournis dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.
La valeur par défaut est `alter_mix = 1.0` (valeurs non contraignantes).

`lower_bound` (optionnel)

Nombre réel spécifiant la borne inférieure par défaut pour les valeurs des séries chronologiques. Les bornes inférieures fournies dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.

La valeur par défaut est `lower_bound = -Inf` (non borné).

`upper_bound` (optionnel)

Nombre réel spécifiant la borne supérieure par défaut pour les valeurs des séries chronologiques. Les bornes supérieures fournies dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.

La valeur par défaut est `upper_bound = Inf` (non borné).

`validation_only`

(optionnel)

Argument logique (*logical*) spécifiant si la fonction doit uniquement effectuer la validation des données d'entrée ou non. Lorsque `validation_only = TRUE`, les *contraintes d'équilibrage* et les *bornes (inférieures et supérieures) des valeurs de période* spécifiées sont validées par rapport aux données de séries chronologiques d'entrée, en permettant des écarts jusqu'à la valeur spécifiée avec l'argument `validation_tol`. Sinon, lorsque `validation_only = FALSE` (par défaut), les données d'entrée sont d'abord réconciliées et les données résultantes (en sortie) sont ensuite validées.

La valeur par défaut est `validation_only = FALSE`.

Details

Voir `tsbalancing()` pour une description détaillée des problèmes d'*équilibrage de séries chronologiques*.

Toute valeur manquante (`NA`) trouvée dans l'objet de série chronologique d'entrée (argument `in_ts`) serait remplacée par 0 dans `values_ts` et déclencherait un message d'avertissement.

Les éléments renvoyés des des problèmes d'équilibrage n'incluent pas les totaux temporels implicites (c.-à-d., les éléments `A2`, `op2` et `b2` ne contiennent que les contraintes d'équilibrage).

Les éléments `A2`, `op2` et `b2` d'un problème d'équilibrage impliquant plusieurs périodes (lorsque `temporal_grp_periodicity > 1`) sont construits *colonne par colonne* (selon le principe « column-major order » en anglais), ce qui correspond au comportement par défaut de R lors de la conversion d'objets de la classe « matrix » en vecteurs. Autrement dit, les contraintes d'équilibrage correspondent conceptuellement à :

- `A1 %*% values_ts[t, ] op1 b1` pour des problèmes impliquant une seule période (`t`)
- `A2 %*% as.vector(values_ts[t1:t2, ]) op2 b2` pour des problèmes impliquant `temporal_grp_periodicity` périodes (`t1:t2`)

Notez que l'argument `alter_temporal` n'a pas encore été appliqué à ce stade et que `altertmp$coefs_ts` ne contient que les coefficients spécifiés dans le *data frame* des spécifications du problème (argument `problem_specs_df`). Autrement dit, `altertmp$coefs_ts` contient des valeurs manquantes (`NA`) à l'exception des coefficients d'altérabilité de total temporel inclus dans (spécifiés avec) `problem_specs_df`. Ceci est fait afin de faciliter l'identification du premier coefficient d'altérabilité non manquant (non `NA`) de chaque groupe temporel complet (à survenir ultérieurement, le cas échéant, dans `tsbalancing()`).

Value

Une liste avec les éléments des problèmes d'équilibrage (excluant l'information sur les totaux temporels) :

- `labels_df` : version nettoyée des *enregistrements de définition d'étiquette* provenant de `problem_specs_df` (enregistrements où `type` n'est pas manquant (n'est pas `NA`)); colonnes supplémentaires :
  - `type.lc` : `tolower(type)`
  - `row.lc` : `tolower(row)`
  - `con.flag` : `type.lc %in% c("eq", "le", "ge")`
- `coefs_df` : version nettoyée des *enregistrements de spécification d'information* provenant de `problem_specs_df` (enregistrements où `type` est manquant (est `NA`)); colonnes supplémentaires :
  - `row.lc` : `tolower(row)`
  - `con.flag` : `labels_df$con.flag` attribuée à travers `row.lc`
- `values_ts`: version réduite de `in_ts` avec seulement les séries pertinentes (voir vecteur `ser_names`)
- `lb` : information sur les bornes inférieures (`type.lc = "lowerbd"`) des séries pertinentes; liste avec les éléments suivants :
  - `coefs_ts` : object « mts » contenant les bornes inférieures des séries pertientes (voir vecteur `ser_names`)
  - `nondated_coefs` : vecteur des bornes non datées de `problem_specs_df` (`timeVal` est `NA`)
  - `nondated_id_vec` : vecteur d'identificateurs de `ser_names` associés au vecteur `nondated_coefs`
  - `dated_id_vec` : vecteur d'identificateurs de `ser_names` associés aux bornes inférieures datées de `problem_specs_df` (`timeVal` n'est pas `NA`)
- `ub` : équivalent de `lb` pour les bornes supérieures (`type.lc = "upperbd"`)
- `alter` : équivalent de `lb` pour les coefficients d'altérabilité des valeurs de période (`type.lc = "alter"`)
- `altertmp` : équivalent de `lb` pour les coefficients d'altérabilité des totaux temporels (`type.lc = "altertmp"`)
- `ser_names` : vecteur des noms de séries pertinentes (ensemble de séries impliquées dans les contraintes d'équilibrage)
- `pos_ser` : vecteur des noms de séries qui n'ont que des coefficients non nuls positifs à travers toutes les contraintes
- `neg_ser` : vecteur des noms de séries qui n'ont que des coefficients non nuls négatifs à travers toutes les contraintes
- `mix_ser` : vecteur des noms de séries qui ont des coefficients non nuls positifs et négatifs à travers toutes les contraintes
- `A1`,`op1`,`b1` : éléments des contraintes d'équilibrage pour les problèmes impliquant une seule période (ex., chacune des périodes d'un groupe temporel incomplet)
- `A2`,`op2`,`b2` : éléments des contraintes d'équilibrage pour les problèmes impliquant `temporal_grp_periodicity` périodes (ex., l'ensemble des périodes d'un groupe temporel complet)

See Also

tsbalancing() build_raking_problem()

Examples

```
##############################################################################################
#  Cadre de dérivation des séries indirectes avec les métadonnées de `tsbalancing()`
##############################################################################################
#
# Il est supposé (convenu) que...
#
# a) Toutes les contraintes d'équilibrage sont des contraintes d'égalité (`type = EQ`).
# b) Toutes les contraintes n'ont qu'une seule série non contraignante (libre) : la
#    série à dériver (c.-à-d., toutes les séries ont un coef. d'alt. de 0 sauf la
#    série à dériver).
# c) Chaque contrainte dérive une série différente (une nouvelle série).
# d) Les contraintes sont les mêmes pour toutes les périodes (c.-à-d., il n'y a pas
#    de coef. d'alt. « datés » spécifiés à l'aide de la colonne `timeVal`).
##############################################################################################


# Dériver les totaux de marge d'un cube de données à deux dimensions (2 x 3) en
# utilisant les métadonnées de `tsbalancing()` (les contraintes d'agrégation d'un
# cube de données respectent les hypothèses ci-dessus).


# Construire les spécifications du problème d'équilibrage à travers les métadonnées
# (plus simples) de ratissage.
mes_specs <- rkMeta_to_blSpecs(
  data.frame(series = c("A1", "A2", "A3",
                        "B1", "B2", "B3"),
             total1 = c(rep("totA", 3),
                        rep("totB", 3)),
             total2 = rep(c("tot1", "tot2", "tot3"), 2)),
  alterSeries = 0,  # séries composantes contraignantes (fixes)
  alterTotal1 = 1,  # totaux de marge non contraignants (libres, à dériver)
  alterTotal2 = 1)  # totaux de marge non contraignants (libres, à dériver)
mes_specs

# 6 périodes (trimestres) de données avec totaux de marge initialisés à zéro (0): ces
# derniers doivent OBLIGATOIREMENT exister dans les données d'entrée ET contenir des
# données valides (non `NA`).
mes_series <- ts(data.frame(A1 = c(12, 10, 12,  9, 15,  7),
                            B1 = c(20, 21, 15, 17, 19, 18),
                            A2 = c(14,  9,  8,  9, 11, 10),
                            B2 = c(20, 29, 20, 24, 21, 17),
                            A3 = c(13, 15, 17, 14, 16, 12),
                            B3 = c(24, 20, 30, 23, 21, 19),
                            tot1 = rep(0, 6),
                            tot2 = rep(0, 6),
                            tot3 = rep(0, 6),
                            totA = rep(0, 6),
```

```
                            totB = rep(0, 6)),
                  start = 2019, frequency = 4)

# Obtenir les éléments du problème d'équilibrage.
n_per <- nrow(mes_series)
p <- build_balancing_problem(mes_series, mes_specs,
                             temporal_grp_periodicity = n_per)

# `A2`, `op2` et `b2` définissent 30 constraintes (5 totaux de marge X 6 périodes)
# impliquant un total de 66 points de données (11 séries X 6 périodes) desquels 36
# réfèrent aux 6 séries composantes et 30 réfèrent aux 5 totaux de marge.
dim(p$A2)

# Obtenir les noms des totaux de marge (séries avec un coef. d'alt. non nul), dans
# l'ordre où les contraintes correspondantes apparaissent dans les spécifications
# (ordre de spécification des constraintes).
tmp <- p$coefs_df$col[p$coefs_df$con.flag]
noms_tot <- tmp[tmp %in% p$ser_names[p$alter$nondated_id_vec[p$alter$nondated_coefs != 0]]]

# Définir des drapeaux logiques identifiant les colonnes de total de marge :
# - `col_tot_logi1` : éléments à période unique (de longueur 11 = nombre de séries)
# - `col_tot_logi2` : éléments multi-périodes (de longueur 66 = nombre de points de
#                     données), selon le principe « column-major order » en anglais
#                     (l'ordre de construction des éléments de la matrice `A2`)
col_tot_logi1 <- p$ser_names %in% noms_tot
col_tot_logi2 <- rep(col_tot_logi1, each = n_per)

# Ordre des totaux de marge à dériver selon
# ... les colonnes des données d'entrée (objet « mts » `mes_series`)
p$ser_names[col_tot_logi1]
# ... la spécification des contraintes (« data frame » `mes_specs`)
noms_tot


# Calculer les 5 totaux de marge pour les 6 périodes.
# Note : le calcul suivant prend en compte les contraintes d'égalité linéaires
#        générales, c.-à-d.,
#        a) des valeurs non nulles du côté droit des contraintes (`b2`) et
#        b) des coefficients de contrainte non nuls autres que 1 pour les séries
#           composantes et -1 pour la série à dériver.
mes_series[, noms_tot] <- {
  (
    # Côté droit des contraintes
    p$b2 -

    # Sommes des composantes (« pondérées » par les coefficients des contraintes)
    p$A2[, !col_tot_logi2, drop = FALSE] %*% as.vector(p$values_ts[, !col_tot_logi1])
  ) /

  # Coefficients des séries dérivées : `t()` permet une recherche « par ligne » dans
  # la matrice `A2` (c.-à-d., selon l'ordre de spécification des constraintes)
  # Note: `diag(p$A2[, tot_col_logi2])` fonctionnerait si `p$ser_names[col_tot_logi1]`
  #       et `noms_tot` étaient identiques (même ordre pour les totaux); par contre,
```

```
  #       la recherche « par ligne » ci-dessous fonctionnera toujours (et est
  #       nécessaire dans le cas qui nous concerne).
  t(p$A2[, col_tot_logi2])[t(p$A2[, col_tot_logi2]) != 0]
}
mes_series
```

`build_raking_problem` *Construire les éléments du problème de ratissage.*

Description

Cette fonction est utilisée à l'interne par `tsraking()` pour construire les éléments du problème de ratissage. Elle peut également être utile pour dériver manuellement les totaux transversaux (des marges) du problème de ratissage (en dehors du contexte de `tsraking()`).

Usage

```
build_raking_problem(
  data_df,
  metadata_df,
  data_df_name = deparse1(substitute(data_df)),
  metadata_df_name = deparse1(substitute(metadata_df)),
  alterability_df = NULL,
  alterSeries = 1,
  alterTotal1 = 0,
  alterTotal2 = 0
)
```

Arguments

`data_df` (obligatoire)

Data frame, ou objet compatible, qui contient les données des séries chronologiques à réconcilier. Il doit au minimum contenir des variables correspondant aux séries composantes et aux totaux de contrôle transversaux spécifiés dans le *data frame* des métadonnées de ratissage (argument `metadata_df`). Si plus d'un enregistrement (plus d'une période) est fournie, la somme des valeurs des séries composantes fournies sera également préservée à travers des contraintes temporelles implicites.

`metadata_df` (obligatoire)

Data frame, ou objet compatible, qui décrit les contraintes d'agrégation transversales (règles d'additivité) pour le problème de ratissage (« *raking* »). Deux variables de type caractère doivent être incluses dans le *data frame* : `series` et `total1`. Deux variables sont optionnelles : `total2` (caractère) et `alterAnnual` (numérique). Les valeurs de la variable `series` représentent les noms des variables des séries composantes dans le *data frame* des données d'entrée (argument `data_df`). De même, les valeurs des variables `total1` et `total2` représentent les

noms des variables des totaux de contrôle transversaux de 1ère et 2ème dimension dans le *data frame* des données d'entrée. La variable `alterAnnual` contient le coefficient d'altérabilité pour la contrainte temporelle associée à chaque série composante. Lorsqu'elle est spécifiée, cette dernière remplace le coefficient d'altérabilité par défaut spécifié avec l'argument `alterAnnual`.

`data_df_name` (optionnel)

Chaîne de caractères contenant la valeur de l'argument `data_df`.

La valeur par défaut est `data_df_name = deparse1(substitute(data_df))`.

`metadata_df_name`

(optionnel)

Chaîne de caractères contenant la valeur de l'argument `metadata_df`.

La valeur par défaut est `data_df_name = deparse1(substitute(metadata_df))`.

`alterability_df`

(optionnel)

Data frame, ou objet compatible, ou `NULL`, qui contient les variables de coefficients d'altérabilité. Elles doivent correspondre à une série composante ou à un total de contrôle transversal, c'est-à-dire qu'une variable portant le même nom doit exister dans le *data frame* des données d'entrée (argument `data_df`). Les valeurs de ces coefficients d'altérabilité remplaceront les coefficients d'altérabilité par défaut spécifiés avec les arguments `alterSeries`, `alterTotal1` et `alterTotal2`. Lorsque le *data frame* des données d'entrée contient plusieurs enregistrements et que le *data frame* des coefficients d'altérabilité n'en contient qu'un seul, les coefficients d'altérabilité sont utilisés (répétés) pour tous les enregistrements du *data frame* des données d'entrée. Le *data frame* des coefficients d'altérabilité peut également contenir autant d'enregistrements que le *data frame* des données d'entrée.

La valeur par défaut est `alterability_df = NULL` (coefficients d'altérabilité par défaut).

`alterSeries` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les valeurs des séries composantes. Il s'appliquera aux séries composantes pour lesquelles des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterSeries = 1.0` (valeurs des séries composantes non contraignantes).

`alterTotal1` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les totaux de contrôle transversaux de la 1ère dimension. Il s'appliquera aux totaux de contrôle transversaux pour lesquels des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterTotal1 = 0.0` (totaux de contrôle transversaux de 1ère dimension contraignants).

`alterTotal2` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les totaux de contrôle transversaux de la 2ème dimension. Il s'appliquera aux

totaux de contrôle transversaux pour lesquels des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterTotal2 = 0.0` (totaux de contrôle transversaux de 2ème dimension contraignants).

Details

Voir `tsraking()` pour une description détaillée des problèmes de *ratissage de séries chronologiques*.

Les éléments du problème de ratissage renvoyés n'incluent pas les totaux temporels implicites des séries de composantes, le cas échéant (c.-à-d., les éléments `g` et `G` ne contiennent que l'information sur les totaux transversaux).

Lorsque les données d'entrée contiennent plusieurs périodes (scénario de préservation des totaux temporels), les éléments `x`, `c_x`, `g`, `c_g` et `G` du problème de ratissage sont construits *colonne par colonne* (selon le principe « column-major order » en anglais), ce qui correspond au comportement par défaut de R lors de la conversion d'objets de la classe « matrix » en vecteurs.

Value

Une liste avec les éléments du problème de ratissage (excluant les totaux temporels implicites) :

- `x` : vecteur des valeurs initiales des séries composantes
- `c_x` : vecteur des coefficients d'altérabilité des séries composantes
- `comp_cols` : vecteur des noms des séries composantes (colonnes de `data_df`)
- `g` : vecteur des valeurs initiales des totaux transversaux
- `c_g` : vecteur des coefficients d'altérabilité des totaux transversaux
- `tot_cols` : vecteur des noms des totaux transversaux (colonnes de `data_df`)
- `G` : matrice d'agrégation des totaux transversaux (`g = G %*% x`)

See Also

`tsraking()` `build_balancing_problem()`

Examples

```
# Dériver les totaux de marge d'un cube de données à deux dimensions (2 x 3)
# en utilisant les métadonnées de `tsraking()`.

mes_meta <- data.frame(series = c("A1", "A2", "A3",
                                  "B1", "B2", "B3"),
                       total1 = c(rep("totA", 3),
                                  rep("totB", 3)),
                       total2 = rep(c("tot1", "tot2", "tot3"), 2))
mes_meta

# 6 périodes de données avec totaux de marge initialisés à `NA` (ces derniers doivent
# OBLIGATOIREMENT exister dans les données d'entrée mais peuvent être `NA`).
mes_series <- data.frame(A1 = c(12, 10, 12,  9, 15,  7),
```

```
                          B1 = c(20, 21, 15, 17, 19, 18),
                          A2 = c(14,  9,  8,  9, 11, 10),
                          B2 = c(20, 29, 20, 24, 21, 17),
                          A3 = c(13, 15, 17, 14, 16, 12),
                          B3 = c(24, 20, 30, 23, 21, 19),
                          tot1 = rep(NA, 6),
                          tot2 = rep(NA, 6),
                          tot3 = rep(NA, 6),
                          totA = rep(NA, 6),
                          totB = rep(NA, 6))

# Obtenir les éléments du problème de ratissage.
p <- build_raking_problem(mes_series, mes_meta)
str(p)

# Calculer les 5 totaux de marge pour les 6 périodes.
mes_series[p$tot_cols] <- p$G %*% p$x
mes_series
```

`gs.build_proc_grps` *Construire des groupes de traitement de réconciliation*

Description

Cette fonction construit le *data frame* des groupes de traitement pour les problèmes de réconciliation. Elle est utilisée à interne par `tsraking_driver()` et `tsbalancing()`.

Usage

```
gs.build_proc_grps(
  ts_yr_vec,
  ts_per_vec,
  n_per,
  ts_freq,
  temporal_grp_periodicity,
  temporal_grp_start
)
```

Arguments

`ts_yr_vec`	(obligatoire) Vecteur des valeurs d'année (unité de temps; voir `gs.time2year()`).
`ts_per_vec`	(obligatoire) Vecteur des valeurs de période (cycle; voir `gs.time2per()`).
`n_per`	(obligatoire) Longueur (nombre de périodes) de la série chronologique.
`ts_freq`	(obligatoire) Fréquence de la srie chronologique (voir `stats::frequency()`).
`temporal_grp_periodicity`	(obligatoire) Nombre de périodes dans les groupes temporels.
`temporal_grp_start`	(obligatoire) Première période des groupes temporels.

Value

Un *data frame* avec les variables (colonnes) suivantes :

- `grp` : vecteur de nombres entiers identifiant le groupe de traitement (1 .. < nombre de groupes >)
- `beg_per` : vecteur de nombres entiers identifiant la première période du groupe de traitement (1 .. `n_per`)
- `end_per` : vecteur de nombres entiers identifiant la dernière période du groupe de traitement (1 .. `n_per`)
- `complete_grp`: Vecteur logique indiquant si le groupe de traitement correspond à un groupe temporel complet

Groupes de traitement

L'ensemble des périodes d'un problème de réconciliation (ratissage ou équilibrage) donné est appelé *groupe de traitement* et correspond soit :

- à une **période unique** lors d'un traitement période par période ou, lorsque les totaux temporels sont préservés, pour les périodes individuelles d'un groupe temporel incomplet (ex., une année incomplète)
- ou à l'**ensemble des périodes d'un groupe temporel complet** (ex., une année complète) lorsque les totaux temporels sont préservés.

Le nombre total de groupes de traitement (nombre total de problèmes de réconciliation) dépend de l'ensemble de périodes des séries chronologiques d'entrée (objet de type série chronologique spécifié avec l'argument `in_ts`) et de la valeur des arguments `temporal_grp_periodicity` et `temporal_grp_start`.

Les scénarios courants incluent `temporal_grp_periodicity = 1` (par défaut) pour un traitement période par période sans préservation des totaux temporels et `temporal_grp_periodicity = frequency(in_ts)` pour la préservation des totaux annuels (années civiles par défaut). L'argument `temporal_grp_start` permet de spécifier d'autres types d'années (*non civile*). Par exemple, des années financières commençant en avril correspondent à `temporal_grp_start = 4` avec des données mensuelles et à `temporal_grp_start = 2` avec des données trimestrielles. La préservation des totaux trimestriels avec des données mensuelles correspondrait à `temporal_grp_periodicity = 3`.

Par défaut, les groupes temporels convrant plus d'une année (c.-à-d., correspondant à `temporal_grp_periodicity > frequency(in_ts)`) débutent avec une année qui est un multiple de `ceiling(temporal_grp_periodicity / frequency(in_ts))`. Par exemple, les groupes bisannuels correspondant à `temporal_grp_periodicity = 2 * frequency(in_ts)` débutent avec une *année paire* par défaut. Ce comportement peut être modifié avec l'argument `temporal_grp_start`. Par exemple, la préservation des totaux bisannuels débutant avec une *année impaire* au lieu d'une *année paire* (par défaut) correspond à `temporal_grp_start = frequency(in_ts) + 1` (avec `temporal_grp_periodicity = 2 * frequency(in_ts)`).

Voir les **Exemples** de `gs.build_proc_grps()` pour des scénarios courants de groupes de traitements.

See Also

tsraking_driver() tsbalancing() time_values_conv

Examples

```
#######
# Configuration préalable

# Série chronologique mensuelle et trimestrielle « bidon » (2.5 années de longueur)
sc_men <- ts(rep(NA, 30), start = c(2019, 1), frequency = 12)
sc_men
sc_tri <- ts(rep(NA, 10), start = c(2019, 1), frequency = 4)
sc_tri

# Information résumée de la série chronologique
ts_info <- function(sc, sep = "-") {
  list(a = gs.time2year(sc),      # années
       p = gs.time2per(sc),       # périodes
       n = length(sc),            # longueur
       f = frequency(sc),         # fréquence
       e = gs.time2str(sc, sep))  # étiquettes
}
info_men <- ts_info(sc_men)
info_tri <- ts_info(sc_tri, sep = "t")

# Fonction qui ajoute une étiquette décrivant le groupe de traitement
ajouter_desc <- function(df_gr, vec_eti, mot, suf = "s") {
  df_gr$description <- ifelse(df_gr$complete_grp,
                      paste0("--- ", df_gr$end_per - df_gr$beg_per + 1, " ", mot, suf, " : ",
                                      vec_eti[df_gr$beg_per], " à ",
                                      vec_eti[df_gr$end_per], " ---"),
                             paste0("--- 1 ", mot, " : ", vec_eti[df_gr$beg_per], " ---"))
  df_gr
}




#######
# Scénarios courants de groupes de traitement pour des données mensuelles


# 0- Traitement mois par mois (chaque mois est un groupe de traitement)
gr_men0 <- gs.build_proc_grps(info_men$a, info_men$p, info_men$n, info_men$f,
                              temporal_grp_periodicity = 1,
                              temporal_grp_start = 1)
tmp <- ajouter_desc(gr_men0, info_men$e, "mois", "")
head(tmp)
tail(tmp)


# Groupes temporels correspondant à ...
```

```
# 1- des années civiles
gr_men1 <- gs.build_proc_grps(info_men$a, info_men$p, info_men$n, info_men$f,
                              temporal_grp_periodicity = 12,
                              temporal_grp_start = 1)
ajouter_desc(gr_men1, info_men$e, "mois", "")

# 2- des années financières commençant en avril
gr_men2 <- gs.build_proc_grps(info_men$a, info_men$p, info_men$n, info_men$f,
                              temporal_grp_periodicity = 12,
                              temporal_grp_start = 4)
ajouter_desc(gr_men2, info_men$e, "mois", "")

# 3- des trimestres réguliers (commençant en janvier, avril, juillet et octobre)
gr_men3 <- gs.build_proc_grps(info_men$a, info_men$p, info_men$n, info_men$f,
                              temporal_grp_periodicity = 3,
                              temporal_grp_start = 1)
ajouter_desc(gr_men3, info_men$e, "mois", "")

# 4- des trimestres décalés d'un mois (commençant en février, mai, août et novembre)
gr_men4 <- gs.build_proc_grps(info_men$a, info_men$p, info_men$n, info_men$f,
                              temporal_grp_periodicity = 3,
                              temporal_grp_start = 2)
ajouter_desc(gr_men4, info_men$e, "mois", "")




#######
# Scénarios courants de groupes de traitement pour des données trimestrielles


# 0- Traitement trimestre par trimestre (chaque trimestre est un groupe de traitement)
gr_tri0 <- gs.build_proc_grps(info_tri$a, info_tri$p, info_tri$n, info_tri$f,
                              temporal_grp_periodicity = 1,
                              temporal_grp_start = 1)
ajouter_desc(gr_tri0, info_tri$e, "trimestre")


# Groupes temporels correspondant à ...

# 1- des années civiles
gr_tri1 <- gs.build_proc_grps(info_tri$a, info_tri$p, info_tri$n, info_tri$f,
                              temporal_grp_periodicity = 4,
                              temporal_grp_start = 1)
ajouter_desc(gr_tri1, info_tri$e, "trimestre")

# 2- des années financières commençant en avril (2ième trimestre)
gr_tri2 <- gs.build_proc_grps(info_tri$a, info_tri$p, info_tri$n, info_tri$f,
                              temporal_grp_periodicity = 4,
                              temporal_grp_start = 2)
ajouter_desc(gr_tri2, info_tri$e, "trimestre")
```

gs.gInv_MP *Inverse de Moore-Penrose*

Description

Cette fonction calcule l'inverse (pseudo inverse) de Moore-Penrose d'une matrice carrée ou rectangulaire en utilisant la décomposition en valeurs singulières (SVD, de l'anglais *singular value decomposition*). Elle est utlilisée à l'interne par `tsraking()` et `benchmarking()`.

Usage

```
gs.gInv_MP(X, tol = NA)
```

Arguments

`X`	(mandatory) Matrice à inverser.
`tol`	(optional) Nombre réel qui spécifie la tolérance pour l'identification des valeurs singulières nulles. Lorsque `tol = NA` (par défaut), la tolérance est calculée comme étant le produit de la taille (dimension) de la matrice, de la norme de la matrice (plus grande valeur singulière) et de l'*epsilon de la machine* (`.Machine$double.eps`). **Default value** is `tol = NA`.

Details

La tolérance utilisée par défaut (argument `tol = NA`) est cohérente avec la tolérance utilisée par les logiciels MATLAB et GNU Octave dans leurs fonctions inverses générales. Lors de nos tests, cette tolérance par défaut a également produit des solutions (résultats) comparables à G-Series 2.0 en SAS®.

Value

L'inverse (pseudo inverse) de Moore-Penrose de la matrice `X`.

See Also

`tsraking()` `benchmarking()`

Examples

```
# Matrice inversible
X1 <- matrix(c(3, 2, 8,
               6, 3, 2,
               5, 2, 4), nrow = 3, byrow = TRUE)
Y1 <- gs.gInv_MP(X1)
all.equal(Y1, solve(X1))
X1 %*% Y1
```

```
# Matrice rectangulaire
X2 <- X1[-1, ]
try(solve(X2))
X2 %*% gs.gInv_MP(X2)

# Matrice carrée non inversible
X3 <- matrix(c(3, 0, 0,
               0, 0, 0,
               0, 0, 4), nrow = 3, byrow = TRUE)
try(solve(X3))
X3 %*% gs.gInv_MP(X3)
```

`osqp_settings_sequence` Data frame *pour la séquence de paramètres d'OSQP*

Description

Data frame contenant une séquence de paramètres d'OSQP pour `tsbalancing()` spécifié avec l'argument `osqp_settings_df`. La librairie inclut deux *data frames* prédéfinis de séquences de paramètres d'OSQP :

- `default_osqp_sequence` : rapide et efficace (valeur par défaut de l'argument `osqp_settings_df`);
- `alternate_osqp_sequence` : orienté vers la précision au détriment du temps d'exécution.

Voir `vignette("osqp-settings-sequence-dataframe")` pour le contenu de ces *data frames*.

Usage

```
# Séquence par défaut :
# tsbalancing(..., osqp_settings_df = default_osqp_sequence)

# Séquence alternative (plus lente) :
# tsbalancing(..., osqp_settings_df = alternate_osqp_sequence)

# Séquence personnalisée (sur mesure) :
# tsbalancing(..., osqp_settings_df = <my-osqp-sequence-data-frame>)

# Séquence unique avec paramètres par défaut d'OSQP (déconseillé !):
# tsbalancing(..., osqp_settings_df = NULL)
```

Format

Un *data frame* avec au moins un enregistrement (une rangée) et au moins une colonne, les colonnes *les plus courantes* étant :

max_iter Nombre maximal d'itérations (*integer*)

sigma Pas sigma (*sigma step*) de la méthode des multiplicateurs à direction alternée (MMDA, ou ADMM en anglais pour *alternating direction method of multipliers*) (*double*)

eps_abs Tolérance absolue (*double*)

eps_rel Tolérance relative (*double*)

eps_prim_inf Tolérance d'infaisabilité du problème primal (*double*)

eps_dual_inf Tolérance d'infaisabilité du problème dual (*double*)

polish Effectuer l'étape de raffinement de la solution (*logical*)

scaling Nombre d'itérations de mise à l'échelle (*integer*)

prior_scaling Mise à l'échelle préalable des données, avant la résolution avec OSQP (*logical*)

require_polished Exiger une solution raffinée (*polished solution*) pour arrêter la séquence (*logical*)

[*any-other-OSQP-setting*] Valeur du paramètre OSQP correspondant

Details

À l'exception de `prior_scaling` et `require_polished`, toutes les colonnes du *data frame* doivent correspondre à un paramètre d'OSQP. Les valeurs par défaut d'OSQP sont utilisées pour tout paramètre non spécifié dans ce *data frame*. Visitez https://osqp.org/docs/interfaces/solver_settings.html pour connaître tous les paramètres d'OSQP disponibles. Notez que le paramètre d'OSQP `verbose` est en fait contrôlé par les arguments `quiet` et `display_level` de `tsbalancing()` (c'est à dire que la colonne `verbose` dans un *data frame pour la séquence de paramètres d'OSQP* serait ignorée).

Chaque enregistrement (rangée) d'un *data frame pour la séquence de paramètres d'OSQP* représente une tentative de résolution d'un problème d'équilibrage avec les paramètres d'OSQP correspondants. La séquence de résolution s'arrête dès qu'une solution valide est obtenue (une solution pour laquelle tous les écarts de contraintes sont inférieurs ou égaux à la tolérance spécifiée avec l'argument `validation_tol` de `tsbalancing()`) à moins que la colonne `require_polished = TRUE`, auquel cas une solution *raffinée* d'OSQP (`status_polish = 1`) serait également nécessaire pour arrêter la séquence. Les écarts de contraintes correspondent à $\max(0, l - Ax, Ax - u)$ avec des contraintes définies comme $l \leq Ax \leq u$. Dans le cas où une solution satisfaisante ne peut être obtenue après avoir parcouru toute la séquence, `tsbalancing()` renvoie la solution qui a généré le plus petit total d'écarts de contraintes parmi les solutions valides, le cas échéant, ou parmi toutes les solutions, dans le cas contraire. Notez que l'exécution de la séquence de résolution entière peut être *forcée* en spécifiant l'argument `full_sequence = TRUE` avec `tsbalancing()`. Les enregistrements avec la colonne `prior_scaling = TRUE` ont les données du problème mises à l'échelle avant la résolution avec OSQP, en utilisant la moyenne des valeurs libres (non contraignantes) du problème comme facteur d'échelle.

En plus de spécifier un *data frame pour la séquence de paramètres d'OSQP* personnalisé avec l'argument `osqp_settings_df`, on peut aussi spécifier `osqp_settings_df = NULL` ce qui résultera en une seule tentative de résolution avec les valeurs par défaut d'OSQP pour tous les paramètres et avec `prior_scaling = FALSE` et `require_polished = FALSE`. Il est cependant recommandé d'essayer d'abord les *data frames* `default_osqp_sequence` et `alternate_osqp_sequence`, avec `full_sequence = TRUE` si nécessaire, avant d'envisager d'autres alternatives.

La vignette *« Data frame » pour la séquence de paramètres d'OSQP* (`vignette("osqp-settings-sequence-dataframe")`) contient des informations supplémentaires.

plot_benchAdj *Tracer les ajustements d'étalonnage*

Description

Tracer les ajustements d'étalonnage pour une série unique dans le périphérique graphique courant (actif). Il est possible de superposer jusqu'à trois types d'ajustements dans le même graphique :

- Ajustements générés par la fonction `benchmarking()`
- Ajustements générés par la fonction `stock_benchmarking()`
- Spline cubique associée aux ajustements générés par la fonction `stock_benchmarking()`

Ces graphiques peuvent être utiles pour évaluer la qualité des résultats d'étalonnage et comparer les ajustements générés par les deux fonctions d'étalonnage (`benchmarking()` et `stock_benchmarking()`) pour des séries de stocks.

Usage

```
plot_benchAdj(
  PB_graphTable = NULL,
  SB_graphTable = NULL,
  SB_splineKnots = NULL,
  legendPos = "bottomright"
)
```

Arguments

`PB_graphTable` (optionnel)

Data frame, ou objet compatible, correspondant au *data frame* de sortie `graphTable` de la fonction `benchmarking()` (PB pour approche « Proc Benchmarking »). Spécifiez `NULL` pour ne pas inclure les ajustements de `benchmarking()` dans le graphique.

La valeur par défaut est `PB_graphTable = NULL`.

`SB_graphTable` (optionnel)

Data frame, ou objet compatible, correspondant au *data frame* de sortie `graphTable` de la fonction `stock_benchmarking()` (SB). Spécifiez `NULL` pour ne pas inclure les ajustements de `stock_benchmarking()` dans le graphique.

La valeur par défaut est `SB_graphTable = NULL`.

`SB_splineKnots` (optionnel)

Data frame, ou objet compatible, correspondant au *data frame* de sortie `splineKnots` de la fonction `stock_benchmarking()` (SB). Spécifiez `NULL` pour ne pas inclure la spline cubique de `stock_benchmarking()` dans le graphique.

La valeur par défaut est `SB_splineKnots = NULL`.

`legendPos` (optionnel)

Chaîne de caractères (mot-clé) spécifiant l'emplacement de la légende dans le graphique. Voir la description de l'argument `x` dans la documentation de `graphics::legend()` pour la liste des mots-clés valides. Spécifiez `NULL` pour ne pas inclure de légende dans le graphique.

La valeur par défaut est `legendPos = "bottomright"` (en bas à droite).

Details

Variables du *data frame* `graphTable` (arguments `PB_graphTable` et `SB_graphTable`) utilisées dans le graphique :

- `t` pour l'axe des x (t)
- `benchmarkedSubAnnualRatio` pour les lignes *Stock Bench. (SB)* et *Proc Bench. (PB)*
- `bias` pour la ligne *Bias* (lorsque $\rho < 1$)

Variables du *data frame* `splineKnots` (argument `SB_splineKnots`) utilisées dans le graphique :

- `x` pour l'axe des x (t)
- `y` pour la ligne *Cubic spline* et les points *Extra knot* et *Original knot*
- `extraKnot` pour le type de nœud (*Extra knot* contre *Original knot*)

Voir la section **Valeur de retour** de `benchmarking()` et `stock_benchmarking()` pour plus d'informations sur ces *data frames*.

Value

Cette fonction ne renvoie rien (`invisible(NULL)`).

See Also

`plot_graphTable()` bench_graphs `benchmarking()` `stock_benchmarking()`

Examples

```
#######
# Étapes préliminaires

# Stocks trimestriels (même patron répété pour 7 années)
sc_tri <- ts(rep(c(85, 95, 125, 95), 7), start = c(2013, 1), frequency = 4)

# Stocks de fin d'année
sc_ann <- ts(c(135, 125, 155, 145, 165), start = 2013, frequency = 1)

# Étalonnage proportionnel
# ... avec `benchmarking()` (approche "Proc Benchmarking")
res_PB <- benchmarking(
  ts_to_tsDF(sc_tri),
  ts_to_bmkDF(sc_ann, discrete_flag = TRUE, alignment = "e", ind_frequency = 4),
  rho = 0.729, lambda = 1, biasOption = 3,
  quiet = TRUE)
```

```
# ... avec `stock_benchmarking()`
res_SB <- stock_benchmarking(
  ts_to_tsDF(sc_tri),
  ts_to_bmkDF(sc_ann, discrete_flag = TRUE, alignment = "e", ind_frequency = 4),
  rho = 0.729, lambda = 1, biasOption = 3,
  quiet = TRUE)


#######
# Tracer les ajustements d'étalonnage

# Ajustements de `benchmarking()` (`res_PB`), sans légende
plot_benchAdj(PB_graphTable = res_PB$graphTable,
              legendPos = NULL)

# Ajouter les de `stock_benchmarking()` (`res_SB`), avec une légende cette fois
plot_benchAdj(PB_graphTable = res_PB$graphTable,
              SB_graphTable = res_SB$graphTable)

# Ajouter la spline cubique de `stock_benchmarking()` utilisée pour générer les ajustements
# (incluant les nœuds supplémentaires aux deux extrémités), avec légende en haut à gauche
plot_benchAdj(PB_graphTable = res_PB$graphTable,
              SB_graphTable = res_SB$graphTable,
              SB_splineKnots = res_SB$splineKnots,
              legendPos = "topleft")


#######
# Simuler l'étalonnage de plusieurs séries (3 séries de stocks)

sc_tri2 <- ts.union(ser1 = sc_tri, ser2 = sc_tri * 100, ser3 = sc_tri * 10)
sc_ann2 <- ts.union(ser1 = sc_ann, ser2 = sc_ann * 100, ser3 = sc_ann * 10)

# Avec l'argument `allCols = TRUE` (stocks identifiés avec la colonne `varSeries`)
res_SB2 <- stock_benchmarking(
  ts_to_tsDF(sc_tri2),
  ts_to_bmkDF(sc_ann2, discrete_flag = TRUE, alignment = "e", ind_frequency = 4),
  rho = 0.729, lambda = 1, biasOption = 3,
  allCols = TRUE,
  quiet = TRUE)

# Ajustements d'étalonnage pour le 2ième stock (ser2)
plot_benchAdj(
  SB_graphTable = res_SB2$graphTable[res_SB2$graphTable$varSeries == "ser2", ])

# Avec l'argument `by = "series"` (stocks identifiés avec la colonne `series`)
res_SB3 <- stock_benchmarking(
  stack_tsDF(ts_to_tsDF(sc_tri2)),
  stack_bmkDF(ts_to_bmkDF(
    sc_ann2, discrete_flag = TRUE, alignment = "e", ind_frequency = 4)),
  rho = 0.729, lambda = 1, biasOption = 3,
  by = "series",
  quiet = TRUE)
```

```
# Spline cubique pour le 3ième stock (ser3)
plot_benchAdj(
  SB_splineKnots = res_SB3$splineKnots[res_SB3$splineKnots$series == "ser3", ])
```

`plot_graphTable` *Générer des graphiques d'étalonnage dans un fichier PDF*

Description

Créer un fichier PDF (format de papier lettre US en orientation paysage) contenant des graphiques d'étalonnage pour l'ensemble des séries contenues dans le *data frame* de sortie **graphTable** (argument `graphTable`) de la fonction d'étalonnage (`benchmarking()` ou `stock_benchmarking()`) spécifiée. Quatre types de graphiques d'étalonnage peuvent être générés pour chaque série :

- **Échelle originale** (argument `ori_plot_flag`) - graphique superposé des composantes :
  - Série indicatrice
  - Moyennes de la série indicatrice
  - Série indicatrice corrigée pour le biais (lorsque $\rho < 1$)
  - Série étalonnée
  - Moyennes des étalons
- **Échelle d'ajustement** (argument `adj_plot_flag`) - graphique superposé des composantes :
  - Ajustements d'étalonnage
  - Moyennes des ajustements d'étalonnage
  - Ligne du biais (lorsque $\rho < 1$)
- **Taux de croissance** (argument `GR_plot_flag`) - diagramme à barres des taux de croissance des séries indicatrice et étalonnée.
- **Tableau des taux de croissance** (argument `GR_table_flag`) - tableau des taux de croissance des séries indicatrice et étalonnée.

Ces graphiques peuvent être utiles pour évaluer la qualité des résultats de l'étalonnage. N'importe lequel des quatre types de graphiques d'étalonnage peut être activé ou désactivé à l'aide du drapeau (*flag*) correspondant. Les trois premiers types graphiques sont générés par défaut alors que le quatrième (le tableau des taux de croissance) ne l'est pas.

Usage

```
plot_graphTable(
  graphTable,
  pdf_file,
  ori_plot_flag = TRUE,
  adj_plot_flag = TRUE,
  GR_plot_flag = TRUE,
  GR_table_flag = FALSE,
  add_bookmarks = TRUE
)
```

Arguments

`graphTable` (obligatoire)

Data frame, ou objet compatible, correspondant au *data frame* de sortie **graphTable** de la fonction d'étalonnage.

`pdf_file` (obligatoire)

Nom (et chemin) du fichier PDF qui contiendra les graphiques d'étalonnage. Le nom doit inclure l'extension de fichier « .pdf ». Le fichier PDF sera créé dans le répertoire de travail de la session R (tel que renvoyé par `getwd()`) si aucun chemin n'est spécifié. La sécification de `NULL` annulerait la création d'un fichier PDF.

`ori_plot_flag`, `adj_plot_flag`, `GR_plot_flag`, `GR_table_flag`

(optionnels)

Arguments logiques (*logical*) indiquant si le type de graphique d'étalonnage correspondant doit être généré ou non. Les trois premiers types de graphiques sont générés par défaut alors que le quatrième (le tableau des taux de croissance) ne l'est pas.

Les valeurs par défaut sont `ori_plot_flag = TRUE`, `adj_plot_flag = TRUE`, `GR_plot_flag = TRUE` et `GR_table_flag = FALSE`.

`add_bookmarks` Argument logique (*logical*) indiquant si des signets doivent être ajoutés au fichier PDF. Voir **Signets** dans la section **Détails** pour plus d'informations.

La valeur par défaut est `add_bookmarks = TRUE`.

Details

Liste des variables du *data frame* **graphTable** (argument `graphTable`) correspondant à chaque élément des quatre types de graphiques d'étalonnage:

- Échelle originale (argument `ori_plot_flag`)
  - `subAnnual` pour la ligne *Indicator Series*
  - `avgSubAnnual` pour les segments *Avg. Indicator Series*
  - `subAnnualCorrected` pour la ligne *Bias Corr. Indicator Series* (lorsque $\rho < 1$)
  - `benchmarked` pour la ligne *Benchmarked Series*
  - `avgBenchmark` pour les segments *Average Benchmark*
- Échelle d'ajustement (argument `adj_plot_flag`)
  - `benchmarkedSubAnnualRatio` pour la ligne *BI Ratios (Benchmarked Series / Indicator Series)* $^{(*)}$
  - `avgBenchmarkSubAnnualRatio` pour les segments *Average BI Ratios* $^{(*)}$
  - `bias` pour la ligne *Bias* (lorsque $\rho < 1$)
- Taux de croissance (argument `GR_plot_flag`)
  - `growthRateSubAnnual` pour les barres *Growth R. in Indicator Series* $^{(*)}$
  - `growthRateBenchmarked` pour les barres *Growth R. in Benchmarked Series* $^{(*)}$
- Tableau des taux de croissance (argument `GR_table_flag`)
  - `year` pour la colonne *Year*
  - `period` pour la colonne *Period*

- `subAnnual` pour la colonne *Indicator Series*
- `benchmarked` pour la colonne *Benchmarked Series*
- `growthRateSubAnnual` pour la colonne *Growth Rate in Indicator Series* (*)
- `growthRateBenchmarked` pour la colonne *Growth Rate in Benchmarked Series* (*)

(*) Les *ratios étalons/indicateurs* (« BI ratios ») et les *taux de croissance* (« growth rates ») correspondent en réalité à des *différences* lorsque $\lambda = 0$ (étalonnage additif).

La fonction utilise les colonnes supplémentaires du *data frame* `graphTable` (colonnes non listées dans la section **Valeur de retour** de `benchmarking()` et `stock_benchmarking()`), le cas échéant, pour construire les groupes-BY. Voir la section **Étalonnage de plusieurs séries** de `benchmarking()` pour plus de détails.

Performance:

Les deux types de graphiques de taux de croissance, c'est-à-dire le diagramme à barres (`GR_plot_flag`) et le tableau (`GR_table_flag`), nécessitent souvent la génération de plusieurs pages dans le fichier PDF, en particulier pour les longues séries mensuelles avec plusieurs années de données. Cette création de pages supplémentaires ralentit l'exécution de `plot_graphTable()`. C'est pourquoi seul le diagramme à barres est généré par défaut (`GR_plot_flag = TRUE` et `GR_table_flag = FALSE`). La désactivation des deux types de graphiques de taux de croissance (`GR_plot_flag = FALSE` et `GR_table_flag = FALSE`) ou la réduction de la taille du *data frame* d'entrée `graphTable` pour les séries très longues (ex., en ne gardant que les années récentes) pourrait ainsi améliorer le temps d'exécution. Notez également que l'impact de l'étalonnage sur les taux de croissance peut être déduit du graphique dans l'échelle d'ajustement (`adj_plot_flag`) en examinant l'ampleur du mouvement vertical (vers le bas ou vers le haut) des ajustements d'étalonnage entre deux périodes adjacentes : plus le mouvement vertical est important, plus l'impact sur le taux de croissance correspondant est important. Le temps d'exécution de `plot_graphTable()` pourrait donc être reduit, si nécessaire, en ne générant que les deux premiers types de graphiques et en se concentrant sur le graphique des d'ajustements d'étalonnage pour évaluer la préservation du mouvement d'une période à l'autre, c'est-à-dire l'impact de l'étalonnage sur les taux de croissance initiaux.

Thèmes de ggplot2:

Les graphiques sont générés avec la librairie ggplot2 qui est livrée avec un ensemble pratique de thèmes complets pour l'aspect général des graphiques (avec `theme_grey()` comme thème par défaut). Utilisez la fonction `theme_set()` pour changer le thème appliqué aux graphiques générés par `plot_graphTable()` (voir les **Exemples**).

Signets:

Des signets sont ajoutés au fichier PDF avec `xmpdf::set_bookmarks()` lorsque l'argument `add_bookmarks = TRUE` (par défault), ce qui nécessite un outil tiers tel que Ghostscript ou PDFtk. Voir la section **Installation** dans `vignette("xmpdf", package = "xmpdf")` pour plus de détails.

Important : les signets seront ajoutés avec succès au fichier PDF **si et seulement si** `xmpdf::supports_set_bookmarks()` renvoie `TRUE` **et** l'exécution de `xmpdf::set_bookmarks()` est réussie. Si Ghostscript est installé sur votre machine mais que `xmpdf::supports_set_bookmarks()` renvoie toujours `FALSE`, essayez de spécifier le chemin de l'exécutable Ghostscript dans la variable d'environnement `R_GSCMD` (ex., `Sys.setenv(R_GSCMD = "C:/Program Files/.../bin/gswin64c.exe")` avec Windows). D'un autre côté, si `xmpdf::supports_set_bookmarks()}` renvoie `TRUE` mais que vous rencontrez des problèmes (insolubles) avec `xmpdf::set_bookmarks()` (ex., erreur liée à l'exécutable Ghostscript), la création de signets peut être désactivée en spécifiant `add_bookmarks = FALSE`.

Value

En plus de créer un fichier PDF contenant les graphiques d'étalonnage (sauf si `pdf_file = NULL`), cette fonction renvoie également de manière invisible une liste comprenant les éléments suivants :

- **pdf_name** : Chaîne de caractères (vecteur de type caractère de longueur un) qui contient le nom complet et le chemin du fichier PDF s'il a été créé avec succès et `invisible(NA_character_)` dans le cas contraire ou si `pdf_file = NULL` a été spécifié.
- **graph_list** : Liste des graphiques d'étalonnage générés (une par série) comprenant les éléments suivants :
  - **name** : Chaîne de caractères décrivant la série (concorde avec le nom du signet dans le fichier PDF).
  - **page** : Entier représentant le numéro de séquence du premier graphique de la série dans la séquence complète des graphiques pour toutes les séries (concorde avec le numéro de page dans le fichier PDF).
  - **ggplot_list** : Liste d'objets ggplot (une par graphique ou par page dans le fichier PDF) correspondant aux graphiques d'étalonnage générés pour la série. Voir la section **Valeur** dans bench_graphs pour plus de détails.

Notez que les objets ggplot renvoyés par la fonction peuvent être affichés *manuellement* avec `print()`, auquel cas certaines mises à jour des paramètres par défaut du thème ggplot2 sont recommandées afin de produire des graphiques ayant une apparence similaire à ceux générés dans le fichier PDF (voir la section **Valeur** dans bench_graphs pour les détails). Gardez également à l'esprit que ces graphiques sont optimisés pour un format de papier Lettre US en orientation paysage, c.-à-d., 11po de large (27.9cm, 1056px avec 96 PPP) et 8.5po de haut (21.6cm, 816px avec 96 PPP).

See Also

bench_graphs `plot_benchAdj()` `benchmarking()` `stock_benchmarking()`

Examples

```
# Définir le répertoire de travail (pour les fichiers graphiques PDF)
rep_ini <- getwd()
setwd(tempdir())


# Ventes trimestrielles de voitures et camionnettes (séries indicatrices)
ind_tri <- ts_to_tsDF(
  ts(matrix(c(# Voitures
              1851, 2436, 3115, 2205, 1987, 2635, 3435, 2361, 2183, 2822,
              3664, 2550, 2342, 3001, 3779, 2538, 2363, 3090, 3807, 2631,
              2601, 3063, 3961, 2774, 2476, 3083, 3864, 2773, 2489, 3082,
              # Camionnettes
              1900, 2200, 3000, 2000, 1900, 2500, 3800, 2500, 2100, 3100,
              3650, 2950, 3300, 4000, 3290, 2600, 2010, 3600, 3500, 2100,
              2050, 3500, 4290, 2800, 2770, 3080, 3100, 2800, 3100, 2860),
            ncol = 2),
    start = c(2011, 1),
    frequency = 4,
    names = c("voitures", "camionnettes")))
```

```
# Ventes annuelles de voitures et camionnettes (étalons)
eta_tri <- ts_to_bmkDF(
  ts(matrix(c(# Voitures
              10324, 10200, 10582, 11097, 11582, 11092,
              # Camionnettes
              12000, 10400, 11550, 11400, 14500, 16000),
            ncol = 2),
     start = 2011,
     frequency = 1,
     names = c("voitures", "camionnettes")),
  ind_frequency = 4)


# Étalonnage proportionnel sans correction pour le biais
res_eta <- benchmarking(ind_tri, eta_tri,
                        rho = 0.729, lambda = 1, biasOption = 1,
                        allCols = TRUE,
                        quiet = TRUE)


# Ensemble de graphiques par défaut (les 3 premiers types de graphiques)
plot_graphTable(res_eta$graphTable, "graphes_etalonnage.pdf")

# Utiliser temporairement `theme_bw()` de ggplot2 pour les graphiques
library(ggplot2)
theme_ini <- theme_get()
theme_set(theme_bw())
plot_graphTable(res_eta$graphTable, "graphes_etalonnage_bw.pdf")
theme_set(theme_ini)

# Generer les 4 types de graphiques (incluant le tableau des taux de croissance)
plot_graphTable(res_eta$graphTable, "graphes_etalonnage_avec_tableauTC.pdf",
                GR_table_flag = TRUE)

# Réduire le temps d'exécution en désactivant les deux types de graphiques
# des taux de croissance
plot_graphTable(res_eta$graphTable, "graphes_etalonnage_sans_TC.pdf",
                GR_plot_flag = FALSE)


# Réinitialiser le répertoire de travail à son emplacement initial
setwd(rep_ini)
```

`rkMeta_to_blSpecs` *Convertir des métadonnées de réconciliation*

Description

Convertir un *data frame* de métadonnées `tsraking()` en un *data frame* de spécifications de problème `tsbalancing()`.

Usage

```
rkMeta_to_blSpecs(
  metadata_df,
  alterability_df = NULL,
  alterSeries = 1,
  alterTotal1 = 0,
  alterTotal2 = 0,
  alterability_df_only = FALSE
)
```

Arguments

`metadata_df` (obligatoire)

Data frame, ou objet compatible, qui décrit les contraintes d'agrégation transversales (règles d'additivité) pour le problème de ratissage (« *raking* »). Deux variables de type caractère doivent être incluses dans le *data frame* : `series` et `total1`. Deux variables sont optionnelles : `total2` (caractère) et `alterAnnual` (numérique). Les valeurs de la variable `series` représentent les noms des variables des séries composantes dans le *data frame* des données d'entrée (argument `data_df`). De même, les valeurs des variables `total1` et `total2` représentent les noms des variables des totaux de contrôle transversaux de 1ère et 2ème dimension dans le *data frame* des données d'entrée. La variable `alterAnnual` contient le coefficient d'altérabilité pour la contrainte temporelle associée à chaque série composante. Lorsqu'elle est spécifiée, cette dernière remplace le coefficient d'altérabilité par défaut spécifié avec l'argument `alterAnnual`.

`alterability_df`

(optionnel)

Data frame, ou objet compatible, ou `NULL`, qui contient les variables de coefficients d'altérabilité. Elles doivent correspondre à une série composante ou à un total de contrôle transversal, c'est-à-dire qu'une variable portant le même nom doit exister dans le *data frame* des données d'entrée (argument `data_df`). Les valeurs de ces coefficients d'altérabilité remplaceront les coefficients d'altérabilité par défaut spécifiés avec les arguments `alterSeries`, `alterTotal1` et `alterTotal2`. Lorsque le *data frame* des données d'entrée contient plusieurs enregistrements et que le *data frame* des coefficients d'altérabilité n'en contient qu'un seul, les coefficients d'altérabilité sont utilisés (répétés) pour tous les enregistrements du *data frame* des données d'entrée. Le *data frame* des coefficients d'altérabilité peut également contenir autant d'enregistrements que le *data frame* des données d'entrée.

La valeur par défaut est `alterability_df = NULL` (coefficients d'altérabilité par défaut).

`alterSeries` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les valeurs des séries composantes. Il s'appliquera aux séries composantes pour lesquelles des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterSeries = 1.0` (valeurs des séries composantes non contraignantes).

`alterTotal1` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les totaux de contrôle transversaux de la 1ère dimension. Il s'appliquera aux totaux de contrôle transversaux pour lesquels des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterTotal1 = 0.0` (totaux de contrôle transversaux de 1ère dimension contraignants).

`alterTotal2` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les totaux de contrôle transversaux de la 2ème dimension. Il s'appliquera aux totaux de contrôle transversaux pour lesquels des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterTotal2 = 0.0` (totaux de contrôle transversaux de 2ème dimension contraignants).

`alterability_df_only`

(optionnel)

Argument logique (*logical*) spécifiant si oui ou non seul l'ensemble des coefficients d'altérabilité trouvés dans le fichier d'altérabilité (argument `alterability_df`) doit être inclus dans le *data frame* de spécifications de problème `tsbalancing()` renvoyé. Lorsque `alterability_df_only = FALSE` (la valeur par défaut), les coefficients d'altérabilité spécifiés avec les arguments `alterSeries`, `alterTotal1` et `alterTotal2` sont combinés avec ceux trouvés dans `alterability_df` (les derniers coefficients remplaçant les premiers) et le *data frame* renvoyé contient donc les coefficients d'altérabilité pour toutes les séries composantes et de totaux de contrôle transversaux. Cet argument n'affecte pas l'ensemble des coefficients d'altérabilité des totaux temporels (associés à l'argument `alterAnnual` de `tsraking()`) qui sont inclus dans le *data frame* de spécifications de problème `tsbalancing()` renvoyé. Ce dernier contient toujours strictement ceux spécifiés dans `metadata_df` avec une valeur non manquante (non `NA`) pour la colonne `alterAnnual`.

La valeur par défaut est `alterability_df_only = FALSE`.

Details

La description précédente de l'argument `alterability_df` provient de `tsraking()`. Cette fonction (`rkMeta_to_blSpecs()`) modifie légèrement la spécification des coefficients d'altérabilité avec l'argument `alterability_df` en permettant

- soit un seul enregistrement, spécifiant l'ensemble des coefficients d'altérabilité à utiliser pour toutes les périodes,
- soit un ou plusieurs enregistrements avec une colonne supplémentaire nommée `timeVal` permettant de spécifier à la fois des coefficients d'altérabilité spécifiques à la période (`timeVal` n'est pas `NA`) et des coefficients génériques à utiliser pour toutes les autres périodes (`timeVal`

est NA). Les valeurs de la colonne timeVal correspondent aux *valeurs de temps* d'un objet « ts » telles que renvoyées par stats::time(), correspondant conceptuellement à $année + (période - 1)/fréquence$.

Une autre différence avec tsraking() est que des valeurs manquantes (NA) sont autorisés dans le *data frame* des coefficients d'altérabilité (argument alterability_df) et que l'on utiliserait alors les coefficients génériques (enregistrements pour lesquels timeVal est NA) ou les coefficients par défaut (arguments alterSeries, alterTotal1 et alterTotal2).

Notez que à part rejeter les coefficients d'altérabilité pour les séries qui ne sont pas énumérées dans le *data frame* des métadonnées de ratissage (argument metadata_df), cette fonction ne valide pas les valeurs trouvées dans le *data frame* des coefficients d'altérabilité (argument alterability_df) ni celles trouvées dans la colonne alterAnnual du *data frame* des métadonnées de ratissage (argument metadata_df). La fonction les transfère *telles quelles* dans le *data frame* des spécifications de problème tsbalancing() renvoyé.

Value

Un *data frame* de spécifications de problème tsbalancing() (argument problem_specs_df).

See Also

tsraking() tsbalancing()

Examples

```
# Métadonnées de `tsraking()` pour un problème à deux dimensions (table 2 x 2)
mes_metadonnees <- data.frame(series = c("A1", "A2", "B1", "B2"),
                              total1 = c("totA", "totA", "totB", "totB"),
                              total2 = c("tot1", "tot2", "tot1", "tot2"))
mes_metadonnees


# Convertir en spécifications de `tsbalancing()`

# Inclure les coefficients d'altérabilité par défaut de `tsraking()`
rkMeta_to_blSpecs(mes_metadonnees)

# Totaux presque contraignants pour la 1ère marge (petits coef. d'altérabilité pour
# les colonnes `totA` et `totB`)
tail(rkMeta_to_blSpecs(mes_metadonnees, alterTotal1 = 1e-6))

# Ne pas inclure les coef. d'altérabilité (contraintes d'agrégation uniquement)
rkMeta_to_blSpecs(mes_metadonnees, alterability_df_only = TRUE)

# Avec un fichier de coefficients d'altérabilité (argument `alterability_df`)
mes_coefsAlt = data.frame(B2 = 0.5)
tail(rkMeta_to_blSpecs(mes_metadonnees, alterability_df = mes_coefsAlt))

# N'inclure que les coefficients d'altérabilité du fichier `alterability_df`
# (c.-à-d. pour la colonne `B2`)
tail(rkMeta_to_blSpecs(mes_metadonnees, alterability_df = mes_coefsAlt,
                       alterability_df_only = TRUE))
```

`stack_bmkDF` *Empiler des « données étalon »*

Description

Convertir un *data frame* d'étalons multivariés (voir `ts_to_bmkDF()`) pour les fonctions d'étalonnage (`benchmarking()` et `stock_benchmarking()`) en un *data frame* empilé (long) avec six variables (colonnes) :

- une (1) pour le nom de l'étalon (ex., nom de série)
- quatre (4) pour la converture de l'étalon
- une (1) pour la valeur de l'étalon

Les valeurs d'étalon manquantes (`NA`) ne sont pas incluses par défaut dans le *data frame* empilé renvoyé par la fonction. Spécifiez l'argument `keep_NA = TRUE` pour les conserver.

Cette fonction est utile lorsque l'on souhaite utiliser l'argument `by` (mode de traitement *groupes-BY*) des fonctions d'étalonnage afin d'étalonner plusieurs séries en un seul appel de fonction.

Usage

```
stack_bmkDF(
  bmk_df,
  ser_cName = "series",
  startYr_cName = "startYear",
  startPer_cName = "startPeriod",
  endYr_cName = "endYear",
  endPer_cName = "endPeriod",
  val_cName = "value",
  keep_NA = FALSE
)
```

Arguments

`bmk_df` (obligatoire)

Data frame, ou objet compatible, qui contient les étalons multivariés à empiler.

`ser_cName` (optionnel)

Chaîne de caractères spécifiant le nom de la variable (colonne) du *data frame* empilé de sortie qui contiendra les nom des étalons (nom des variables d'étalons dans le *data frame* d'étalons multivariés d'entrée). Cette variable peut ensuite être utilisée comme variable de groupes-BY (argument `by`) avec les fonctions d'étalonnage.

La valeur par défaut est `ser_cName = "series"`.

`startYr_cName`, `startPer_cName`, `endYr_cName`, `endPer_cName`

(optionnel)

Chaînes de caractères spécifiant le nom des variables (colonnes) numériques du *data frame* d'étalons multivariés d'entrée qui définissent la couverture des

étalons, c'est-à-dire les identificateurs de l'année et de la période (cycle) de début et de fin des étalons. Ces variables sont *transférées* dans le *data frame* empilé de sortie avec les mêmes noms de variable.

Les valeurs par défaut sont `startYr_cName = "startYear"`, `startPer_cName = "startPeriod"` `endYr_cName = "endYear"` et `endPer_Name = "endPeriod"`.

`val_cName` (optionnel)

Chaîne de caractères spécifiant le nom de la variable (colonne) du *data frame* empilé de sortie qui contiendra les valeurs des étalons.

La valeur par défaut est `val_cName = "value"`.

`keep_NA` (optionnel)

Argument logique (*logical*) spécifiant si les valeurs d'étalon manquantes (`NA`) du *data frame* d'étalons multivariés d'entrée doivent être conservées dans le *data frame* empilé de sortie.

La valeur par défaut est `keep_NA = FALSE`.

Value

La fonction renvoie un *data frame* avec six variables :

- Nom de l'étalon (de la série), type caractère (voir l'argument `ser_cName`)
- Année de début de la couverture de l'étalon, type numérique (voir argument `startYr_cName`)
- Période de début de la couverture de l'étalon, type numérique (voir argument `startPer_cName`)
- Année de fin de la couverture de l'étalon, type numérique (voir argument `endtYr_cName`)
- Période de fin de la couverture de l'étalon, type numérique (voir argument `endPer_cName`)
- Valeur de l'étalon, type numérique (voir argument `val_cName`)

Note : la fonction renvoie un objet « data.frame » qui peut être explicitement converti en un autre type d'objet avec la fonction `as*()` appropriée (ex., `tibble::as_tibble()` le convertirait en tibble).

See Also

`stack_tsDF()` `ts_to_bmkDF()` `benchmarking()` `stock_benchmarking()`

Examples

```
# Créer un « data frame » d'étalons annuels pour 2 séries indicatrices trimestrielles
# (avec des valeurs manquantes pour les étalons des 2 dernières années)
mes_etalons <- ts_to_bmkDF(ts(data.frame(ser1 = c(1:3 *  10, NA, NA),
                                         ser2 = c(1:3 * 100, NA, NA)),
                              start = c(2019, 1), frequency = 1),
                           ind_frequency = 4)
mes_etalons


# Empiler les étalons ...

# en rejetant les `NA` dans les données empilées (comportement par défaut)
```

```
stack_bmkDF(mes_etalons)

# en conservant les `NA` dans les données empilées
stack_bmkDF(mes_etalons, keep_NA = TRUE)

# en utilisant des noms de variables (colonnes) personnalisés
stack_bmkDF(mes_etalons, ser_cName = "nom_eta", val_cName = "val_eta")
```

`stack_tsDF` *Empiler des données de séries chronologiques*

Description

Convertir un *data frame* de séries chronologiques multivariées (voir `ts_to_tsDF()`) pour les fonctions d'étalonnage (`benchmarking()` et `stock_benchmarking()`) en un *data frame* empilé (long) avec quatre variables (colonnes) :

- une (1) pour le nom de la série
- deux (2) pour l'identification du point de données (année et période)
- une (1) pour la valeur du point de données

Les valeurs de série manquantes (`NA`) ne sont pas incluses par défaut dans le *data frame* empilé renvoyé par la fonction. Spécifiez l'argument `keep_NA = TRUE` pour les conserver.

Cette fonction est utile lorsque l'on souhaite utiliser l'argument `by` (mode de traitement *groupes-BY*) des fonctions d'étalonnage afin d'étalonner plusieurs séries en un seul appel de fonction.

Usage

```
stack_tsDF(
  ts_df,
  ser_cName = "series",
  yr_cName = "year",
  per_cName = "period",
  val_cName = "value",
  keep_NA = FALSE
)
```

Arguments

`ts_df`	(obligatoire) *Data frame*, ou objet compatible, qui contient les données de séries chronologiques multivariées à empiler.
`ser_cName`	(optionnel) Chaîne de caractères spécifiant le nom de la variable (colonne) du *data frame* empilé de sortie qui contiendra les nom des séries (nom des variables des séries

dans le *data frame* de séries chronologiques multivariées d'entrée). Cette variable peut ensuite être utilisée comme variable de groupes-BY (argument by) avec les fonctions d'étalonnage.

La valeur par défaut est `ser_cName = "series"`.

`yr_cName, per_cName`

(optionnel)

Chaînes de caractères spécifiant le nom des variables (colonnes) numériques du *data frame* de séries chronologiques multivariées d'entrée qui identifient l'année et la période (cycle) des points de données. Ces variables sont *transférées* dans le *data frame* empilé de sortie avec les mêmes noms de variable.

Les valeurs par défaut sont `yr_cName = "year"` et `per_cName = "period"`.

`val_cName` (optionnel)

Chaîne de caractères spécifiant le nom de la variable (colonne) du *data frame* empilé de sortie qui contiendra la valeur des points de données.

La valeur par défaut est `val_cName = "value"`.

`keep_NA` (optionnel)

Argument logique (*logical*) spécifiant si les valeurs de série manquantes (`NA`) du *data frame* de séries chronologiques multivariées d'entrée doivent être conservées dans le *data frame* empilé de sortie.

La valeur par défaut est `keep_NA = FALSE`.

Value

La fonction renvoie un *data frame* avec quatre variables :

- Nom de la série, type caractère (voir l'argument `ser_cName`)
- Année du point de données, type numérique (voir argument `yr_cName`)
- Période du point de données, type numérique (voir argument `per_cName`)
- Valeur du point de données, type numérique (voir argument `val_cName`)

Note : la fonction renvoie un objet « data.frame » qui peut être explicitement converti en un autre type d'objet avec la fonction `as*()` appropriée (ex., `tibble::as_tibble()` le convertirait en tibble).

See Also

`unstack_tsDF()` `stack_bmkDF()` `ts_to_tsDF()` `benchmarking()` `stock_benchmarking()`

Examples

```
# Créer un « data frame » de 2 séries indicatrices trimestrielles
# (avec des valeurs manquantes pour les 2 dernières trimestres)
mes_indicateurs <- ts_to_tsDF(ts(data.frame(ser1 = c(1:5 *  10, NA, NA),
                                            ser2 = c(1:5 * 100, NA, NA)),
                                 start = c(2019, 1), frequency = 4))
mes_indicateurs
```

```
# Empiler les séries indicatrices ...

# en rejetant les `NA` dans les données empilées (comportement par défaut)
stack_tsDF(mes_indicateurs)

# en conserver les `NA` dans les données empilées
stack_tsDF(mes_indicateurs, keep_NA = TRUE)

# en utilisant des noms de variables (colonnes) personnalisés
stack_tsDF(mes_indicateurs, ser_cName = "nom_ind", val_cName = "val_ind")
```

`stock_benchmarking` *Rétablir les contraintes temporelles pour des séries de stocks*

Description

Fonction spécifiquement destinée à l'étalonnage des séries de stocks où les étalons sont des points d'ancrage couvrant une seule période de la série indicatrice. Les étalons couvrant plus d'une période de la série indicatrice ne peuvent pas être utilisés avec cette fonction. La fonction `benchmarking()` doit être utilisée à la place pour étalonner des séries de flux (« non-stock »).

Plusieurs séries de stocks peuvent être étalonnées en un seul appel de fonction.

Notez que les fonctions `stock_benchmarking()` et `benchmarking()` partagent principalement les mêmes arguments et renvoient le même type d'objet. Les différences sont énumérées ci-dessous :

- L'argument `verbose` n'est pas défini pour `stock_benchmarking()`.
- Des arguments supplémentaires sont définis pour `stock_benchmarking()` :
  - `low_freq_periodicity`
  - `n_low_freq_proj`
  - `proj_knots_rho_bd`
- La liste renvoyée par `stock_benchmarking()` contient un *data frame* supplémentaire :
  - `splineKnots`

Voir la section **Détails** pour plus d'informations sur les similitudes et les différences entre les fonctions `stock_benchmarking()` et `benchmarking()`.

Un équivalent direct de `stock_benchmarking()` *n'existe pas dans G-Séries 2.0 en SAS®.*

Usage

```
stock_benchmarking(
  series_df,
  benchmarks_df,
  rho,
  lambda,
  biasOption,
  bias = NA,
  low_freq_periodicity = NA,
```

```
  n_low_freq_proj = 1,
  proj_knots_rho_bd = 0.995,
  tolV = 0.001,
  tolP = NA,
  warnNegResult = TRUE,
  tolN = -0.001,
  var = "value",
  with = NULL,
  by = NULL,
  constant = 0,
  negInput_option = 0,
  allCols = FALSE,
  quiet = FALSE
)
```

Arguments

`series_df`	(obligatoire) *Data frame*, ou objet compatible, qui contient les données de la (des) série(s) indicatrice(s) à étalonner. En plus de la (des) variable(s) contenant les données, spécifiée(s) avec l'argument `var`, le *data frame* doit aussi contenir deux variables numériques, `year` et `period`, identifiant les périodes des séries indicatrices.
`benchmarks_df`	(obligatoire) *Data frame*, ou objet compatible, qui contient les étalons. En plus de la (des) variable(s) contenant les données, spécifiée(s) avec l'argument `with`, le *data frame* doit aussi contenir quatre variables numériques, `startYear`, `startPeriod`, `endYear` et `endPeriod`, identifiant les périodes des séries indicatrices couvertes par chaque étalon.
`rho`	(obligatoire) Nombre réel compris dans l'intervalle $[0, 1]$ qui spécifie la valeur du paramètre autorégressif ρ. Voir la section **Détails** pour plus d'informations sur l'effet du paramètre ρ.
`lambda`	(obligatoire) Nombre réel, avec des valeurs suggérées dans l'intervalle $[-3, 3]$, qui spécifie la valeur du paramètre du modèle d'ajustement λ. Les valeurs typiques sont `lambda = 0.0` pour un modèle additif et `lambda = 1.0` pour un modèle proportionnel.
`biasOption`	(obligatoire) Spécification de l'option d'estimation du biais : • `1` : Ne pas estimer le biais. Le biais utilisé pour corriger la série indicatrice sera la valeur spécifiée avec l'argument `bias`. • `2` : Estimer le biais, afficher le résultat, mais ne pas l'utiliser. Le biais utilisé pour corriger la série indicatrice sera la valeur spécifiée avec l'argument `bias`. • `3` : Estimer le biais, afficher le résultat et utiliser le biais estimé pour corriger la série indicatrice. Toute valeur spécifiée avec l'argument `bias` sera ignorée.

L'argument biasOption n'est pas utilisé quand rho = 1.0. Voir la section **Détails** pour plus d'informations sur le biais.

bias (optionnel)

Nombre réel, ou NA, spécifiant la valeur du biais défini par l'utilisateur à utiliser pour la correction de la série indicatrice avant de procéder à l'étalonnage. Le biais est ajouté à la série indicatrice avec un modèle additif (argument lambda = 0.0) alors qu'il est multiplié dans le cas contraire (argument lambda != 0.0). Aucune correction de biais n'est appliquée lorsque bias = NA, ce qui équivaut à spécifier bias = 0.0 lorsque lambda = 0.0 et bias = 1.0 dans le cas contraire. L'argument bias n'est pas utilisé lorsque biasOption = 3 ou rho = 1.0. Voir la section **Détails** pour plus d'informations sur le biais.

La valeur par défaut est bias = NA (pas de biais défini par l'utilisateur).

low_freq_periodicity

(optionnel)

Nombre entier positif représentant le nombre de périodes définissant la *basse fréquence* (e.g., celle des étalons) pour l'ajout de nœuds supplémentaires à la spline cubique (avant le premier étalon et après le dernier étalon). Par exemple, low_freq _periodicity = 3 avec des indicateurs mensuels définira des nœuds trimestriels. Des nœuds annuels sont ajoutés lorsque low_freq_periodicity = NA.

La valeur par défaut est low_freq_periodicity = NA (nœuds annuels).

n_low_freq_proj

(optionnel)

Entier non négatif représentant le nombre de nœuds de basse fréquence (tel que défini avec l'argument low_freq_periodicity) à ajouter aux deux extrémités (avant le premier étalon et après le dernier étalon) avant de commencer à ajouter des nœuds de *haute fréquence* (celle de la série indicatrice).

La valeur par défaut est n_low_freq_proj = 1.

proj_knots_rho_bd

(optionnel)

Limite qui s'applique à la valeur spécifiée avec l'argument rho et qui determine le type noœuds supplémentaires à ajouter aux deux extrémités (avant le premier étalon et après le dernier étalon). Lorsque rho > proj_knots_rho_bd, des nœuds de *haute fréquence* (celle de la série indicatrice) sont utilisés immédiatement aux deux extrémité. Autrement, lorsque rho <= proj_knots_rho_bd, des nœuds de *basse fréquence* (voir les arguments low_freq_periodicity et n_low_freq_proj) sont d'abord projetés de part et d'autre. Notez que pour des stocks trimestriels, le cube de proj_knots_rho_bd est utilisé. Par conséquent, la valeur de l'argument proj_knots_rho_bd doit correspondre à des indicateurs de stocks mensuels; elle est ajustée à l'interne pour des stocks trimestriels. Cet argument vise à atteindre un compromis pour les périodes à l'extérieur (avant ou après) les étalons (points d'ancrage) fournis en entrée, c'est-à-dire des ajustements de type Denton (en ligne droite) lorsque rho s'approche de 1 (lorsque rho > proj_knots_rho_bd) et une spline cubique d'apparence normale (sans contorsions excessives) dans le cas contraire (lorsque rho <= proj_knots_rho_bd). La section **Détails** contient plus d'informations sur ce sujet et certains cas illustratifs sont fournis dans la section **Exemples**.

La valeur par défaut est `proj_knots_rho_bd = 0.995` (0.995^3 pour des indicateurs de stocks trimestriels).

`tolV`, `tolP` (optionnel)
Nombre réel non négatif, ou `NA`, spécifiant la tolérance, en valeur absolue ou en pourcentage, à utiliser pour la validation des étalons contraignants (coefficient d'altérabilité de 0.0) en sortie. Cette validation consiste à comparer la valeur des étalons contraignants en entrée à la valeur équivalente calculée à partir des données de la série étalonnée (sortie). Les arguments `tolV` et `tolP` ne peuvent pas être spécifiés tous les deux à la fois (l'un doit être spécifié tandis que l'autre doit être `NA`).
Exemple : pour une tolérance de 10 *unités*, spécifiez `tolV = 10, tolP = NA`; pour une tolérance de 1%, spécifiez `tolV = NA, tolP = 0.01`.
Les valeurs par défaut sont `tolV = 0.001` et `tolP = NA`.

`warnNegResult` (optionnel)
Argument logique (*logical*) spécifiant si un message d'avertissement doit être affiché lorsqu'une valeur négative créée par la fonction dans la série étalonnée (en sortie) est inférieure au seuil spécifié avec l'argument `tolN`.
La valeur par défaut est `warnNegResult = TRUE`.

`tolN` (optionnel)
Nombre réel négatif spécifiant le seuil pour l'identification des valeurs négatives. Une valeur est considérée négative lorsqu'elle est inférieure à ce seuil.
La valeur par défaut est `tolN = -0.001`.

`var` (optionnel)
Vecteur (longueur minimale de 1) de chaînes de caractères spécifiant le(s) nom(s) de variable(s) du *data frame* des séries indicatrices (argument `series_df`) contenant les valeurs et (optionnellement) les coefficients d'altérabilité définis par l'utilisateur de la (des) série(s) à étalonner. Ces variables doivent être numériques. La syntaxe est `var = c("serie1 </ alt_ser1>", "serie2 </ alt_ser2>", ...)`. Des coefficients d'altérabilité par défaut de 1.0 sont utilisés lorsqu'une variable de coefficients d'altérabilité définie par l'utilisateur n'est pas spécifiée à côté d'une variable de série indicatrice. Voir la section **Détails** pour plus d'informations sur les coefficients d'altérabilité.
Exemple : `var = "value / alter"` étalonnerait la variable `value` du *data frame* des séries indicatrices avec les coefficients d'altérabilité contenus dans la variable `alter` tandis que `var = c("value / alter", "value2")` étalonnerait en plus la variable `value2` avec des coefficients d'altérabilité par défaut de 1.0.
La valeur par défaut est `var = "value"` (étalonner la variable `value` avec des coefficients d'altérabilité par défaut de 1.0).

`with` (optionnel)
Vecteur (même longueur que l'argument `var`) de chaînes de caractères, ou `NULL`, spécifiant le(s) nom(s) de variable(s) du *data frame* des étalons (argument `benchmarks_df`) contenant les valeurs et (optionnellement) les coefficients d'altérabilité définis par l'utilisateur des étalons. Ces variables doivent être numériques. La spécification de `with = NULL` entraîne l'utilisation de variable(s) d'étalons correspondant à la (aux) variable(s) spécifiée(s) avec l'argument `var` sans coefficients d'altérabilité d'étalons définis par l'utilisateur (c'est à dire des coefficients d'altérabilité par défaut de 0.0 correspondant à des étalons contraignants).

La syntaxe est `with = NULL` ou `with = c("bmk1 </ alt_bmk1>", "bmk2 </ alt_bmk2>", ...)`. Des coefficients d'altérabilité par défaut de 0.0 (étalons contraignants) sont utilisés lorsqu'une variable de coefficients d'altérabilité définie par l'utilisateur n'est pas spécifiée à côté d'une variable d'étalon. Voir la section **Détails** pour plus d'informations sur les coefficients d'altérabilité.

Exemple : `with = "val_bmk"` utiliserait la variable `val_bmk` du *data frame* des étalons avec les coefficients d'altérabilité par défaut de 0.0 pour étalonner la série indicatrice tandis que `with = c("val_bmk", "val_bmk2 / alt_bmk2")` étalonnerait en plus une deuxième série indicatrice en utilisant la variable d'étalons `val_bmk2` avec les coefficients d'altérabilité d'étalons contenus dans la variable `alt_bmk2`.

La valeur par défaut est `with = NULL` (même(s) variable(s) d'étalons que l'argument `var` avec des coefficients d'altérabilité d'étalons par défaut de 0.0).

`by` (optionnel)

Vecteur (longueur minimale de 1) de chaînes de caractères, ou `NULL`, spécifiant le(s) nom(s) de variable(s) dans les *data frames* d'entrée (arguments `series_df` et `benchmarks_df`) à utiliser pour former des groupes (pour le traitement « groupes-BY ») et permettre l'étalonnage de plusieurs séries en un seul appel de fonction. Les variables groupes-BY peuvent être numériques ou caractères (facteurs ou non), doivent être présentes dans les deux *data frames* d'entrée et apparaîtront dans les trois *data frames* de sortie (voir la section **Valeur de retour**). Le traitement groupes-BY n'est pas implémenté lorsque `by = NULL`. Voir « Étalonnage de plusieurs séries » dans la section **Détails** pour plus d'informations.

La valeur par défaut est `by = NULL` (pas de traitement groupes-BY).

`constant` (optionnel)

Nombre réel qui spécifie une valeur à ajouter temporairement à la fois à la (aux) série(s) indicatrice(s) et aux étalons avant de résoudre les problèmes d'étalonnage proportionnels (`lambda != 0.0`). La constante temporaire est enlevée de la série étalonnée finale en sortie. Par exemple, la spécification d'une (petite) constante permettrait l'étalonnage proportionnel avec `rho = 1` (étalonnage de Denton proportionnel) sur avec des séries indicatrices qui comprennent des valeurs de 0. Sinon, l'étalonnage proportionnel avec des valeurs de 0 pour la série indicatrice n'est possible que lorsque `rho < 1`. Spécifier une constante avec l'étalonnage additif (`lambda = 0.0`) n'a pas d'impact sur les données étalonnées résultantes. Les variables de données dans le *data frame* de sortie **graphTable** incluent la constante, correspondant au problème d'étalonnage effectivement résolu par la fonction.

La valeur par défaut est `constant = 0` (pas de constante additive temporaire).

`negInput_option`

(optionnel)

Traitement des valeurs négatives dans les données d'entrée pour l'étalonnage proportionnel (`lambda != 0.0`) :

- `0` : Ne pas autoriser les valeurs négatives pour l'étalonnage proportionnel. Un message d'erreur est affiché en présence de valeurs négatives dans les séries indicatrices ou les étalons d'entrée et des valeurs manquantes (`NA`) sont renvoyées pour les séries étalonnées. Ceci correspond au comportement de G-Séries 2.0.

- `1` : Autoriser les valeurs négatives pour l'étalonnage proportionnel mais avec l'affichage d'un message d'avertissement.
- `2` : Autoriser les valeurs négatives pour l'étalonnage proportionnel sans afficher de message.

La valeur par défaut est `negInput_option = 0` (ne pas autoriser les valeurs négatives pour l'étalonnage proportionnel).

`allCols` (optionnel)

Argument logique (*logical*) spécifiant si toutes les variables du *data frame* des séries indicatrices (argument `series_df`), autres que `year` et `period`, déterminent l'ensemble des séries à étalonner. Les valeurs spécifiées avec les arguments `var` et `with` sont ignorées lorsque `allCols = TRUE`, ce qui implique automatiquement des coefficients d'altérabilité par défaut, et des variables avec les mêmes noms que les séries indicatrices doivent exister dans le *data frame* des étalons (argument `benchmarks_df`).

La valeur par défaut est `allCols = FALSE`.

`quiet` (optionnel)

Argument logique (*logical*) spécifiant s'il faut ou non afficher uniquement les informations essentielles telles que les messages d'avertissements, les messages d'erreurs et les informations sur les variables (séries) ou les groupes-BY lorsque plusieurs séries sont étalonnées en un seul appel à la fonction. Nous vous déconseillons d'*envelopper* votre appel à `benchmarking()` avec `suppressMessages()` afin de supprimer l'affichage des informations sur les variables (séries) ou les groupes-BY lors du traitement de plusieurs séries, car cela compliquerait le dépannage en cas de problèmes avec des séries individuelles. Notez que la spécification de `quiet = TRUE` annulera également l'argument `verbose`.

La valeur par défaut est `quiet = FALSE`.

Details

Comparaison avec `benchmarking()`:

Avec des séries de stocks, `benchmarking()` est connu pour produire des bris dans les ajustements d'étalonnage aux périodes correspondant aux étalons (points d'ancrage). `stock_benchmarking()` résout ce problème en travaillant directement sur les ajustements d'étalonnage. Des ajustements lisses pour les stocks sont garantis en estimant une spline cubique de *pente=0* (une spline qui est *plate* aux deux extrémités) passant par les nœuds correspondant à la différence (lorsque l'argument `lambda = 0.0`) ou au ratio (sinon) entre les étalons (points d'ancrage) et les valeurs correspondantes de la série indicatrice. Ces nœuds sont parfois appelés *différences BI* ou *ratios BI* (**B***enchmark-to-***I***ndicator* en anglais). Les interpolations à partir de la spline cubique estimée fournissent alors les ajustements d'étalonnage pour les périodes entre les étalons.

Les arguments `rho`, `lambda`, `biasOption` et `bias` jouent un rôle similaire à ceux de `benchmarking()`. Cependant, notez que pour `stock_benchmarking()`, l'argument `rho` n'affecte les résultats que pour les périodes à l'extérieur, ou autour, du premier et du dernier étalon et `lambda` ne prend que deux valeurs en pratique : `lambda = 0.0` pour des ajustements additifs (interpolations par spline cubique où les nœuds sont des *différences BI*) ou `lambda = 1.0` pour des ajustements multiplicatifs (interpolations par spline cubique où les nœuds sont des *ratios BI*). Toute valeur non nulle pour `lambda` donnerait le même résultat que `lambda = 1.0`. Les coefficients d'altérabilité

jouent également un rôle similaire à ceux de benchmarking() et ont les mêmes valeurs par défaut, c'est-à-dire 1.0 pour la série indicatrice (valeurs non contraignantes) et 0.0 pour les étalons (étalons contraignants). Cependant, comme pour l'argument lambda, les coefficients d'altérabilité de cette fonction ne prennent que deux valeurs en pratique : 0.0 pour des valeurs contraignantes ou 1.0 pour des valeurs non contraignantes. Tout coefficient d'altérabilité non nul renverrait le même résultat qu'un coefficient de 1.0. Une autre différence avec benchmarking() est que les coefficients d'altérabilité définis par l'utilisateur sont autorisés même si rho = 1 avec stock_benchmarking(). Enfin, le fait de spécifier un étalon non contraignant avec stock_benchmarking() équivaut à l'ignorer complètement, comme si l'étalon en question n'était pas inclus dans le fichier d'entrée des étalons. Par rapport à benchmarking(), cette approche se traduit généralement par un impact plus important des étalons non contraignants sur les résultats de l'étalonnage (sur les stocks étalonnés résultants).

Solution autour des premier et dernier étalons (*problème d'actualité* de l'étalonnage):

Une spline de *pente=0* est choisie parce qu'elle correspond conceptuellement à l'approche (populaire) d'*étalonnage de Denton* (rho = 1). Afin de fournir une solution avant le premier étalon et après le dernier étalon qui soit semblable à celle de benchmarking() lorsque rho < 1, c'est-à-dire des ajustements convergeant vers le biais à une vitesse dictée par l'argument rho, des nœuds supplémentaires sont ajoutés aux deux extrémités avant d'estimer la spline. Par défaut, un nœud supplémentaire de basse fréquence (défini par l'argument low_freq_periodicity) est ajouté de chaque côté (au début et à la fin), c'est-à-dire qu'un nœud supplémentaire est ajouté avant le premier étalon et après le dernier étalon. Ensuite, des nœuds de haute fréquence (celle de la série indicatrice) sont ajoutés pour couvrir l'étendue de la série indicatrice, à laquelle est ajoutée une année supplémentaire de nœuds de haute fréquence. La valeur de tous ces nœuds supplémentaires est basée sur les arguments rho, biasOption et bias. Cela produit des ajustements lisses et naturels pour les périodes à l'extérieur, ou autour, des premier et dernier étalons qui convergent progressivement vers le biais, de manière similaire à benchmarking(). Le nombre de nœuds supplémentaires de basse fréquence à ajouter peut être modifié avec l'argument n_low_freq_proj. L'utilisation immédiate de nœuds de haute fréquence (n_low_freq_proj = 0) produirait les mêmes ajustements projetés que benchmarking(). Cependant, notez que cela tend à produire une spline d'apparence peu naturelle (exagérément contortionnée) autour des premier et dernier étalons qui pourrait être révisée de manière substantielle une fois que le prochain étalon sera disponible. L'utilisation de la valeur par défaut n_low_freq_proj = 1 fonctionne généralement mieux. Cependant, lorsque rho est *proche de 1* (voir l'argument proj_knots_rho_bd), des noeuds de haute fréquence sont immédiatement ajoutés de chaque côté afin d'assurer des ajustements projetés de type Denton (en ligne droite) pour les périodes à l'extérieur des premier et dernier étalons. Enfin, une spline cubique de *pente=0* passant à travers les nœuds (originaux et supplémentaires) est estimée. Notez qu'en pratique, la spline de *pente=0* est en fait approximée en reproduisant la valeur des nœuds aux extrémités 100 fois au cours de la période suivante (à une fréquence correspondant à 100 fois la fréquence de la série indicatrice).

Une *spline naturelle* aux nœuds d'extrémité originaux (premier et dernier étalons) peut être approximée en spécifiant une grande valeur pour l'argument low_freq_periodicity. Plus la valeur de low_freq_periodicity est grande, plus la spline cubique se comportera comme une *spline naturelle* (dérivée seconde égale à 0 aux extrémités, c'est-à-dire une spline qui garde une pente constante aux extrémités au lieu d'être plate comme une spline de *pente=0*).

En résumé, les ajustements projetés sont contrôlés avec les arguments rho, bias (et biasOption), n_low_freq_proj, proj_knots_rho_bd et low_freq_periodicity :

- Les valeurs par défaut de ces arguments produisent des ajustements projetés du type fonction

benchmarking (convergence raisonnablement lente vers le biais).

- Des valeurs plus petites de rho généreraient une convergence plus rapide vers le biais.
- Spécifier un biais défini par l'utilisateur avec l'argument bias lorsque rho < 1 est une autre façon d'influencer la forme des ajustements projetés.
- Spécifier rho = 1 produit des ajustements projetés de type Denton (premiers/derniers ajustements répétés sans convergence vers le biais).
- Spécifier une grande valeur pour low_freq_periodicity génère des ajustements projetés qui se comportent plus comme une spline naturelle, c'est-à-dire des ajustements qui continuent dans la même direction au premier/dernier étalon. Plus la valeur de low_freq_periodicity est grande, plus les ajustements projetés continuent à aller dans la même direction avant de *tourner*.

La spline cubique associée aux ajustements de stock_benchmarking() peut être commodément tracée avec plot_benchAdj().

Note sur les révisions des ajustements d'étalonnage:

Les ajustements de benchmarking() ne seraient pas révisés si tous les futurs étalons tombaient exactement sur ceux qui sont projetés (sur la base du biais et de la valeur de rho) et si le biais était fixé. La même chose pourrait être obtenue avec stock_benchmarking() si *suffisamment* de nœuds de basse fréquence (celle des étalons) étaient projetés. Le problème avec cette approche, cependant, est que les ajustements projetés peuvent ne pas sembler naturels car la spline peut osciller plus que souhaité autour des nœuds projetés. Ceci est clairement perceptible lorsque rho s'approche de 1 et que la spline oscille autour des nœuds projetés alignés horizontalement au lieu d'être alignée sur une ligne parfaitement droite. L'implémentation par défaut de la spline autour des premier et dernier étalons décrite précédemment vise à atteindre une *solution de meilleur compromis* :

- une spline d'apparence naturelle aux extrémités évitant les oscillations et les contorsions excessives;
- de petites révisions de la spline si l'étalon suivant est proche de celui projeté lorsque rho est *assez éloigné* de 1 (rho <= proj_knots_rho_bd);
- ajustements projetés qui sont en ligne droite (sans oscillations) lorsque rho s'approche de 1 (rho > proj_knots_rho_bd).

Les sous-sections *Étalonnage de plusieurs séries*, *Arguments* constant *et* negInput_option et *Traitement des valeurs manquantes (*NA*)* à la fin de la section **Détails** de benchmarking() sont également pertinentes pour stock_benchmarking(). Consultez-les au besoin.

Enfin, notez que la spline cubique associée aux ajustements de stock_benchmarking() peut être commodément tracée avec plot_benchAdj(). Cette dernière est utilisée dans les **Exemples** pour illustrer certains des sujets abordés ci-dessus.

Value

La fonction renvoie une liste de quatre *data frames* :

- **series** : *data frame* contenant les données étalonnées (sortie principale de la fonction). Les variables BY spécifiées avec l'argument by sont incluses dans le *data frame* mais pas les variables de coefficient d'altérabilité spécifiées avec l'argument var.
- **benchmarks** : copie du *data frame* d'entrée des étalons (à l'exclusion des étalons non valides, le cas échéant). Les variables BY spécifiées avec l'argument by sont incluses dans le *data frame* mais pas les variables de coefficient d'altérabilité spécifiées avec l'argument with.

- **graphTable** : *data frame* contenant des données supplémentaires utiles pour produire des tableaux et des graphiques analytiques (voir la fonction `plot_graphTable()`). Il contient les variables suivantes en plus des variables BY spécifiées avec l'argument `by` :
  - `varSeries` : Nom de la variable de la série indicatrice
  - `varBenchmarks` : Nom de la variable des étalons
  - `altSeries` : Nom de la variable des coefficients d'altérabilité définis par l'utilisateur pour la série indicatrice
  - `altSeriesValue` : Coefficients d'altérabilité de la série indicatrice
  - `altbenchmarks` : Nom de la variable des coefficients d'altérabilité définis par l'utilisateur pour les étalons
  - `altBenchmarksValue` : Coefficients d'altérabilité des étalons
  - `t` : Identificateur de la période de la série indicatrice (1 à T)
  - `m` : Identificateur des périodes de couverture de l'étalon (1 à M)
  - `year` : Année civile du point de données
  - `period` : Valeur de la période (du cycle) du point de données (1 à `periodicity`)
  - `rho` : Paramètre autorégressif ρ (argument `rho`)
  - `lambda` : Paramètre du modèle d'ajustement λ (argument `lambda`)
  - `bias` : Ajustement du biais (par défaut, défini par l'utilisateur ou biais estimé selon les arguments `biasOption` et `bias`)
  - `periodicity` : Le nombre maximum de périodes dans une année (par exemple 4 pour une série indicatrice trimestrielle)
  - `date` : Chaîne de caractères combinant les valeurs des variables `year` et `period`
  - `subAnnual` : Valeurs de la série indicatrice
  - `benchmarked` : Valeurs de la série étalonnée
  - `avgBenchmark` : Valeurs des étalons divisées par le nombre de périodes de couverture
  - `avgSubAnnual` : Valeurs moyennes de la série indicatrice (variable `subAnnual`) pour les périodes couvertes par les étalons
  - `subAnnualCorrected` : Valeurs de la série indicatrice corrigée pour le biais
  - `benchmarkedSubAnnualRatio` : Différence ($\lambda = 0$) ou ratio ($\lambda \neq 0$) des valeurs des variables `benchmarked` et `subAnnual`
  - `avgBenchmarkSubAnnualRatio` : Différence ($\lambda = 0$) ou ratio ($\lambda \neq 0$) des valeurs des variables `avgBenchmark` et `avgSubAnnual`
  - `growthRateSubAnnual` : Différence ($\lambda = 0$) ou différence relative ($\lambda \neq 0$) d'une période à l'autre des valeurs de la série indicatrice (variable `subAnnual`)
  - `growthRateBenchmarked` : Différence ($\lambda = 0$) ou différence relative ($\lambda \neq 0$) d'une période à l'autre des valeurs de la série étalonnée (variable `benchmarked`)
- **splineKnots** : ensemble de coordonnées `x` et `y` (nœuds) utilisées pour estimer la spline cubique naturelle avec la fonction `stats::spline()`. En plus de l'ensemble original de nœuds correspondant aux étalons (points d'ancrage) contraignants, des nœuds supplémentaires sont également ajoutés au début et à la fin afin de traiter le *problème d'actualité* de l'étalonnage et d'approximer une spline de *pente=0* aux deux extrémités (voir section **Détails**). Il contient les variables suivantes en plus des variables BY spécifiées avec l'argument `by` :
  - `varSeries` : Nom de la variable de la série indicatrice
  - `varBenchmarks` : Nom de la variable des étalons

  - x : Coordonnée x de la spline cubique
  - y : Coordonnée y de la spline cubique
  - extraKnot : Valeur logique (*logical*) identifiant les nœuds supplémentaires ajoutés au début et à la fin.

  Les enregistrements pour lesquels extraKnot == FALSE correspondent aux enregistrements du *data frame* de sortie **graphTable** pour lesquels m n'est pas manquant (pas NA), avec x = t et y = benchmarkedSubAnnualRatio.

Notes :

- Le *data frame* de sortie **benchmarks** contient toujours les étalons originaux fournis dans le *data frame* d'entrée des étalons. Les étalons modifiés non contraignants, le cas échéant, peuvent être récupérés (calculés) à partir du *data frame* de sortie **series**.
- La fonction renvoie un objet NULL si une erreur se produit avant que le traitement des données ne puisse commencer. Dans le cas contraire, si l'exécution est suffisamment avancée pour que le traitement des données puisse commencer, alors un objet incomplet sera renvoyé en cas d'erreur (par exemple, un *data frame* de sortie **series** avec des valeurs NA pour les données étalonnées).
- La fonction renvoie des objets « data.frame » qui peuvent être explicitement convertis en d'autres types d'objets avec la fonction as*() appropriée (ex., tibble::as_tibble() convertirait n'importe lequel d'entre eux en tibble).

References

Statistique Canada (2012). « Chapitre 5 : Étalonnage de stocks ». **Théorie et application de l'étalonnage (Code du cours 0436)**. Statistique Canada, Ottawa, Canada.

See Also

benchmarking() plot_graphTable() bench_graphs plot_benchAdj()

Examples

```
# Série de stocks trimestriels (même patron répété chaque année)
mes_ind <- ts_to_tsDF(ts(rep(c(85, 95, 125, 95), 7),
                         start = c(2013, 1),
                         frequency = 4))
head(mes_ind)

# Étalons annuels (stocks de fin d'année)
mes_eta <- ts_to_bmkDF(ts(c(135, 125, 155, 145, 165),
                          start = 2013,
                          frequency = 1),
                       discrete_flag = TRUE,
                       alignment = "e",
                       ind_frequency = 4)
mes_eta

# Étalonnage avec...
#   - valeur de `rho` recommandée pour des séries trimestrielles (`rho = 0.729`)
```

```
#   - modèle proportionnel (`lambda = 1`)
#   - correction de la série indicatrice pour le biais avec estimation du biais
#     (`biasOption = 3`)

# ... avec `benchmarking()` (approche « Proc Benchmarking »)
res_PB <- benchmarking(mes_ind,
                       mes_eta,
                       rho = 0.729,
                       lambda = 1,
                       biasOption = 3)

# ... avec `stock_benchmarking()` (approche « Stock Benchmarking »)
res_SB <- stock_benchmarking(mes_ind,
                             mes_eta,
                             rho = 0.729,
                             lambda = 1,
                             biasOption = 3)

# Comparer les ajustements d'étalonnage des deux approches
plot_benchAdj(PB_graphTable = res_PB$graphTable,
              SB_graphTable = res_SB$graphTable)

# Avez-vous remarqué que les ajustements de `stock_benchmarking()` sont plus lisses
# que ceux de `benchmarking()` ?

# L'amélioration de la qualité des données étalonnées qui en résulte n'est pas
# nécessairement évidente dans cet exemple.
plot(res_SB$graphTable$t, res_SB$graphTable$benchmarked,
     type = "b", col = "red", xlab = "t", ylab = "Stocks étalonnés")
lines(res_PB$graphTable$t, res_PB$graphTable$benchmarked,
      type = "b", col = "blue")
legend(x = "topleft", bty = "n", inset = 0.05, lty = 1, pch = 1,
       col = c("red", "blue"), legend = c("res_SB", "res_PB"))
title("Stocks étalonnés")

# Qu'en est-il des cas où un indicateur plat (rectiligne) est utilisé, ce qui se produit
# souvent en pratique en l'absence d'un bon indicateur des mouvements infra-annuels ?
mes_inds2 <- mes_ind
mes_inds2$value <- 1  # indicateur plat
res_PB2 <- benchmarking(mes_inds2,
                        mes_eta,
                        rho = 0.729,
                        lambda = 1,
                        biasOption = 3,
                        quiet = TRUE)  # ne pas afficher l'en-tête

res_SB2 <- stock_benchmarking(mes_inds2,
                              mes_eta,
                              rho = 0.729,
                              lambda = 1,
                              biasOption = 3,
                              quiet = TRUE)  # ne pas afficher l'en-tête
```

```
plot(res_SB2$graphTable$t, res_SB2$graphTable$benchmarked,
     type = "b", col = "red", xlab = "t", ylab = "Stocks étalonnés")
lines(res_PB2$graphTable$t, res_PB2$graphTable$benchmarked,
      type = "b", col = "blue")
legend(x = "topleft", bty = "n", inset = 0.05, lty = 1, pch = 1,
       col = c("red", "blue"), legend = c("res_SB2", "res_PB2"))
title("Stocks étalonnés - Indicateur plat")

# L'apparence plutôt étrange des valeurs étalonnées produites par `benchmarking()` devient
# soudainement plus évidente. En effet, la série étalonnée correspond aux ajustements
# d'étalonnage lorsqu'on utilise un indicateur plat (par exemple, une série de 1 avec
# un étalonnage proportionnel) :
plot_benchAdj(PB_graphTable = res_PB2$graphTable,
              SB_graphTable = res_SB2$graphTable)

# Les lacunes de l'approche « Proc Benchmarking » (fonction `benchmarking()`) avec
# des stocks sont également très visibles lorsque l'on regarde les taux de croissance
# trimestriels résultants, qui sont commodément produits par `plot_graphTable()`.
# Portez une attention particulière à la transition des taux de croissance de T4 à T1
# à chaque année dans les graphiques PDF générés.
plot_graphTable(res_PB2$graphTable, file.path(tempdir(), "Stock_ind_plat_PB.pdf"))
plot_graphTable(res_SB2$graphTable, file.path(tempdir(), "Stock_ind_plat_SB.pdf"))


# Illustrer l'approximation d'une spline cubique naturelle aux nœuds d'extrémité originaux
# (premier et dernier étalons) en spécifiant une grande valeur pour `low_freq_periodicity`.
res_SB3 <- stock_benchmarking(mes_ind,
                              mes_eta,
                              rho = 0.729,
                              lambda = 1,
                              biasOption = 3,

                             # Grande valeur pour approximer une spline cubique naturelle
                              low_freq_periodicity = 100,

                              quiet = TRUE)

plot_benchAdj(SB_graphTable = res_SB3$graphTable,
              SB_splineKnots = res_SB3$splineKnots,
              legendPos = "topleft")


# Illustrer les « oscillations » pour les ajustements projetés au-delà des nœuds
# d'extrémité originaux avec l'étalonnage de type Denton (`rho ~ 1`) causées par
# l'utilisation de nœuds supplémentaires de basse fréquence (annuelle).
res_SB4 <- stock_benchmarking(mes_ind,
                              mes_eta,
                              rho = 0.999,
                              lambda = 1,
                              biasOption = 3,

                              # Utiliser d'abord 3 noœuds supplémentaires annuels
                              n_low_freq_proj = 3,
```

```
                              proj_knots_rho_bd = 1,

                              quiet = TRUE)

plot_benchAdj(SB_graphTable = res_SB4$graphTable,
              SB_splineKnots = res_SB4$splineKnots)

# Pas d'« oscillations » avec la valeur par défaut de `proj_knots_rho_bd` parce que
# des nœuds supplémentaires de haute fréquence (trimestrielle) sont utilisés immédiatement
# (`n_low_freq_proj` est ignoré) puisque `rho = 0.999` excède la valeur par défaut de
# `proj_knots_rho_bd` (0.995^3 pour des données trimestrielles). Ces ajustements projetés
# correspondent davantage à des ajustements de type Denton (en ligne droite).
res_SB4b <- stock_benchmarking(mes_ind,
                               mes_eta,
                               rho = 0.999,
                               lambda = 1,
                               biasOption = 3,
                               quiet = TRUE)

plot_benchAdj(SB_graphTable = res_SB4b$graphTable,
              SB_splineKnots = res_SB4b$splineKnots)


# Illustrer les « contorsions » de la spline cubique autour des nœuds d'extrémité originaux
# causées par l'utilisation immédiate de nœuds supplémentaires de haute fréquence
# (`n_low_freq_proj = 0`), c.à-d., en utilisant les mêmes ajustements projetés que ceux qui
# seraient obtenus avec `benchmarking()`.
#
# Pour exacerber le phénomène, nous utiliserons des données mensuelles (11 périodes entre
# chaque étalon annuel contre seulement 3 pour des données trimestrielles, c.-à-d., une
# spline moins contrainte) et une valeur plutôt faible de `rho` (0.5 < 0.9 = valeur
# recommandée pour des données mensuelles) pour une convergence plus rapide vers le biais
# des ajustements projetés.
vec_ans <- unique(mes_ind$year)
mes_ind3 <- data.frame(year = rep(vec_ans, each = 12),
                       period = rep(1:12, length(vec_ans)),
                       value = rep(1, 12 * length(vec_ans)))  # indicateur plat
mes_eta2 <- mes_eta
mes_eta2[c("startPeriod", "endPeriod")] <- 12

res_SB5 <- stock_benchmarking(mes_ind3,
                              mes_eta2,
                              rho = 0.5,
                              lambda = 1,
                              biasOption = 3,

                              # Utilisation immédiate de noœuds supplémentaires mensuels
                              n_low_freq_proj = 0,

                              quiet = TRUE)

plot_benchAdj(SB_graphTable = res_SB5$graphTable,
              SB_splineKnots = res_SB5$splineKnots)
```

```
# Pas de « contorsions » excessives autour des nœuds d'extrémité originaux avec la valeur
# par défaut `n_low_freq_proj = 1`, c.-à-d., utiliser d'abord 1 nœud supplémentaire de
# basse fréquence (annuelle).
res_SB5b <- stock_benchmarking(mes_ind3,
                               mes_eta2,
                               rho = 0.5,
                               lambda = 1,
                               biasOption = 3,
                               quiet = TRUE)

plot_benchAdj(SB_graphTable = res_SB5b$graphTable,
              SB_splineKnots = res_SB5b$splineKnots)

# Afin de mettre encore mieux en évidence les « contorsions » excessives potentielles de
# la spline cubique lorsqu'on impose les ajustements projetés de `benchmarking()` (c.-à-d.,
# des nœuds supplémentaires de basse fréquence immédiats avec `n_low_freq_proj = 0`),
# traçons les deux précédents ensembles d'ajustements sur le même graphique (la ligne
# bleue correspond ici au cas `n_low_freq_proj = 0`, soit les ajustements projetés de
# `benchmarking()` alors que la ligne rouge correspond aux ajustements par défaut de
# `stock_benchmarking()`, soit `n_low_freq_proj = 1`).
plot_benchAdj(PB_graphTable = res_SB5$graphTable,
              SB_graphTable = res_SB5b$graphTable,
              legend = NULL)
```

`time_values_conv` *Fonctions de conversion de valeurs de temps*

Description

Fonctions de conversion de valeurs de temps

Usage

```
gs.time2year(ts)

gs.time2per(ts)

gs.time2str(ts, sep = "-")
```

Arguments

`ts`	(obligatoire) Objet de type série chronologique (« ts » ou « mts ») ou objet compatible.
`sep`	(optionnel) Chaîne de caractères (constante de type caractère) spécifiant le séparateur à utiliser entre les valeurs d'année et de période (`"-"` par défaut).

Value

`gs.time2year()` renvoie un vecteur de nombres entiers correspondant à l'année (l'unité de temps) « la plus proche ». Cette fonction est l'équivalent de `stats::cycle()` pour les valeurs d'unités de temps.

`gs.time2per()` renvoie un vecteur de nombres entiers contenant les valeurs des périodes (cycles; voir `stats::cycle()`)).

`gs.time2str()` renvoie un vecteur de chaînes de caractères correspondant à `gs.time2year(ts)` lorsque `stats::frequency(ts) == 1` ou à `gs.time2year(ts)` et `gs.time2per(ts)` séparé par `sep` dans le cas contraire.

See Also

`ts_to_tsDF()` `ts_to_bmkDF()` `gs.build_proc_grps()`

Examples

```
# Série chronologique mensuelle « bidon »
sc_men <- ts(rep(NA, 15), start = c(2019, 1), frequency = 12)
sc_men
gs.time2year(sc_men)
gs.time2per(sc_men)
gs.time2str(sc_men)
gs.time2str(sc_men, sep = "m")

# Série chronologique trimestrielle « bidon »
sc_tri <- ts(rep(NA, 5), start = c(2019, 1), frequency = 4)
sc_tri
gs.time2year(sc_tri)
gs.time2per(sc_tri)
gs.time2str(sc_tri)
gs.time2str(sc_tri, sep = "t")
```

---

tsbalancing *Rétablir les contraintes linéaires transversales (contemporaines)*

Description

Réplication de la macro **GSeriesTSBalancing** *de G-Séries 2.0 en SAS®. Voir la documentation de G-Séries 2.0 pour plus de détails (Statistique Canada 2016).*

Cette fonction équilibre (réconcilie) un système de séries chronologiques selon un ensemble de contraintes linéaires. La solution d'équilibrage (« *balancing* ») est obtenue en résolvant un ou plusieurs problèmes de minimisation quadratique (voir la section **Détails**) avec le solveur OSQP (Stellato et al. 2020). Étant donné la faisabilité du (des) problème(s) d'équilibrage, les données des séries chronologiques résultantes respectent les contraintes spécifiées pour chaque période. Des contraintes linéaires d'égalité et d'inégalité sont permises. Optionnellement, la préservation des totaux temporels peut également être spécifiée.

Usage

```
tsbalancing(
  in_ts,
  problem_specs_df,
  temporal_grp_periodicity = 1,
  temporal_grp_start = 1,
  osqp_settings_df = default_osqp_sequence,
  display_level = 1,
  alter_pos = 1,
  alter_neg = 1,
  alter_mix = 1,
  alter_temporal = 0,
  lower_bound = -Inf,
  upper_bound = Inf,
  tolV = 0,
  tolV_temporal = 0,
  tolP_temporal = NA,

  # Nouveau dans G-Séries 3.0
  validation_tol = 0.001,
  trunc_to_zero_tol = validation_tol,
  full_sequence = FALSE,
  validation_only = FALSE,
  quiet = FALSE
)
```

Arguments

`in_ts` (obligatoire)

Objet de type série chronologique (« ts » ou « mts »), ou objet compatible, qui contient les données des séries chronologiques à réconcilier. Il s'agit des données d'entrée (solutions initiales) des problèmes d'équilibrage (« *balancing* »).

`problem_specs_df`

(obligatoire)

Data frame des spécifications du problème d'équilibrage. En utilisant un format clairsemé (épars) inspiré de la procédure LP de SAS/OR® (SAS Institute 2015), il ne contient que les informations pertinentes telles que les coefficients non nuls des contraintes d'équilibrage ainsi que les coefficients d'altérabilité et les bornes inférieures/supérieures à utiliser au lieu des valeurs par défaut (c.-à-d., les valeurs qui auraient la priorité sur celles définies avec les arguments `alter_pos`, `alter_neg`, `alter_mix`, `alter_temporal`, `lower_bound` et `upper_bound`).

Les informations sont fournies à l'aide de quatre variables obligatoires (`type`, `col`, `row` et `coef`) et d'une variable facultative (`timeVal`). Un enregistrement (une rangée) dans le *data frame* des spécifications du problème définit soit une étiquette pour l'un des sept types d'éléments du problème d'équilibrage avec les colonnes `type` et `row` (voir *Enregistrements de définition d'étiquette* ci-dessous) ou bien spécifie des coefficients (valeurs numériques) pour ces éléments du

problème d'équilibrage avec les variables `col`, `row`, `coef` et `timeVal` (voir *Enregistrements de spécification d'information* ci-dessous).

- **Enregistrements de définition d'étiquette** (`type` n'est pas manquant (n'est pas `NA`))
  - `type` (car) : mot-clé réservé identifiant le type d'élément du problème en cours de définition :
    - `EQ` : contrainte d'équilibrage d'égalité ($=$)
    - `LE` : contrainte d'équilibrage d'inégalité de type inférieure ou égale ($\leq$)
    - `GE` : contrainte d'équilibrage d'inégalité de type supérieure ou égale ($\geq$)
    - `lowerBd` : borne inférieure des valeurs de période
    - `upperBd` : borne supérieure des valeurs de période
    - `alter` : coefficient d'altérabilité des valeurs de période
    - `alterTmp` : coefficient d'altérabilité des totaux temporels
  - `row` (car) : étiquette à associer à l'élément du problème (*mot-clé* `type`)
  - *toutes les autres variables ne sont pas pertinentes et devraient contenir des données manquantes (valeurs* `NA`*)*

- **Enregistrements de spécification d'information** (`type` est manquant (est `NA`))
  - `type` (car) : non applicable (`NA`)
  - `col` (car) : nom de la série ou mot réservé `_rhs_` pour spécifier la valeur du côté droit (*RHS* pour **R***ight-***H***and* **S***ide*) d'une contrainte d'équilibrage.
  - `row` (car) : étiquette de l'élément du problème.
  - `coef` (num) : valeur de l'élément du problème :
    - coefficient de la série dans la contrainte d'équilibrage ou valeur *RHS*
    - borne inférieure ou supérieure des valeurs de période de la série
    - coefficient d'altérabilité des valeurs de période ou des totaux temporels de la série
  - `timeVal` (num) : valeur de temps optionnelle pour restreindre l'application des bornes ou coefficients d'altérabilité des séries à une période (ou groupe temporel) spécifique. Elle correspond à la valeur de temps, telle que renvoyée par `stats::time()`, pour une période (observation) donnée des séries chronologiques d'entrée (argument `in_ts`) et correspond conceptuellement à $année + (période - 1)/fréquence$.

Notez que les chaînes de caractères vides (`""` ou `''`) pour les variables de type caractère sont interprétées comme manquantes (`NA`) par la fonction. La variable `row` identifie les éléments du problème d'équilibrage et est la variable clé qui fait le lien entre les deux types d'enregistrements. La même étiquette (`row`) ne peut être associée à plus d'un type d'éléments du problème (`type`) et plusieurs étiquettes (`row`) ne peuvent pas être définies pour un même type d'éléments du problème donné (`type`), à l'exception des contraintes d'équilibrage (valeurs

"EQ", "LE" et "GE" de la colonne type). Voici certaines caractéristiques conviviales du *data frame* des spécifications du problème :

- L'ordre des enregistrements (rangées) n'est pas important.
- Les valeurs des variables de type caractère (type, row et col) ne sont pas sensibles à la casse (ex., les chaînes de caractères "Constraint 1" et "CONSTRAINT 1" pour la variable row seraient considérées comme une même étiquette d'élément du problème), sauf lorsque col est utilisé pour spécifier un nom de série (une colonne de l'objet d'entrée de type série chronologique) où **la sensibilité à la casse est appliquée**.
- Les noms des variables du *data frame* des spécifications du problème ne sont pas non plus sensibles à la casse (ex., type, Type ou TYPE sont tous des noms de variable valides) et time_val est un nom de variable accepté (au lieu de timeVal).

Enfin, le tableau suivant dresse la liste des alias valides (acceptés) pour les *mots-clés* type (type d'éléments du problème) :

Mot-clé	**Alias**
EQ	==, =
LE	<=, <
GE	>=, >
lowerBd	lowerBound, lowerBnd, + *mêmes termes avec '_', '.' ou ' ' entre les mots*
upperBd	upperBound, upperBnd, + *mêmes termes avec '_', '.' ou ' ' entre les mots*
alterTmp	alterTemporal, alterTemp, + *mêmes termes avec '_', '.' ou ' ' entre les mots*

L'examen des **Exemples** devrait aider à conceptualiser le *data frame* des spécifications du problème d'équilibrage.

temporal_grp_periodicity

(optionnel)

Nombre entier positif définissant le nombre de périodes dans les groupes temporels pour lesquels les totaux doivent être préservés. Par exemple, spécifiez temporal_grp_periodicity = 3 avec des séries chronologiques mensuelles pour la préservation des totaux trimestriels et temporal_grp_periodicity = 12 (ou temporal_grp_periodicity = frequency(in_ts)) pour la préservation des totaux annuels. Spécifier temporal_grp_periodicity = 1 (*défaut*) correspond à un traitement période par période sans préservation des totaux temporels.

La valeur par défaut est temporal_grp_periodicity = 1 (traitement période par période sans préservation des totaux temporels).

temporal_grp_start

(optionnel)

Entier dans l'intervalle [1 .. temporal_grp_periodicity] spécifiant la période (cycle) de départ pour la préservation des totaux temporels. Par exemple, des totaux annuels correspondant aux années financières définies d'avril à mars de l'année suivante seraient spécifiés avec temporal_grp_start = 4 pour des séries chronologiques mensuelles (frequency(in_ts) = 12) et temporal_grp_start = 2 pour des séries chronologiques trimestrielles (frequency(in_ts) = 4). Cet

argument n'a pas d'effet pour un traitement période par période sans préservation des totaux temporels (`temporal_grp_periodicity = 1`).

La valeur par défaut est `temporal_grp_start = 1`.

`osqp_settings_df`

(optionnel)

Data frame contenant une séquence de paramètres d'OSQP pour la résolution des problèmes d'équilibrage. La librairie inclut deux *data frames* prédéfinis de séquences de paramètres d'OSQP :

- default_osqp_sequence : rapide et efficace (par défaut);
- alternate_osqp_sequence : orienté vers la précision au détriment du temps d'exécution.

Voir la `vignette("osqp-settings-sequence-dataframe")` pour plus de détails sur ce sujet et pour voir le contenu de ces deux *data frames*. Notez que le concept d'une *séquence de résolution* avec différents ensembles de paramètres pour le solveur est nouveau dans G-Séries 3.0 (une seule tentative de résolution était effectuée dans G-Séries 2.0).

La valeur par défaut est `osqp_settings_df = default_osqp_sequence`.

`display_level` (optionnel)

Entier dans l'intervalle [0 .. 3] spécifiant le niveau d'information à afficher dans la console (`stdout()`). Notez que spécifier l'argument `quiet = TRUE` annulerait l'argument `display_level` (aucune des informations suivantes ne serait affichée).

Information affichée	`0`	`1`	`2`	`3`
En-tête de la fonction	✓	✓	✓	✓
Éléments du problème d'équilibrage		✓	✓	✓
Détails de résolution de chaque problème			✓	✓
Résultats de chaque problème (valeurs et contraintes)				✓

La valeur par défaut est `display_level = 1`.

`alter_pos` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut associé aux valeurs des séries chronologiques avec des coefficients **positifs** dans toutes les contraintes d'équilibrage dans lesquelles elles sont impliquées (ex., les séries composantes dans les problèmes de ratissage (« *raking* ») de tables d'agrégation). Les coefficients d'altérabilité fournis dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.

La valeur par défaut est `alter_pos = 1.0` (valeurs non contraignantes).

`alter_neg` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut associé aux valeurs des séries chronologiques avec des coefficients **négatifs** dans toutes les contraintes d'équilibrage dans lesquelles elles sont impliquées (ex., les séries de total de marge dans les problèmes de ratissage (« *raking* ») de tables d'agrégation). Les coefficients d'altérabilité fournis dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.

La valeur par défaut est `alter_neg = 1.0` (valeurs non contraignantes).

`alter_mix` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut associé aux valeurs des séries chronologiques avec un mélange de coefficients **positifs et négatifs** dans les contraintes d'équilibrage dans lesquelles elles sont impliquées. Les coefficients d'altérabilité fournis dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.

La valeur par défaut est `alter_mix = 1.0` (valeurs non contraignantes).

`alter_temporal` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut associé aux totaux temporels des séries chronologiques. Les coefficients d'altérabilité fournis dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.

La valeur par défaut est `alter_temporal = 0.0` (valeurs contraignantes).

`lower_bound` (optionnel)

Nombre réel spécifiant la borne inférieure par défaut pour les valeurs des séries chronologiques. Les bornes inférieures fournies dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.

La valeur par défaut est `lower_bound = -Inf` (non borné).

`upper_bound` (optionnel)

Nombre réel spécifiant la borne supérieure par défaut pour les valeurs des séries chronologiques. Les bornes supérieures fournies dans le *data frame* des spécifications du problème (argument `problem_specs_df`) remplacent cette valeur.

La valeur par défaut est `upper_bound = Inf` (non borné).

`tolV` (optionnel)

Nombre réel non négatif spécifiant la tolérance, en valeur absolue, de la valeur du côté droit (*RHS*) des contraintes d'équilibrage :

- Contraintes `EQ` : $A\mathbf{x} = \mathbf{b}$ devient $\mathbf{b} - \epsilon \leq A\mathbf{x} \leq \mathbf{b} + \epsilon$
- Contraintes `LE` : $A\mathbf{x} \leq \mathbf{b}$ devient $A\mathbf{x} \leq \mathbf{b} + \epsilon$
- Contraintes `GE` : $A\mathbf{x} \geq \mathbf{b}$ devient $A\mathbf{x} \geq \mathbf{b} - \epsilon$

où ϵ est la tolérance spécifiée avec `tolV`. Cet argument ne s'applique pas aux *bornes (inférieures et supérieures) des valeurs de période* spécifiées avec les arguments `lower_bound` et `upper_bound` ou dans le *data frame* des spécifications du problème (argument `prob_specs_df`). Autrement dit, `tolV` n'affecte pas les bornes inférieure et supérieure des valeurs des séries chronologiques, à moins qu'elles ne soient spécifiées comme *contraintes d'équilibrage* à la place (avec des contraintes `GE` et `LE` dans le *data frame* des spécifications du problème).

La valeur par défaut est `tolV = 0.0` (pas de tolérance).

`tolV_temporal, tolP_temporal`

(optionnel)

Nombre réel non négatif, ou `NA`, spécifiant la tolérance, en pourcentage (`tolP_temporal`) ou en valeur absolue (`tolV_temporal`), pour les contraintes implicites d'agrégation temporelle associées aux **totaux temporels contraignants** ($\sum_t x_{i,t} = \sum_t y_{i,t}$), qui deviennent :

$$\sum_t y_{i,t} - \epsilon_{\text{abs}} \leq \sum_t x_{i,t} \leq \sum_t y_{i,t} + \epsilon_{\text{abs}}$$

ou

$$\sum_t y_{i,t}\,(1-\epsilon_{\text{rel}}) \le \sum_t x_{i,t} \le \sum_t y_{i,t}\,(1+\epsilon_{\text{rel}})$$

où ϵ_{abs} et ϵ_{rel} sont les tolérances absolues et en pourcentage spécifiées respectivement avec `tolV_temporal` et `tolP_temporal`. Les deux arguments ne peuvent pas être spécifiés tous les deux à la fois (l'un doit être spécifié tandis que l'autre doit être `NA`).

Exemple : pour une tolérance de 10 *unités*, spécifiez `tolV_temporal = 10, tolP_temporal = NA`; pour une tolérance de 1%, spécifiez `tolV_temporal = NA, tolP_temporal = 0.01`.

Les valeurs par défaut sont `tolV_temporal = 0.0` et `tolP_temporal = NA` (pas de tolérance).

`validation_tol` (optionnel)

Nombre réel non négatif spécifiant la tolérance pour la validation des résultats d'équilibrage. La fonction vérifie si les valeurs finales des séries chronologiques (réconciliées) satisfont les contraintes, en autorisant des écarts jusqu'à la valeur spécifiée avec cet argument. Un avertissement est émis dès qu'une contrainte n'est pas respectée (écart supérieur à `validation_tol`).

Avec des contraintes définies comme $\mathbf{l} \le A\mathbf{x} \le \mathbf{u}$, où $\mathbf{l} = \mathbf{u}$ pour les contraintes `EQ`, $\mathbf{l} = -\infty$ pour les contraintes `LE` et $\mathbf{u} = \infty$ pour les contraintes `GE`, **les écarts de contraintes** correspondent à $\max(0, \mathbf{l} - A\mathbf{x}, A\mathbf{x} - \mathbf{u})$, où les bornes de contraintes $\mathbf{l}$ et $\mathbf{u}$ incluent les tolérances, le cas échéant, spécifiées avec les arguments `tolV`, `tolV_temporal` et `tolP_temporal`.

La valeur par défaut est `validation_tol = 0.001`.

`trunc_to_zero_tol`

(optionnel)

Nombre réel non négatif spécifiant la tolérance, en valeur absolue, pour le remplacement par zéro de (petites) valeurs dans les données (réconciliées) de séries chronologiques de sortie (objet de sortie `out_ts`). Spécifiez `trunc_to_zero_tol = 0` pour désactiver ce processus de *troncation à zéro* des données réconciliées. Sinon, spécifiez `trunc_to_zero_tol > 0` pour remplacer par 0.0 toute valeur dans l'intervalle $[-\epsilon, \epsilon]$, où ϵ est la tolérance spécifiée avec `trunc_to_zero_tol`.

Notez que les écarts de contraintes finaux (voir l'argument `validation_tol`) sont calculées sur les séries chronologiques réconciliées *tronquées à zéro*, ce qui garantit une validation précise des données réconciliées réelles renvoyées par la fonction.

La valeur par défaut est `trunc_to_zero_tol = validation_tol`.

`full_sequence` (optionnel)

Argument logique (*logical*) spécifiant si toutes les étapes du *data frame pour la séquence de paramètres d'OSQP* doivent être exécutées ou non. Voir l'argument `osqp_settings_df` et la `vignette("osqp-settings-sequence-dataframe")` pour plus de détails sur ce sujet.

La valeur par défaut est `full_sequence = FALSE`.

`validation_only`

(optionnel)

Argument logique (*logical*) spécifiant si la fonction doit uniquement effectuer la validation des données d'entrée ou non. Lorsque `validation_only = TRUE`, les *contraintes d'équilibrage* et les *bornes (inférieures et supérieures) des valeurs de période* spécifiées sont validées par rapport aux données de séries chronologiques d'entrée, en permettant des écarts jusqu'à la valeur spécifiée avec l'argument `validation_tol`. Sinon, lorsque `validation_only = FALSE` (par défaut), les données d'entrée sont d'abord réconciliées et les données résultantes (en sortie) sont ensuite validées.

La valeur par défaut est `validation_only = FALSE`.

`quiet` (optionnel)

Argument logique (*logical*) spécifiant s'il faut ou non afficher uniquement les informations essentielles telles que les avertissements, les erreurs et la période (ou l'ensemble de périodes) en cours de traitement. Vous pouvez également supprimer, si vous le souhaitez, l'affichage des informations relatives à la (aux) période(s) en cours de traitement en *enveloppant* votre appel à `tsbalancing()` avec `suppressMessages()`. Dans ce cas, le *data frame* de sortie **proc_grp_df** peut être utilisé pour identifier les problèmes d'équilibrage (infructueux) associés aux messages d'avertissement (le cas échéant). Notez que la spécification de `quiet = TRUE` annulera également l'argument `display_level`.

La valeur par défaut est `quiet = FALSE`.

Details

Cette fonction résout un problème d'équilibrage par groupe de traitement (voir la section **Groupes de traitement** pour plus de détails). Chacun de ces problèmes d'équilibrage est un problème de minimisation quadratique de la forme suivante :

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimiser}} && (\mathbf{y}-\mathbf{x})^{\mathrm{T}} W (\mathbf{y}-\mathbf{x}) \\ &\text{sous contrainte(s)} && \mathbf{l} \leq A\mathbf{x} \leq \mathbf{u} \end{aligned}$$

où

- $\mathbf{y}$ est le vecteur des valeurs initiales du problème, c.-à-d., les valeurs de période initiales et, le cas échéant, les totaux temporels initiaux des séries chronologiques;
- $\mathbf{x}$ est la version finale (réconciliée) du vecteur $\mathbf{y}$;
- la matrice $W = \mathrm{diag}\,(\mathbf{w})$ avec les éléments du vecteur $\mathbf{w}$ définis comme $w_i = \begin{cases} 0 & \text{if } |c_i y_i| = 0 \\ \frac{1}{|c_i y_i|} & \text{sinon} \end{cases}$, où c_i est coefficient d'altérabilité de la valeur du problème y_i et où les cas correspondant à $|c_i y_i| = 0$ sont des valeurs fixes (valeurs de période ou totaux temporels contraignants);
- la matrice A et les vecteurs $\mathbf{l}$ et $\mathbf{u}$ définissent les *contraintes d'équilibrage*, les *contraintes implicites d'agrégation temporelle* (le cas échéant), les *bornes (inférieures et supérieures) des valeurs de période* et *les contraintes $x_i = y_i$ pour les valeurs y_i fixes* ($|c_i y_i| = 0$).

En pratique, la fonction objectif du problème résolu par OSQP exclut le terme constant $\mathbf{y}^{\mathrm{T}} W \mathbf{y}$, correspondant alors à $\mathbf{x}^{\mathrm{T}} W \mathbf{x} - 2\,(\mathbf{w}\mathbf{y})^{\mathrm{T}}\,\mathbf{x}$, et les valeurs y_i fixes ($|c_i y_i| = 0$) sont exclues du problème, en ajustant les contraintes en conséquence, c.-à-d. :

- les lignes correspondant aux *contraintes $x_i = y_i$ pour les valeurs y_i fixes* sont supprimées de A, $\mathbf{l}$ et $\mathbf{u}$;

- les colonnes correspondant aux valeurs y_i fixes sont supprimées de A tout en ajustant de manière appropriée $\mathbf{l}$ et $\mathbf{u}$.

Coefficients d'altérabilité:

Les coefficients d'altérabilité sont des nombres non négatifs qui modifient le coût relatif de la modification d'une valeur initiale du problème. En modifiant la fonction objectif à minimiser, ils permettent de générer un large éventail de solutions. Puisqu'ils apparaissent dans le dénominateur de la fonction objectif (matrice W), plus le coefficient d'altérabilité est élevé, moins il est coûteux de modifier une valeur du problème (valeur de période ou total temporel) et, inversement, plus le coefficient d'altérabilité est petit, plus il devient coûteux de le faire. Il en résulte que les valeurs du problème ayant des coefficients d'altérabilité plus élevés changent proportionnellement plus que celles ayant des coefficients d'altérabilité plus petits. Un coefficient d'altérabilité de 0.0 définit une valeur de problème fixe (contraignante), tandis qu'un coefficient d'altérabilité supérieur à 0.0 définit une valeur libre (non contraignante). Les coefficients d'altérabilité par défaut sont 0.0 pour les totaux temporels (argument `alter_temporal`) et 1.0 pour les valeurs de période (arguments `alter_pos`, `alter_neg`, `alter_mix`). Dans le cas courant des problèmes de ratissage (« *raking* ») de tables d'agrégation, les valeurs de période des totaux de marge (séries chronologiques avec un coefficient de -1 dans les contraintes d'équilibrage) sont généralement contraignantes (spécifié avec `alter_neg = 0`) tandis que les valeurs de période des séries composantes (séries chronologiques avec un coefficient 1 dans les contraintes d'équilibrage) sont généralement non contraignantes (spécifié avec `alter_pos > 0`, ex., `alter_pos = 1`). Des valeurs de problème *presque contraignantes* (ex., pour les totaux de marge ou les totaux temporels) peuvent être obtenues en pratique en spécifiant de très petits (presque 0.0) coefficents d'altérabilité par rapport à ceux des autres valeurs (non contraignantes) du problème.

La préservation des totaux temporels fait référence au fait que les totaux temporels, le cas échéant, sont généralement conservés « aussi près que possible » de leur valeur initiale. Une *préservation pure* est obtenue par défaut avec des totaux temporels contraignants, tandis que le changement est minimisé avec des totaux temporels non contraignants (conformément à l'ensemble de coefficients d'altérabilité utilisés).

Validation et dépannage:

Les problèmes d'équilibrage fructueux (problèmes avec une solution valide) ont `sol_status_val > 0` ou, de manière équivalente, `n_unmet_con = 0` ou `max_discr <= validation_tol` dans le *data frame* de sortie **proc_grp_df**. Le dépannage des problèmes d'équilibrage infructueux n'est pas nécessairement simple. Voici quelques suggestions :

- Examinez les contraintes qui ont échoué (`unmet_flag = TRUE` ou, de manière équivalente, `discr_out > validation_tol` dans le *data frame* de sortie **prob_conf_df**) pour s'assurer qu'elles ne causent pas un espace de solution vide (problème infaisable).
- Modifier la séquence de résolution d'OSQP. Par exemple, essayez :
  1. l' argument `full_sequence = TRUE`
  2. l' argument `osqp_settings_df = alternate_osqp_sequence`
  3. les arguments `osqp_settings_df = alternate_osqp_sequence` et `full_sequence = TRUE`

  Voir la `vignette("osqp-settings-sequence-dataframe")` pour plus de détails sur ce sujet.
- Augmenter (revoir) la valeur de `validation_tol`. Bien que cela puisse ressembler à de la *tricherie*, la valeur par défaut de `validation_tol` (1×10^{-3}) peut en fait être trop petite

pour les problèmes d'équilibrage qui impliquent de très grandes valeurs (ex., en milliards) ou, inversement, trop grande avec des valeurs de problème très petites (ex., < 1.0). Multiplier l'échelle moyenne des données du problème par la *tolérance de la machine* (`.Machine$double.eps`) donne une approximation de la taille moyenne des écarts que `tsbalancing()` devrait être capable de détecter (distinguer de 0) et devrait probablement constituer une **limite inférieure absolue** pour l'argument `validation_tol`. En pratique, une valeur raisonnable de `validation_tol` devrait probablement être de 1×10^3 à 1×10^6 fois plus grande que cette *limite inférieure*.

- S'attaquer aux contraintes redondantes. Les problèmes de ratissage (« *raking* ») de tables d'agrégation multidimensionnelles sont surspécifiés (ils impliquent des contraintes redondantes) lorsque tous les totaux de toutes les dimensions du *cube de données* sont contraignants (fixes) et qu'une contrainte est définie pour chacun d'entre eux. La redondance se produit également pour les contraintes implicites d'agrégation temporelle dans les tables d'agrégation unidimensionnelles ou multidimensionnelles avec des totaux temporels contraignants (fixes). La surspécification n'est généralement pas un problème pour `tsbalancing()` si les données d'entrée ne sont pas contradictoires en ce qui concerne les contraintes redondantes, c'est-à-dire, s'il n'y a pas d'incohérences (d'écarts) associées aux contraintes redondantes dans les données d'entrée ou si elles sont *négligeables* (raisonnablement faibles par rapport à l'échelle des données du problème). Dans le cas contraire, cela peut conduire à des problèmes d'équilibrage infructueux `tsbalancing()`. Les solutions possibles sont alors les suivantes :
  1. Résoudre (ou réduire) les écarts associés aux contraintes redondantes dans les données d'entrée.
  2. Sélectionner un total de marge dans chaque dimension, sauf une, du cube de données et supprimer du problème les contraintes d'équilibrage correspondantes. *Cela ne peut pas être fait pour les contraintes implicites d'agrégation temporelle.*
  3. Sélectionnez un total de marge dans chaque dimension, sauf une, du cube de données et rendez-les non contraignantes (coefficient d'altérabilité de, disons, 1.0).
  4. Faire la même chose que (3) pour les totaux temporels d'une des séries composantes de l'intérieur du cube (les rendre non contraignants).
  5. Rendre tous les totaux de marge de chaque dimension, sauf une, du cube de données *presque contraignants*, c.-à-d., spécifier de très petits coefficients d'altérabilité (disons 1×10^{-6}) par rapport à ceux des séries composantes de l'intérieur du cube.
  6. Faire la même chose que (5) pour les totaux temporels de toutes les séries composantes de l'intérieur du cube (coefficients d'altérabilité très petits, par exemple, avec l'argument `alter_temporal`).
  7. Utilisez `tsraking()` (le cas échéant), qui gère ces incohérences en utilisant l'inverse de Moore-Penrose (distribution uniforme à travers tous les totaux contraignants).

  Les solutions (2) à (7) ci-dessus ne doivent être envisagées que si les écarts associés aux contraintes redondantes dans les données d'entrée sont *raisonnablement faibles* car ils seraient distribués parmi les totaux omis ou non contraignants avec `tsbalancing()` et tous les totaux contraignants avec `tsraking()`. Sinon, il faut d'abord étudier la solution (1) ci-dessus.
- Assouplir (relaxer) les bornes des contraintes du problème, par exemple :
  - avec l'argument `tolV` pour les contraintes d'équilibrage;
  - avec les arguments `tolV_temporal` et `tolP_temporal` pour les contraintes implicites d'agrégation temporelle;
  - avec les arguments `lower_bound` et `upper_bound`.

Value

La fonction renvoie une liste de sept objets :

- **out_ts** : version modifiée de l'objet d'entrée de type série chronologique (« ts » ou « mts »; voir l'argument `in_ts`) contenant les valeurs réconciliées des séries chronologiques qui résultent de l'exécution de la fonction (sortie principale de la fonction). Il peut être explicitement converti en un autre type d'objet avec la fonction `as*()` appropriée (ex., `tsibble::as_tsibble()` le convertirait en tsibble).
- **proc_grp_df** : *data frame* récapitulatif des groupes de traitement, utile pour identifier les problèmes fructueux ou infructueux. Il contient un enregistrement (une rangée) pour chaque problème d'équilibrage avec les colonnes suivantes :
  - `proc_grp` (num) : identificateur du groupe de traitement.
  - `proc_grp_type` (car) : type de groupe de traitement. Les valeurs possibles sont :
    - `"period"` (périodes uniques);
    - `"temporal group"` (groupes temporels).
  - `proc_grp_label` (car) : chaîne de caractères décrivant le groupe de traitement dans le format suivant :
    - `"<year>-<period>"` (périodes uniques)
    - `"<start year>-<start period> - <end year>-<end period>"` (groupes temporels)
  - `sol_status_val`, `sol_status` (num, car) : valeur numérique (entière) et chaîne de caractères associés au statut de la solution :
    - `1` : `"valid initial solution"` (solution initiale valide);
    - `-1` : `"invalid initial solution"` (solution initiale invalide);
    - `2` : `"valid polished osqp solution"` (solution OSQP *raffinée* valide);
    - `-2` : `"invalid polished osqp solution"` (solution OSQP *raffinée* invalide);
    - `3` : `"valid unpolished osqp solution"` (solution OSQP *non raffinée* valide);
    - `-3` : `"invalid unpolished osqp solution"` (solution OSQP *non raffinée* invalide);
    - `-4` : `"unsolvable fixed problem"` (problème fixe insoluble, avec solution initiale invalide).
  - `n_unmet_con` (num) : nombre de contraintes non satisfaites (`sum(prob_conf_df$unmet_flag)`).
  - `max_discr` (num) : écart de contrainte maximal (`max(prob_conf_df$discr_out)`).
  - `validation_tol` (num) : tolérance spécifiée à des fins de validation (argument `validation_tol`).
  - `sol_type` (car) : type de solution renvoyée. Les valeurs possibles sont :
    - `"initial"` (solution initiale, c.-à-d., les valeurs des données d'entrée);
    - `"osqp"` (solution OSQP).
  - `osqp_attempts` (num) : nombre de tentatives effectuées avec OSQP (profondeur atteinte dans la séquence de résolution).
  - `osqp_seqno` (num) : numéro d'étape de la séquence de résolution correspondant à la solution renvoyée. `NA` lorsque `sol_type = "initial"`.
  - `osqp_status` (car) : chaîne de caractères décrivant le statut OSQP (`osqp_sol_info_df$status`). `NA` lorsque `sol_type = "initial"`.
  - `osqp_polished` (logi) : `TRUE` si la solution OSQP renvoyée est *raffinée* (`osqp_sol_info_df $status_polish = 1`), `FALSE` sinon. `NA` lorsque `sol_type = "initial"`.
  - `total_solve_time` (num) : temps total, en secondes, de la séquence de résolution.

La colonne proc_grp constitue une *clé unique* (enregistrements distincts) pour le *data frame*. Les problèmes d'équilibrage fructueux (problèmes avec une solution valide) correspondent aux enregistrements avec sol_status_val > 0 ou, de manière équivalente, à n_unmet_con = 0 ou à max_discr <= validation_tol. La *solution initiale* (sol_type = "initial") n'est renvoyée que si **a)** il n'y a pas de d'écarts de contraintes initiaux, **b)** le problème est fixé (toutes les valeurs sont contraignantes) ou **c)** elle est meilleure que la solution OSQP (total des écarts de contraintes plus faible). La séquence de résolution est décrite dans la vignette("osqp-settings-sequence-dataframe").

- **periods_df** : *data frame* sur les périodes de temps, utile pour faire correspondre les périodes aux groupes de traitement. Il contient un enregistrement (une rangée) pour chaque période de l'objet d'entrée de type série chronologique (argument in_ts) avec les colonnes suivantes :
  - proc_grp (num) : identificateur du groupe de traitement.
  - t (num) : identificateur de la période (1:nrow(in_ts)).
  - time_val (num) : valeur de temps (stats::time(in_ts)). Correspond conceptuellement à $année + (période - 1)/fréquence$.

  Les colonnes t et time_val constituent toutes deux une *clé unique* (enregistrements distincts) pour le *data frame*.
- **prob_val_df** : *data frame* sur les valeurs du problème, utile pour analyser les changements entre les valeurs initiales et finales (réconciliées). Il contient un enregistrement (une rangée) pour chaque valeur impliquée dans chaque problème d'équilibrage, avec les colonnes suivantes :
  - proc_grp (num) : identificateur du groupe de traitement.
  - val_type (car) : type de valeur du problème. Les valeurs possibles sont :
    - "period value" (valeur de période);
    - "temporal total" (total temporel).
  - name (car) : nom de la série chronologique (variable).
  - t (num) : identificateur de la période (1:nrow(in_ts)); identificateur de la première période du groupe temporel pour un *total temporel*.
  - time_val (num) : valeur de temps (stats::time(in_ts)); valeur de la première période du groupe temporel pour un *total temporel*. Correspond conceptuellement à $année + (période - 1)/fréquence$.
  - lower_bd, upper_bd (num) : bornes des valeurs de période; toujours -Inf et Inf pour un *total temporel*.
  - alter (num) : coefficient d'altérabilité.
  - value_in, value_out (num) : valeurs initiales et finales (réconciliées).
  - dif (num) : value_out - value_in.
  - rdif (num) : dif / value_in; NA si value_in = 0.

  Les colonnes val_type + name + t et val_type + name + time_val constituent toutes deux une *clé unique* (enregistrements distincts) pour le *data frame*. Les valeurs contraignantes (fixes) des problèmes correspondent aux enregistrements avec alter = 0 ou value_in = 0. Inversement, les valeurs de problèmes non contraignantes (libres) correspondent aux enregistrements avec alter != 0 et value_in != 0.
- **prob_con_df** : *data frame* sur les contraintes du problème, utile pour dépanner les problèmes infructueux (identifier les contraintes non satisfaites). Il contient un enregistrement (une rangée) pour chaque contrainte impliquée dans chaque problème d'équilibrage, avec les colonnes suivantes :

  - `proc_grp` (num) : identificateur du groupe de traitement.
  - `con_type` (car) : type de contrainte. Les valeurs possibles sont :
    * `"balancing constraint"` (contrainte d'équilibrage);
    * `"temporal aggregation constraint"` (contrainte d'agrégation temporelle);
    * `"period value bounds"` (bornes de valeur de période).

    Alors que les *contraintes d'équilibrage* sont spécifiées par l'utilisateur, les deux autres types de contraintes (*contraintes d'agrégation temporelle* et *bornes de valeur de période*) sont automatiquement ajoutées au problème par la fonction (le cas échéant).
  - `name` (car) : étiquette de la contrainte ou nom de la série chronologique (variable).
  - `t` (num) : identificateur de la période (`1:nrow(in_ts)`); identificateur de la première période du groupe temporel pour une *contrainte d'agrégation temporelle*.
  - `time_val` (num) : valeur de temps (`stats::time(in_ts)`); valeur de la première période du groupe temporel pour une *contrainte d'agrégation temporelle*. Correspond conceptuellement à $année + (période - 1)/fréquence$.
  - `l`, `u`, `Ax_in`, `Ax_out` (num) : éléments de contrainte initiaux et finaux ($\mathbf{l} \leq A\mathbf{x} \leq \mathbf{u}$).
  - `discr_in`, `discr_out` (num) : écarts de contrainte initiaux et finaux ($\max(0, \mathbf{l} - A\mathbf{x}, A\mathbf{x} - \mathbf{u})$).
  - `validation_tol` (num) : tolérance spécifiée à des fins de validation (argument `validation_tol`).
  - `unmet_flag` (logi) : `TRUE` si la contrainte n'est pas satisfaite (`discr_out > validation_tol`), `FALSE` sinon.

  Les colonnes `con_type + name + t` et `con_type + name + time_val` constituent toutes deux une *clé unique* (enregistrements distincts) pour le *data frame*. Les bornes de contrainte $\mathbf{l} = \mathbf{u}$ pour des contraintes `EQ`, $\mathbf{l} = -\infty$ pour des contraintes `LE`, $\mathbf{u} = \infty$ pour des contraintes `GE`, et incluent les tolérances, le cas échéant, spécifiées avec les arguments `tolV`, `tolV_temporal` et `tolP_temporal`.

- **osqp_settings_df** : *data frame* des paramètres d'OSQP. Il contient un enregistrement (une rangée) pour chaque problème (groupe de traitement) résolu avec OSQP (`proc_grp_df$sol_type = "osqp"`), avec les colonnes suivantes :
  - `proc_grp` (num) : identificateur du groupe de traitement.
  - une colonne correspondant à chaque élément de la liste renvoyée par la méthode `osqp::GetParams()` appliquée à un *objet solveur d'OSQP* (objet de classe « osqp_model » tel que renvoyé par `osqp::osqp()`), ex. :
    * Nombre maximal d'itérations (`max_iter`);
    * Tolérances d'infaisabilité primale et duale (`eps_prim_inf` et `eps_dual_inf`);
    * Drapeau d'exécution de l'étape de raffinement de la solution (`polish`);
    * Nombre d'itérations de mise à l'échelle (`scaling`);
    * etc.
  - paramètres supplémentaires spécifiques à `tsbalancing()` :
    * `prior_scaling` (logi) : `TRUE` si les données du problème ont été mises à l'échelle (en utilisant la moyenne des valeurs libres (non contraignantes) du problème comme facteur d'échelle) avant la résolution avec OSQP, `FALSE` sinon.
    * `require_polished` (logi) : `TRUE` si une solution *raffinée* d'OSQP (`osqp_sol_info_df $status_polish = 1`) était nécessaire pour cette étape afin de terminer la séquence de résolution, `FALSE` sinon. Voir la `vignette("osqp-settings-sequence-dataframe")` pour plus de détails sur la séquence de résolution utilisée par `tsbalancing()`.

La colonne `proc_grp` constitue une *clé unique* (enregistrements distincts) pour le *data frame*. Visitez le site `https://osqp.org/docs/interfaces/solver_settings.html` pour tous les paramètres d'OSQP disponibles. Les problèmes (groupes de traitement) pour lesquels la solution initiale a été renvoyée (`proc_grp_df$sol_type = "initial"`) ne sont pas inclus dans ce *data frame*.

- **osqp_sol_info_df** : *data frame* d'informations sur les solutions OSQP. Il contient un enregistrement (une rangée) pour chaque problème (groupe de traitement) résolu avec OSQP (`proc_grp_df$sol_type = "osqp"`), avec les colonnes suivantes :
  - `proc_grp` (num) : identificateur du groupe de traitement.
  - une colonne correspondant à chaque élément de la liste `info` d'un *objet solveur d'OSQP* (objet de classe « osqp_model » tel que renvoyé par `osqp::osqp()`), ex. :
    - Statut de la solution (`status` et `status_val`) ;
    - Statut de raffinement de la solution (`status_polish`) ;
    - Nombre d'itérations (`iter`) ;
    - Valeur de la fonction objectif (`obj_val`) ;
    - Résidus primal et dual (`pri_res` et `dua_res`) ;
    - Temps de résolution (`solve_time`) ;
    - etc.
  - informations supplémentaires spécifiques à `tsbalancing()` :
    - `prior_scaling_factor` (num) : valeur du facteur d'échelle lorsque `osqp_settings_df $prior_scaling = TRUE` (`prior_scaling_factor = 1.0` sinon).
    - `obj_val_ori_prob` (num) : valeur de la fonction objectif du problème d'équilibrage original, qui est la valeur de la fonction objectif d'OSQP (`obj_val`) sur l'échelle originale (lorsque `osqp_settings_df$prior_scaling = TRUE`) plus le terme constant de la fonction objectif du problème d'équilibrage original, c.-à-d., `obj_val_ori_prob = obj_val * prior_scaling_factor + <terme constant>`, où `<terme constant>` correspond à $\mathbf{y}^{\mathrm{T}} W \mathbf{y}$. Voir la section **Détails** pour la définition du vecteur $\mathbf{y}$, de la matrice W et, plus généralement, de l'expression complète de la fonction objectif du problème d'équilibrage.

  La colonne `proc_grp` constitue une *clé unique* (enregistrements distincts) pour le *data frame*. Visitez `https://osqp.org` pour plus d'informations sur OSQP. Les problèmes (groupes de traitement) pour lesquels la solution initiale a été renvoyée (`proc_grp_df$sol_type = "initial"`) ne sont pas inclus dans ce *data frame*.

Notez que les objets de type « data.frame » renvoyés par la fonction peuvent être explicitement convertis en d'autres types d'objets avec la fonction `as*()` appropriée (ex., `tibble::as_tibble()` convertirait n'importe lequel d'entre eux en tibble).

Groupes de traitement

L'ensemble des périodes d'un problème de réconciliation (ratissage ou équilibrage) donné est appelé *groupe de traitement* et correspond soit :

- à une **période unique** lors d'un traitement période par période ou, lorsque les totaux temporels sont préservés, pour les périodes individuelles d'un groupe temporel incomplet (ex., une année incomplète)

- ou à l'**ensemble des périodes d'un groupe temporel complet** (ex., une année complète) lorsque les totaux temporels sont préservés.

Le nombre total de groupes de traitement (nombre total de problèmes de réconciliation) dépend de l'ensemble de périodes des séries chronologiques d'entrée (objet de type série chronologique spécifié avec l'argument `in_ts`) et de la valeur des arguments `temporal_grp_periodicity` et `temporal_grp_start`.

Les scénarios courants incluent `temporal_grp_periodicity = 1` (par défaut) pour un traitement période par période sans préservation des totaux temporels et `temporal_grp_periodicity = frequency(in_ts)` pour la préservation des totaux annuels (années civiles par défaut). L'argument `temporal_grp_start` permet de spécifier d'autres types d'années (*non civile*). Par exemple, des années financières commençant en avril correspondent à `temporal_grp_start = 4` avec des données mensuelles et à `temporal_grp_start = 2` avec des données trimestrielles. La préservation des totaux trimestriels avec des données mensuelles correspondrait à `temporal_grp_periodicity = 3`.

Par défaut, les groupes temporels convrant plus d'une année (c.-à-d., correspondant à `temporal_grp_periodicity > frequency(in_ts)`) débutent avec une année qui est un multiple de `ceiling(temporal_grp_periodicity / frequency(in_ts))`. Par exemple, les groupes bisannuels correspondant à `temporal_grp_periodicity = 2 * frequency(in_ts)` débutent avec une *année paire* par défaut. Ce comportement peut être modifié avec l'argument `temporal_grp_start`. Par exemple, la préservation des totaux bisannuels débutant avec une *année impaire* au lieu d'une *année paire* (par défaut) correspond à `temporal_grp_start = frequency(in_ts) + 1` (avec `temporal_grp_periodicity = 2 * frequency(in_ts)`).

Voir les **Exemples** de `gs.build_proc_grps()` pour des scénarios courants de groupes de traitements.

Comparaison de `tsraking()` et `tsbalancing()`

- `tsraking()` est limitée aux problèmes de ratissage (« *raking* ») de tables d'agrégation unidimensionnelles et bidimensionnelles (avec préservation des totaux temporels si nécessaire) alors que `tsbalancing()` traite des problèmes d'équilibrage plus généraux (ex., des problèmes de ratissage de plus grande dimension, solutions non négatives, contraintes linéaires générales d'égalité et d'inégalité par opposition à des règles d'agrégation uniquement, etc.)
- `tsraking()` renvoie la solution des moindres carrés généralisés du modèle de ratissage basé sur la régression de Dagum et Cholette (Dagum et Cholette 2006) tandis que `tsbalancing()` résout le problème de minimisation quadratique correspondant à l'aide d'un solveur numérique. Dans la plupart des cas, la *convergence vers le minimum* est atteinte et la solution de `tsbalancing()` correspond à la solution (exacte) des moindres carrés de `tsraking()`. Cela peut ne pas être le cas, cependant, si la convergence n'a pas pu être atteinte après un nombre raisonnable d'itérations. Cela dit, ce n'est qu'en de très rares occasions que la solution de `tsbalancing()` différera *significativement* de celle de `tsraking()`.
- `tsbalancing()` est généralement plus rapide que `tsraking()`, en particulier pour les gros problèmes de ratissage, mais est généralement plus sensible à la présence de (petites) incohérences dans les données d'entrée associées aux contraintes redondantes des problèmes de ratissage *entièrement spécifiés* (ou surspécifiés). `tsraking()` gère ces incohérences en utilisant l'inverse de Moore-Penrose (distribution uniforme à travers tous les totaux contraignants).

- `tsbalancing()` permet de spécifier des problèmes épars (clairsemés) sous leur forme réduite. Ce n'est pas le cas de `tsraking()` où les règles d'agrégation doivent toujours être entièrement spécifiées étant donné qu'un *cube de données complet*, sans données manquantes, est attendu en entrée (chaque série composante de l'*intérieur du cube* doit contribuer à toutes les dimensions du cube, c.-à-d., à chaque série totale des *faces extérieures du cube*).
- Les deux outils traitent différemment les valeurs négatives dans les données d'entrée par défaut. Alors que les solutions des problèmes de ratissage obtenues avec `tsbalancing()` et `tsraking()` sont identiques lorsque tous les points de données d'entrée sont positifs, elles seront différentes si certains points de données sont négatifs (à moins que l'argument `Vmat_option = 2` ne soit spécifié avec `tsraking()`).
- Alors que `tsbalancing()` et `tsraking()` permettent toutes les deux de préserver les totaux temporels, la gestion du temps n'est pas incorporée dans `tsraking()`. Par exemple, la construction des groupes de traitement (ensembles de périodes de chaque problème de ratissage) est laissée à l'utilisateur avec `tsraking()` et des appels séparés doivent être soumis pour chaque groupe de traitement (chaque problème de ratissage). De là l'utilité de la fonction d'assistance `tsraking_driver()` pour `tsraking()`.
- `tsbalancing()` renvoie le même ensemble de séries que l'objet d'entrée de type série chronologique (argument `in_ts`) alors que `tsraking()` renvoie l'ensemble des séries impliquées dans le problème de ratissage plus celles spécifiées avec l'argument `id` (qui pourrait correspondre à un sous-ensemble des séries d'entrée).

References

Dagum, E. B. et P. Cholette (2006). **Benchmarking, Temporal Distribution and Reconciliation Methods of Time Series**. Springer-Verlag, New York, Lecture Notes in Statistics, Vol. 186.

Ferland, M., S. Fortier et J. Bérubé (2016). « A Mathematical Optimization Approach to Balancing Time Series: Statistics Canada's GSeriesTSBalancing ». Dans **JSM Proceedings, Business and Economic Statistics Section**. Alexandria, VA: American Statistical Association. 2292-2306.

Ferland, M. (2018). « Time Series Balancing Quadratic Problem — Hessian matrix and vector of linear objective function coefficients ». **Document interne**. Statistique Canada, Ottawa, Canada.

Quenneville, B. et S. Fortier (2012). « Restoring Accounting Constraints in Time Series – Methods and Software for a Statistical Agency ». **Economic Time Series: Modeling and Seasonality**. Chapman & Hall, New York.

SAS Institute Inc. (2015). « The LP Procedure Sparse Data Input Format ». **SAS/OR® 14.1 User's Guide: Mathematical Programming Legacy Procedures**. https://support.sas.com/documentation/cdl/en/ormplpug/68158/HTML/default/viewer.htm#ormplpug_lp_details03.htm

Statistique Canada (2016). « La macro **GSeriesTSBalancing** ». **Guide de l'utilisateur de G-Séries 2.0**. Statistique Canada, Ottawa, Canada.

Statistique Canada (2018). **Théorie et application de la réconciliation (Code du cours 0437)**. Statistique Canada, Ottawa, Canada.

Stellato, B., G. Banjac, P. Goulart et al. (2020). « OSQP: an operator splitting solver for quadratic programs ». **Math. Prog. Comp. 12**, 637–672 (2020). https://doi.org/10.1007/s12532-020-00179-2

See Also

tsraking() tsraking_driver() rkMeta_to_blSpecs() gs.build_proc_grps() build_balancing_problem() aliases

Examples

```
############
# Exemple 1 : Dans ce premier exemple, l'objectif est d'équilibrer un tableau comptable simple
#          (`Profits = Revenus - Depenses`), pour 5 trimestres, sans modifier les `Profits`
#             et où `Revenus >= 0` et `Depenses >= 0`.

# Spécifications du problème
mes_specs1 <- data.frame(type = c("EQ", rep(NA, 3),
                                  "alter", NA,
                                  "lowerBd", NA, NA),
                         col = c(NA, "Revenus", "Depenses", "Profits",
                                 NA, "Profits",
                                 NA, "Revenus", "Depenses"),
                         row = c(rep("Règle comptable", 4),
                                 rep("Coefficient d'altérabilité", 2),
                                 rep("Borne inférieure", 3)),
                         coef = c(NA, 1, -1, -1,
                                  NA, 0,
                                  NA, 0, 0))
mes_specs1

# Données du problème
mes_series1 <- ts(matrix(c( 15,  10,  10,
                               4,   8,  -1,
                            250, 250,   5,
                              8,  12,   0,
                              0,  45, -55),
                         ncol = 3,
                         byrow = TRUE,
                         dimnames = list(NULL, c("Revenus", "Depenses", "Profits"))),
                  start = c(2022, 1),
                  frequency = 4)

# Réconcilier les données
res_equi1 <- tsbalancing(in_ts = mes_series1,
                         problem_specs_df = mes_specs1,
                         display_level = 3)

# Données initiales
mes_series1

# Données réconciliées
res_equi1$out_ts

# Vérifier la présence de solutions invalides
any(res_equi1$proc_grp_df$sol_status_val < 0)
```

```
# Afficher les écarts maximaux des contraintes en sortie
res_equi1$proc_grp_df[, c("proc_grp_label", "max_discr")]


# La solution renvoyée par `tsbalancing()` correspond à des changements proportionnels
# égaux (au prorata) et est associée aux coefficients d'altérabilité par défaut de 1.
# Des changements absolus égaux peuvent être obtenus en spécifiant des coefficients
# d'altérabilité égaux à l'inverse des valeurs initiales.
#
# Faisons cela pour le groupe de traitement 2022T2 (`timeVal = 2022.25`), avec le niveau
# d'information affiché par défaut (`display_level = 1`).

mes_specs1b <- rbind(cbind(mes_specs1,
                           data.frame(timeVal = rep(NA_real_, nrow(mes_specs1)))),
                     data.frame(type = rep(NA, 2),
                                col = c("Revenus", "Depenses"),
                                row = rep("Coefficient d'altérabilité", 2),
                                coef = c(0.25, 0.125),
                                timeVal = rep(2022.25, 2)))
mes_specs1b

res_equi1b <- tsbalancing(in_ts = mes_series1,
                          problem_specs_df = mes_specs1b)

# Afficher les valeurs initiales de 2022T2 et les deux solutions
cbind(data.frame(Statut = c("initial", "prorata", "changement égal")),
      rbind(as.data.frame(mes_series1[2, , drop = FALSE]),
            as.data.frame(res_equi1$out_ts[2, , drop = FALSE]),
            as.data.frame(res_equi1b$out_ts[2, , drop = FALSE])),
      data.frame(Ecart_comptable = c(mes_series1[2, 1] - mes_series1[2, 2] -
                                       mes_series1[2, 3],
                                     res_equi1$out_ts[2, 1] -
                                       res_equi1$out_ts[2, 2] -
                                       res_equi1$out_ts[2, 3],
                                     res_equi1b$out_ts[2, 1] -
                                       res_equi1b$out_ts[2, 2] -
                                       res_equi1b$out_ts[2, 3]),
                 ChgRel_Rev = c(NA,
                                res_equi1$out_ts[2, 1] / mes_series1[2, 1] - 1,
                                res_equi1b$out_ts[2, 1] / mes_series1[2, 1] - 1),
                 ChgRel_Dep = c(NA,
                                res_equi1$out_ts[2, 2] / mes_series1[2, 2] - 1,
                                res_equi1b$out_ts[2, 2] / mes_series1[2, 2] - 1),
                 ChgAbs_Rev = c(NA,
                                res_equi1$out_ts[2, 1] - mes_series1[2, 1],
                                res_equi1b$out_ts[2, 1] - mes_series1[2, 1]),
                 ChgAbs_Dep = c(NA,
                                res_equi1$out_ts[2, 2] - mes_series1[2, 2],
                                res_equi1b$out_ts[2, 2] - mes_series1[2, 2])))


###########
# Exemple 2 : Dans ce deuxième exemple, nous considérons les données simulées des
```

```
#             ventes trimestrielles de véhicules par région (Ouest, Centre et Est),
#            ainsi qu'un total national pour les trois régions, et par type de véhicules
#           (voitures, camions et un total qui peut inclure d'autres types de véhicules).
#            Les données correspondent à des données directement désaisonnalisées qui
#            ont été étalonnées aux totaux annuels des séries originales (non
#            désaisonnalisées) correspondantes dans le cadre du processus de
#            désaisonnalisation (par exemple, avec le « spec » FORCE du logiciel
#            X-13ARIMA-SEATS).
#
#            L'objectif est de réconcilier les ventes régionales avec les ventes
#            nationales sans modifier ces dernières, tout en veillant à ce que la somme
#            des ventes de voitures et de camions ne dépasse pas 95% des ventes de tous
#            les types de véhicules au cours d'un trimestre donné. À titre d'exemple,
#            nous supposons que les ventes de camions dans la région Centre pour le 2e
#            trimestre 2022 ne peuvent pas être modifiées.

# Spécifications du problème
mes_specs2 <- data.frame(

  type = c("EQ", rep(NA, 4),
           "EQ", rep(NA, 4),
           "EQ", rep(NA, 4),
           "LE", rep(NA, 3),
           "LE", rep(NA, 3),
           "LE", rep(NA, 3),
           "alter", rep(NA, 4)),

  col = c(NA, "Ouest_Tous", "Centre_Tous", "Est_Tous", "National_Tous",
          NA, "Ouest_Autos", "Centre_Autos", "Est_Autos", "National_Autos",
          NA, "Ouest_Camions", "Centre_Camions", "Est_Camions", "National_Camions",
          NA, "Ouest_Autos", "Ouest_Camions", "Ouest_Tous",
          NA, "Centre_Autos", "Centre_Camions", "Centre_Tous",
          NA, "Est_Autos", "Est_Camions", "Est_Tous",
          NA, "National_Tous", "National_Autos", "National_Camions", "Centre_Camions"),

  row = c(rep("Total national - Tous les véhicules", 5),
          rep("Total national - Autos", 5),
          rep("Total national - Camions", 5),
          rep("Somme région Ouest", 4),
          rep("Somme région Centre", 4),
          rep("Somme région Est", 4),
          rep("Coefficient d'altérabilité", 5)),

  coef = c(NA, 1, 1, 1, -1,
           NA, 1, 1, 1, -1,
           NA, 1, 1, 1, -1,
           NA, 1, 1, -.95,
           NA, 1, 1, -.95,
           NA, 1, 1, -.95,
           NA, 0, 0, 0, 0),

  time_val = c(rep(NA, 31), 2022.25))
```

```
# Début et fin du « data frame » des spécifications
head(mes_specs2, n = 10)
tail(mes_specs2)

# Données du problème
mes_series2 <- ts(
  matrix(c(43, 49, 47, 136, 20, 18, 12, 53, 20, 22, 26, 61,
           40, 45, 42, 114, 16, 16, 19, 44, 21, 26, 21, 59,
           35, 47, 40, 133, 14, 15, 16, 50, 19, 25, 19, 71,
           44, 44, 45, 138, 19, 20, 14, 52, 21, 18, 27, 74,
           46, 48, 55, 135, 16, 15, 19, 51, 27, 25, 28, 54),
         ncol = 12,
         byrow = TRUE,
         dimnames = list(NULL,
                         c("Ouest_Tous", "Centre_Tous", "Est_Tous",
                           "National_Tous", "Ouest_Autos", "Centre_Autos",
                           "Est_Autos", "National_Autos", "Ouest_Camions",
                           "Centre_Camions", "Est_Camions", "National_Camions"))),
  start = c(2022, 1),
  frequency = 4)

# Réconcilier sans afficher l'en-tête de la fonction et imposer des données non négatives
res_equi2 <- tsbalancing(
  in_ts                    = mes_series2,
  problem_specs_df         = mes_specs2,
  temporal_grp_periodicity = frequency(mes_series2),
  lower_bound              = 0,
  quiet                    = TRUE)

# Données initiales
mes_series2

# Données réconciliées
res_equi2$out_ts

# Vérifier la présence de solutions invalides
any(res_equi2$proc_grp_df$sol_status_val < 0)

# Afficher les écarts maximaux des contraintes en sortie
res_equi2$proc_grp_df[, c("proc_grp_label", "max_discr")]


###########
# Exemple 3 : Reproduire le 2ème exemple de `tsraking_driver()` avec `tsbalancing()`
#             (ratissage à 1 dimension avec préservation des totaux annuels).

# Métadonnées de `tsraking()`
mes_meta3 <- data.frame(series = c("autos_alb", "autos_sask", "autos_man"),
                        total1 = rep("autos_tot", 3))
mes_meta3

# Spécifications du problème de `tsbalancing()`
mes_specs3 <- rkMeta_to_blSpecs(mes_meta3)
```

```
mes_specs3

# Données du problème
mes_series3 <- ts(matrix(c(14, 18, 14, 58,
                           17, 14, 16, 44,
                           14, 19, 18, 58,
                           20, 18, 12, 53,
                           16, 16, 19, 44,
                           14, 15, 16, 50,
                           19, 20, 14, 52,
                           16, 15, 19, 51),
                         ncol = 4,
                         byrow = TRUE,
                         dimnames = list(NULL, c("autos_alb", "autos_sask",
                                                 "autos_man", "autos_tot"))),
                  start = c(2019, 2),
                  frequency = 4)

# Réconcilier les données avec `tsraking()` (via `tsraking_driver()`)
res_ratis3 <- tsraking_driver(in_ts = mes_series3,
                              metadata_df = mes_meta3,
                              temporal_grp_periodicity = frequency(mes_series3),
                              quiet = TRUE)

# Réconcilier les données avec `tsbalancing()`
res_equi3 <- tsbalancing(in_ts = mes_series3,
                         problem_specs_df = mes_specs3,
                         temporal_grp_periodicity = frequency(mes_series3),
                         quiet = TRUE)

# Données initiales
mes_series3

# Les deux ensembles de données réconciliées
res_ratis3
res_equi3$out_ts

# Vérifier la présence de solutions de `tsbalancing()` invalides
any(res_equi3$proc_grp_df$sol_status_val < 0)

# Afficher les écarts maximaux des contraintes en sortie dans les solutions de `tsbalancing()`
res_equi3$proc_grp_df[, c("proc_grp_label", "max_discr")]

# Confirmer que les deux solutions (`tsraking() et `tsbalancing()`) sont les mêmes
all.equal(res_ratis3, res_equi3$out_ts)
```

---

tsDF_to_ts *Fonction réciproque de* ts_to_tsDF()

Description

Convertir un *data frame* (non empilé) de séries chronologiques (format de données de `benchmarking()` et `stock_benchmarking()`) en un objet « ts » (ou « mts »).

Cette fonction est utile pour convertir le *data frame* renvoyé par un appel à `benchmarking()` ou `stock_benchmarking()` en un objet « ts », où une ou plusieurs séries ont été étalonnées en mode de traitement *non groupes-BY*. Les *data frame* empilés de séries chronologiques associées à des exécutions en mode *groupes-BY* doivent d'abord être *désempilés* avec `unstack_tsDF()`.

Usage

```
tsDF_to_ts(
  ts_df,
  frequency,
  yr_cName = "year",
  per_cName = "period"
)
```

Arguments

`ts_df`	(obligatoire) *Data frame*, ou objet compatible, à convertir.
`frequency`	(obligatoire) Entier spécifiant la fréquence de la (des) série(s) à convertir. La fréquence d'une série chronologique correspond au nombre maximum de périodes dans une année (par exemple, 12 pour des données mensuelles, 4 pour des données trimestrielles, 1 pour des données annuelles).
`yr_cName`, `per_cName`	(optionnel) Chaînes de caractères spécifiant le nom des variables (colonnes) numériques dans le *data frame* d'entrée qui contiennent les identificateurs d'année et de période du point de données. **Les valeurs par défaut** sont `yr_cName = "year"` et `per_cName = "period"`.

Value

La fonction renvoie un objet de type série chronologique (« ts » ou « mts »), qui peut être explicitement converti en un autre type d'objet avec la fonction `as*()` appropriée (ex., `tsibble::as_tsibble()` le convertirait en tsibble).

See Also

`ts_to_tsDF()` `unstack_tsDF()` `benchmarking()` `stock_benchmarking()`

Examples

```
# Série chronologique trimestrielle initiale (série indicatrice à étalonner)
sc_tri <- ts(c(1.9, 2.4, 3.1, 2.2, 2.0, 2.6, 3.4, 2.4, 2.3),
             start = c(2015, 1), frequency = 4)
```

```
# Série chronologique annuelle (étalons)
sc_ann <- ts(c(10.3, 10.2), start = 2015, frequency = 1)


# Étalonnage proportionnel
res_eta <- benchmarking(ts_to_tsDF(sc_tri),
                        ts_to_bmkDF(sc_ann, ind_frequency = 4),
                        rho = 0.729, lambda = 1, biasOption = 3,
                        quiet = TRUE)

# Séries chronologiques initiale et finale (étalonnée) - objects « ts »
sc_tri
tsDF_to_ts(res_eta$series, frequency = 4)


# Étalonnage proportionnel de stocks de fin d'année - plusieurs (3) séries
# traitées avec l'argument `by` (en mode groupes-BY)
sc_tri2 <- ts.union(ser1 = sc_tri,     ser2 = sc_tri * 100, ser3 = sc_tri * 10)
sc_ann2 <- ts.union(ser1 = sc_ann / 4, ser2 = sc_ann * 25,  ser3 = sc_ann * 2.5)
res_eta2 <- stock_benchmarking(stack_tsDF(ts_to_tsDF(sc_tri2)),
                               stack_bmkDF(ts_to_bmkDF(
                                 sc_ann2, ind_frequency = 4,
                                 discrete_flag = TRUE, alignment = "e")),
                               rho = 0.729, lambda = 1, biasOption = 3,
                               by = "series",
                               quiet = TRUE)

# Séries chronologiques initiales et finales (étalonnées) - objects « mts »
sc_tri2
tsDF_to_ts(unstack_tsDF(res_eta2$series), frequency = 4)
```

`tsraking` *Rétablir les contraintes d'agrégation transversales (contemporaines)*

Description

Réplication de la procédure TSRAKING de G-Séries 2.0 en SAS® (PROC TSRAKING). Voir la documentation de G-Séries 2.0 pour plus de détails (Statistique Canada 2016).

Cette fonction rétablit les contraintes d'agrégation transversales dans un système de séries chronologiques. Les contraintes d'agrégation peuvent provenir d'une table à 1 ou 2 dimensions. Optionnellement, des contraintes temporelles peuvent également être préservées.

En pratique, `tsraking()` est généralement appelée à travers `tsraking_driver()` afin de réconcilier toutes les périodes du système de séries chronologiques en un seul appel de fonction.

Usage

```
tsraking(
  data_df,
```

```
  metadata_df,
  alterability_df = NULL,
  alterSeries = 1,
  alterTotal1 = 0,
  alterTotal2 = 0,
  alterAnnual = 0,
  tolV = 0.001,
  tolP = NA,
  warnNegResult = TRUE,
  tolN = -0.001,
  id = NULL,
  verbose = FALSE,

  # Nouveau dans G-Séries 3.0
  Vmat_option = 1,
  warnNegInput = TRUE,
  quiet = FALSE
)
```

Arguments

`data_df` (obligatoire)

Data frame, ou objet compatible, qui contient les données des séries chronologiques à réconcilier. Il doit au minimum contenir des variables correspondant aux séries composantes et aux totaux de contrôle transversaux spécifiés dans le *data frame* des métadonnées de ratissage (argument `metadata_df`). Si plus d'un enregistrement (plus d'une période) est fournie, la somme des valeurs des séries composantes fournies sera également préservée à travers des contraintes temporelles implicites.

`metadata_df` (obligatoire)

Data frame, ou objet compatible, qui décrit les contraintes d'agrégation transversales (règles d'additivité) pour le problème de ratissage (« *raking* »). Deux variables de type caractère doivent être incluses dans le *data frame* : `series` et `total1`. Deux variables sont optionnelles : `total2` (caractère) et `alterAnnual` (numérique). Les valeurs de la variable `series` représentent les noms des variables des séries composantes dans le *data frame* des données d'entrée (argument `data_df`). De même, les valeurs des variables `total1` et `total2` représentent les noms des variables des totaux de contrôle transversaux de 1ère et 2ème dimension dans le *data frame* des données d'entrée. La variable `alterAnnual` contient le coefficient d'altérabilité pour la contrainte temporelle associée à chaque série composante. Lorsqu'elle est spécifiée, cette dernière remplace le coefficient d'altérabilité par défaut spécifié avec l'argument `alterAnnual`.

`alterability_df`

(optionnel)

Data frame, ou objet compatible, ou `NULL`, qui contient les variables de coefficients d'altérabilité. Elles doivent correspondre à une série composante ou à un total de contrôle transversal, c'est-à-dire qu'une variable portant le même nom doit exister dans le *data frame* des données d'entrée (argument `data_df`). Les

valeurs de ces coefficients d'altérabilité remplaceront les coefficients d'altérabilité par défaut spécifiés avec les arguments `alterSeries`, `alterTotal1` et `alterTotal2`. Lorsque le *data frame* des données d'entrée contient plusieurs enregistrements et que le *data frame* des coefficients d'altérabilité n'en contient qu'un seul, les coefficients d'altérabilité sont utilisés (répétés) pour tous les enregistrements du *data frame* des données d'entrée. Le *data frame* des coefficients d'altérabilité peut également contenir autant d'enregistrements que le *data frame* des données d'entrée.

La valeur par défaut est `alterability_df = NULL` (coefficients d'altérabilité par défaut).

`alterSeries` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les valeurs des séries composantes. Il s'appliquera aux séries composantes pour lesquelles des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterSeries = 1.0` (valeurs des séries composantes non contraignantes).

`alterTotal1` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les totaux de contrôle transversaux de la 1ère dimension. Il s'appliquera aux totaux de contrôle transversaux pour lesquels des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterTotal1 = 0.0` (totaux de contrôle transversaux de 1ère dimension contraignants).

`alterTotal2` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les totaux de contrôle transversaux de la 2ème dimension. Il s'appliquera aux totaux de contrôle transversaux pour lesquels des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).

La valeur par défaut est `alterTotal2 = 0.0` (totaux de contrôle transversaux de 2ème dimension contraignants).

`alterAnnual` (optionnel)

Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les contraintes temporelles (ex., totaux annuels) des séries composantes. Il s'appliquera aux séries composantes pour lesquelles des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des métadonnées de ratissage (argument `metadata_df`).

La valeur par défaut est `alterAnnual = 0.0` (totaux de contrôle temporels contraignants).

`tolV`, `tolP` (optionnel)

Nombre réel non négatif, ou `NA`, spécifiant la tolérance, en valeur absolue ou en pourcentage, à utiliser lors du test ultime pour les totaux de contrôle contraignants (coefficient d'altérabilité de 0.0 pour les totaux de contrôle temporels ou transversaux). Le test compare les totaux de contrôle contraignants d'entrée

avec ceux calculés à partir des séries composantes réconciliées (en sortie). Les arguments `tolV` et `tolP` ne peuvent pas être spécifiés tous les deux à la fois (l'un doit être spécifié tandis que l'autre doit être `NA`).

Exemple : pour une tolérance de 10 *unités*, spécifiez `tolV = 10, tolP = NA`; pour une tolérance de 1%, spécifiez `tolV = NA, tolP = 0.01`.

Les valeurs par défaut sont `tolV = 0.001` et `tolP = NA`.

`warnNegResult` (optionnel)

Argument logique (*logical*) spécifiant si un message d'avertissement doit être affiché lorsqu'une valeur négative créée par la fonction dans une série réconciliée (en sortie) est inférieure au seuil spécifié avec l'argument `tolN`.

La valeur par défaut est `warnNegResult = TRUE`.

`tolN` (optionnel)

Nombre réel négatif spécifiant le seuil pour l'identification des valeurs négatives. Une valeur est considérée négative lorsqu'elle est inférieure à ce seuil.

La valeur par défaut est `tolN = -0.001`.

`id` (optionnel)

Vecteur de chaînes de caractère (longueur minimale de 1), ou `NULL`, spécifiant le nom des variables additionnelles à transférer du *data frame* d'entrée (argument `data_df`) au *data frame* de sortie, c.-à-d., l'objet renvoyé par la fonction (voir la section **Valeur de retour**). Par défaut, le *data frame* de sortie ne contient que les variables énumérées dans le *data frame* des métadonnées de ratissage (argument `metadata_df`).

La valeur par défaut est `id = NULL`.

`verbose` (optionnel)

Argument logique (*logical*) spécifiant si les informations sur les étapes intermédiaires avec le temps d'exécution (temps réel et non le temps CPU) doivent être affichées. Notez que spécifier l'argument `quiet = TRUE` annulerait l'argument `verbose`.

La valeur par défaut est `verbose = FALSE`.

`Vmat_option` (optionnel)

Spécification de l'option pour les matrices de variance (V_e et V_ϵ; voir la section **Détails**) :

Valeur	Description
1	Utiliser les vecteurs x et g dans les matrices de variance.
2	Utiliser les vecteurs $\|x\|$ et $\|g\|$ dans les matrices de variance.

Voir Ferland (2016) et la sous-section **Arguments** `Vmat_option` **et** `warnNegInput` dans la section **Détails** pour plus d'informations.

La valeur par défaut est `Vmat_option = 1`.

`warnNegInput` (optionnel)

Argument logique (*logical*) spécifiant si un message d'avertissement doit être affiché lorsqu'une valeur négative plus petite que le seuil spécifié par l'argument `tolN` est trouvée dans le *data frame* des données d'entrée (argument `data_df`).

La valeur par défaut est `warnNegInput = TRUE`.

`quiet` (optionnel)

Argument logique (*logical*) spécifiant s'il faut ou non afficher uniquement les informations essentielles telles que les messages d'avertissements et d'erreurs. Spécifier `quiet = TRUE` annulera également l'argument `verbose` et est équivalent à *envelopper* votre appel à `tsraking()` avec `suppressMessages()`.

La valeur par défaut est `quiet = FALSE`.

Details

Cette fonction renvoie la solution des moindres carrés généralisés d'une variante spécifique, simple du modèle général de ratissage (*raking*) basé sur la régression proposé par Dagum et Cholette (Dagum et Cholette 2006). Le modèle, sous forme matricielle, est le suivant :

$$\begin{bmatrix} x \\ g \end{bmatrix} = \begin{bmatrix} I \\ G \end{bmatrix} \theta + \begin{bmatrix} e \\ \varepsilon \end{bmatrix}$$

où

- x est le vecteur des valeurs initiales des séries composantes.
- θ est le vecteur des valeurs finales (réconciliées) des séries composantes.
- $e \sim (0, V_e)$ est le vecteur des erreurs de mesure de x avec la matrice de covariance $V_e = \text{diag}(c_x x)$, ou $V_e = \text{diag}(|c_x x|)$ quand l'argument `Vmat_option = 2`, où c_x est le vecteur des coefficients d'alterabilité de x.
- g est le vecteur des totaux de contrôle initiaux, incluant les totaux temporels des séries composantes (le cas échéant).
- $\varepsilon \sim (0, V_\varepsilon)$ est le vecteur des erreurs de mesure de g avec la matrice de covariance $V_\varepsilon = \text{diag}(c_g g)$, ou $V_\varepsilon = \text{diag}(|c_g g|)$ quand l'argument `Vmat_option = 2`, où c_g est le vecteur des coefficients d'alterabilité de g.
- G est la matrice des contraintes d'agrégation, y compris les contraintes temporelles implicites (le cas échéant).

La solution généralisée des moindres carrés est :

$$\hat{\theta} = x + V_e G^{\mathrm{T}} \left(G V_e G^{\mathrm{T}} + V_\varepsilon \right)^{+} (g - Gx)$$

où A^{+} désigne l'inverse de Moore-Penrose de la matrice A.

`tsraking()` résout un seul problème de ratissage à la fois, c'est-à-dire, soit une seule période du système de séries chronologiques, ou un seul groupe temporel (ex., toutes les périodes d'une année donnée) lorsque la préservation des totaux temporels est requise. Plusieurs appels à `tsraking()` sont donc nécessaires pour réconcilier toutes les périodes du système de séries chronologiques. `tsraking_driver()` peut réaliser cela en un seul appel : il détermine commodément l'ensemble des problèmes à résoudre et génère à l'interne les appels individuels à `tsraking()`.

Coefficients d'altérabilité:

Les coefficients d'altérabilité c_x et c_g représentent conceptuellement les erreurs de mesure associées aux valeurs d'entrée des séries composantes x et des totaux de contrôle g respectivement. Il s'agit de nombres réels non négatifs qui, en pratique, spécifient l'ampleur de la modification permise d'une valeur initiale par rapport aux autres valeurs. Un coefficients d'altérabilité de 0.0

définit une valeur fixe (contraignante), tandis qu'un coefficient d'altérabilité supérieur à 0.0 définit une valeur libre (non contraignante). L'augmentation du coefficient d'altérabilité d'une valeur initiale entraîne davantage de changements pour cette valeur dans les données réconciliées (en sortie) et, inversement, moins de changements lorsque l'on diminue le coefficient d'altérabilité. Les coefficients d'altérabilité par défaut sont 1.0 pour les valeurs des séries composantes et 0.0 pour les totaux de contrôle transversaux et, le cas échéant, les totaux temporels des séries composantes. Ces coefficients d'altérabilité par défaut entraînent une répartition proportionnelle des écarts entre les séries composantes. En fixant les coefficients d'altérabilité des séries composantes à l'inverse des valeurs initiales des séries composantes, on obtiendrait une répartition uniforme des écarts à la place. Des totaux *presque contraignants* peuvent être obtenus en pratique en spécifiant des coefficients d'altérabilité très petits (presque 0.0) par rapport à ceux des séries composantes (non contraignantes).

La préservation des totaux temporels fait référence au fait que les totaux temporels, le cas échéant, sont généralement conservés « aussi près que possible » de leur valeur initiale. Une *préservation pure* est obtenue par défaut avec des totaux temporels contraignants, tandis que le changement est minimisé avec des totaux temporels non contraignants (conformément à l'ensemble de coefficients d'altérabilité utilisés).

Arguments `Vmat_option` et `warnNegInput`:

Ces arguments permettent une gestion alternative des valeurs négatives dans les données d'entrée, similaire à celle de `tsbalancing()`. Leurs valeurs par défaut correspondent au comportement de G-Séries 2.0 (PROC TSRAKING en SAS®) pour lequel des options équivalentes ne sont pas définies. Ce dernier a été développé en présumant des « données d'entrée non négatives uniquement », à l'instar de PROC BENCHMARKING dans G-Séries 2.0 en SAS® qui n'autorisait pas non plus les valeurs négatives avec l'étalonnage proportionnel, ce qui explique l'avertissement « suspicious use of proportional raking » (*utilisation suspecte du ratissage proportionnel*) en présence de valeurs négatives avec PROC TSRAKING dans G-Series 2.0 et lorsque `warnNegInput = TRUE` (par défault). Cependant, le ratissage (proportionnel) en présence de valeurs négatives fonctionne généralement bien avec `Vmat_option = 2` et produit des solutions raisonnables et intuitives. Par exemple, alors que l'option par défaut `Vmat_option = 1` échoue à résoudre la contrainte `A + B = C` avec les données d'entrée `A = 2`, `B = -2`, `C = 1` et les coefficients d'altérabilité par défaut, `Vmat_option = 2` renvoie la solution (intuitive) `A = 2.5`, `B = -1.5`, `C = 1` (augmentation de 25% pour `A` et `B`). Voir Ferland (2016) pour plus de détails.

Traitement des valeurs manquantes (`NA`):

Une valeur manquante dans le *data frame* des données d'entrée (argument `data_df`) ou dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`) pour n'importe quelle donnée du problème de ratissage (variables énumérées dans le *data frame* des métadonnées avec l'argument `metadata_df`) générera un message d'erreur et arrêtera l'exécution de la fonction.

Value

La fonction renvoie un *data frame* contenant les séries composantes réconciliées, les totaux de contrôle transversaux réconciliés et les variables spécifiées avec l'argument `id`. Notez que l'objet « data.frame » peut être explicitement converti en un autre type d'objet avec la fonction `as*()` appropriée (ex., `tibble::as_tibble()` le convertirait en tibble).

Comparaison de `tsraking()` et `tsbalancing()`

- `tsraking()` est limitée aux problèmes de ratissage (« *raking* ») de tables d'agrégation unidimensionnelles et bidimensionnelles (avec préservation des totaux temporels si nécessaire) alors que `tsbalancing()` traite des problèmes d'équilibrage plus généraux (ex., des problèmes de ratissage de plus grande dimension, solutions non négatives, contraintes linéaires générales d'égalité et d'inégalité par opposition à des règles d'agrégation uniquement, etc.)
- `tsraking()` renvoie la solution des moindres carrés généralisés du modèle de ratissage basé sur la régression de Dagum et Cholette (Dagum et Cholette 2006) tandis que `tsbalancing()` résout le problème de minimisation quadratique correspondant à l'aide d'un solveur numérique. Dans la plupart des cas, la *convergence vers le minimum* est atteinte et la solution de `tsbalancing()` correspond à la solution (exacte) des moindres carrés de `tsraking()`. Cela peut ne pas être le cas, cependant, si la convergence n'a pas pu être atteinte après un nombre raisonnable d'itérations. Cela dit, ce n'est qu'en de très rares occasions que la solution de `tsbalancing()` différera *significativement* de celle de `tsraking()`.
- `tsbalancing()` est généralement plus rapide que `tsraking()`, en particulier pour les gros problèmes de ratissage, mais est généralement plus sensible à la présence de (petites) incohérences dans les données d'entrée associées aux contraintes redondantes des problèmes de ratissage *entièrement spécifiés* (ou surspécifiés). `tsraking()` gère ces incohérences en utilisant l'inverse de Moore-Penrose (distribution uniforme à travers tous les totaux contraignants).
- `tsbalancing()` permet de spécifier des problèmes épars (clairsemés) sous leur forme réduite. Ce n'est pas le cas de `tsraking()` où les règles d'agrégation doivent toujours être entièrement spécifiées étant donné qu'un *cube de données complet*, sans données manquantes, est attendu en entrée (chaque série composante de l'*intérieur du cube* doit contribuer à toutes les dimensions du cube, c.-à-d., à chaque série totale des *faces extérieures du cube*).
- Les deux outils traitent différemment les valeurs négatives dans les données d'entrée par défaut. Alors que les solutions des problèmes de ratissage obtenues avec `tsbalancing()` et `tsraking()` sont identiques lorsque tous les points de données d'entrée sont positifs, elles seront différentes si certains points de données sont négatifs (à moins que l'argument `Vmat_option = 2` ne soit spécifié avec `tsraking()`).
- Alors que `tsbalancing()` et `tsraking()` permettent toutes les deux de préserver les totaux temporels, la gestion du temps n'est pas incorporée dans `tsraking()`. Par exemple, la construction des groupes de traitement (ensembles de périodes de chaque problème de ratissage) est laissée à l'utilisateur avec `tsraking()` et des appels séparés doivent être soumis pour chaque groupe de traitement (chaque problème de ratissage). De là l'utilité de la fonction d'assistance `tsraking_driver()` pour `tsraking()`.
- `tsbalancing()` renvoie le même ensemble de séries que l'objet d'entrée de type série chronologique (argument `in_ts`) alors que `tsraking()` renvoie l'ensemble des séries impliquées dans le problème de ratissage plus celles spécifiées avec l'argument `id` (qui pourrait correspondre à un sous-ensemble des séries d'entrée).

References

Bérubé, J. and S. Fortier (2009). « PROC TSRAKING: An in-house SAS® procedure for balancing time series ». Dans **JSM Proceedings, Business and Economic Statistics Section**. Alexandria, VA: American Statistical Association.

Dagum, E. B. and P. Cholette (2006). **Benchmarking, Temporal Distribution and Reconciliation Methods of Time Series**. Springer-Verlag, New York, Lecture Notes in Statistics, Vol. 186.

Ferland, M. (2016). « Negative Values with PROC TSRAKING ». **Document interne**. Statistique Canada, Ottawa, Canada.

Fortier, S. and B. Quenneville (2009). « Reconciliation and Balancing of Accounts and Time Series ». Dans **JSM Proceedings, Business and Economic Statistics Section**. Alexandria, VA: American Statistical Association.

Quenneville, B. and S. Fortier (2012). « Restoring Accounting Constraints in Time Series – Methods and Software for a Statistical Agency ». **Economic Time Series: Modeling and Seasonality**. Chapman & Hall, New York.

Statistique Canada (2016). « La procédure TSRAKING ». **Guide de l'utilisateur de G-Séries 2.0**. Statistique Canada, Ottawa, Canada.

Statistique Canada (2018). **Théorie et application de la réconciliation (Code du cours 0437)**. Statistique Canada, Ottawa, Canada.

See Also

tsraking_driver() tsbalancing() rkMeta_to_blSpecs() gs.gInv_MP() build_raking_problem() aliases

Examples

```
############
# Exemple 1 : Problème simple de ratissage à une dimension dans lequel les valeurs des
#             `autos` et des `camions` doivent être égales à la valeur du `total`.

# Métadonnées du problème
mes_meta1 <- data.frame(series = c("autos", "camions"),
                        total1 = c("total", "total"))
mes_meta1

# Données du problème
mes_series1 <- data.frame(autos = 25, camions = 5, total = 40)

# Réconcilier les données
res_ratis1 <- tsraking(mes_series1, mes_meta1)

# Données initiales
mes_series1

# Données réconciliées
res_ratis1

# Vérifier les contraintes transversales en sortie
all.equal(rowSums(res_ratis1[c("autos", "camions")]), res_ratis1$total)

# Vérifier le total de contrôle (fixe)
all.equal(mes_series1$total, res_ratis1$total)
```

```
############
# Exemple 2 : problème de ratissage à 2 dimensions similaire au 1er exemple mais
#             en ajoutant les ventes régionales pour les 3 provinces des prairies
#             (Alb., Sask. et Man.) et où les ventes de camions en Sask. ne sont
#             pas modifiables (coefficient d'altérabilité = 0), avec `quiet = TRUE`
#             pour éviter l'affichage de l'en-tête de la fonction.

# Métadonnées du problème
mes_meta2 <- data.frame(series = c("autos_alb", "autos_sask", "autos_man",
                                   "camions_alb", "camions_sask", "camions_man"),
                        total1 = c(rep("total_autos", 3),
                                   rep("total_camions", 3)),
                        total2 = rep(c("total_alb", "total_sask", "total_man"), 2))

# Données du problème
mes_series2 <- data.frame(autos_alb = 12, autos_sask = 14, autos_man = 13,
                          camions_alb = 20, camions_sask = 20, camions_man = 24,
                          total_alb = 30, total_sask = 31, total_man = 32,
                          total_autos = 40, total_camions = 53)

# Réconcilier les données
res_ratis2 <- tsraking(mes_series2, mes_meta2,
                       alterability_df = data.frame(camions_sask = 0),
                       quiet = TRUE)

# Données initiales
mes_series2

# Données réconciliées
res_ratis2

# Vérifier les contraintes transversales en sortie
all.equal(rowSums(res_ratis2[c("autos_alb", "autos_sask", "autos_man")]), res_ratis2$total_autos)
all.equal(rowSums(res_ratis2[c("camions_alb", "camions_sask", "camions_man")]), res_ratis2$total_camions)
all.equal(rowSums(res_ratis2[c("autos_alb", "camions_alb")]), res_ratis2$total_alb)
all.equal(rowSums(res_ratis2[c("autos_sask", "camions_sask")]), res_ratis2$total_sask)
all.equal(rowSums(res_ratis2[c("autos_man", "camions_man")]), res_ratis2$total_man)

# Vérifier le total de contrôle (fixe)
cols_tot <- union(unique(mes_meta2$total1), unique(mes_meta2$total2))
all.equal(mes_series2[cols_tot], res_ratis2[cols_tot])

# Vérifier la valeur des camions en Saskatchewan (fixée à 20)
all.equal(mes_series2$camions_sask, res_ratis2$camions_sask)
```

---

tsraking_driver *Fonction d'assistance pour* tsraking()

Description

Fonction d'assistance pour `tsraking()` qui détermine de manière pratique l'ensemble des problèmes de ratissage (« *raking* ») à résoudre et génère à l'interne les appels individuels à `tsraking()`. Cette fonction est particulièrement utile dans le contexte de la préservation des totaux temporels (ex., totaux annuels) où chaque problème de ratissage individuel implique une seule période pour les groupes temporels incomplets (ex., années incomplètes) ou plusieurs périodes pour les groupes temporels complets (ex., l'ensemble des périodes d'une année complète).

Usage

```
tsraking_driver(
  in_ts,
  ...,  # arguments de `tsraking()` excluant `data_df`
  temporal_grp_periodicity = 1,
  temporal_grp_start = 1
)
```

Arguments

`in_ts` (obligatoire)

Objet de type série chronologique (« ts » ou « mts »), ou objet compatible, qui contient les données des séries chronologiques à réconcilier. Il s'agit des données d'entrée (solutions initiales) des problèmes de ratissage (« *raking* »).

`...` Arguments transmis à `tsraking`

`metadata_df` (obligatoire)

Data frame, ou objet compatible, qui décrit les contraintes d'agrégation transversales (règles d'additivité) pour le problème de ratissage (« *raking* »). Deux variables de type caractère doivent être incluses dans le *data frame* : `series` et `total1`. Deux variables sont optionnelles : `total2` (caractère) et `alterAnnual` (numérique). Les valeurs de la variable `series` représentent les noms des variables des séries composantes dans le *data frame* des données d'entrée (argument `data_df`). De même, les valeurs des variables `total1` et `total2` représentent les noms des variables des totaux de contrôle transversaux de 1ère et 2ème dimension dans le *data frame* des données d'entrée. La variable `alterAnnual` contient le coefficient d'altérabilité pour la contrainte temporelle associée à chaque série composante. Lorsqu'elle est spécifiée, cette dernière remplace le coefficient d'altérabilité par défaut spécifié avec l'argument `alterAnnual`.

`alterability_df` (optionnel)

Data frame, ou objet compatible, ou `NULL`, qui contient les variables de coefficients d'altérabilité. Elles doivent correspondre à une série composante ou à un total de contrôle transversal, c'est-à-dire qu'une variable portant le même nom doit exister dans le *data frame* des données d'entrée (argument `data_df`). Les valeurs de ces coefficients d'altérabilité remplaceront les coefficients d'altérabilité par défaut spécifiés avec les arguments `alterSeries`, `alterTotal1` et `alterTotal2`. Lorsque le *data frame* des données d'entrée contient plusieurs enregistrements et que le *data frame* des coefficients

d'altérabilité n'en contient qu'un seul, les coefficients d'altérabilité sont utilisés (répétés) pour tous les enregistrements du *data frame* des données d'entrée. Le *data frame* des coefficients d'altérabilité peut également contenir autant d'enregistrements que le *data frame* des données d'entrée.
La valeur par défaut est `alterability_df = NULL` (coefficients d'altérabilité par défaut).

`alterSeries` (optionnel)
Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les valeurs des séries composantes. Il s'appliquera aux séries composantes pour lesquelles des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).
La valeur par défaut est `alterSeries = 1.0` (valeurs des séries composantes non contraignantes).

`alterTotal1` (optionnel)
Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les totaux de contrôle transversaux de la 1ère dimension. Il s'appliquera aux totaux de contrôle transversaux pour lesquels des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).
La valeur par défaut est `alterTotal1 = 0.0` (totaux de contrôle transversaux de 1ère dimension contraignants).

`alterTotal2` (optionnel)
Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les totaux de contrôle transversaux de la 2ème dimension. Il s'appliquera aux totaux de contrôle transversaux pour lesquels des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des coefficients d'altérabilité (argument `alterability_df`).
La valeur par défaut est `alterTotal2 = 0.0` (totaux de contrôle transversaux de 2ème dimension contraignants).

`alterAnnual` (optionnel)
Nombre réel non négatif spécifiant le coefficient d'altérabilité par défaut pour les contraintes temporelles (ex., totaux annuels) des séries composantes. Il s'appliquera aux séries composantes pour lesquelles des coefficients d'altérabilité n'ont pas déjà été spécifiés dans le *data frame* des métadonnées de ratissage (argument `metadata_df`).
La valeur par défaut est `alterAnnual = 0.0` (totaux de contrôle temporels contraignants).

`tolV,tolP` (optionnel)
Nombre réel non négatif, ou `NA`, spécifiant la tolérance, en valeur absolue ou en pourcentage, à utiliser lors du test ultime pour les totaux de contrôle contraignants (coefficient d'altérabilité de 0.0 pour les totaux de contrôle temporels ou transversaux). Le test compare les totaux de contrôle contraignants d'entrée avec ceux calculés à partir des séries composantes réconciliées (en sortie). Les arguments `tolV` et `tolP` ne peuvent pas être spécifiés tous les deux à la fois (l'un doit être spécifié tandis que l'autre doit être `NA`).
Exemple : pour une tolérance de 10 *unités*, spécifiez `tolV = 10, tolP = NA`; pour une tolérance de 1%, spécifiez `tolV = NA, tolP = 0.01`.

Les valeurs par défaut sont `tolV = 0.001` et `tolP = NA`.

`warnNegResult` (optionnel)
: Argument logique (*logical*) spécifiant si un message d'avertissement doit être affiché lorsqu'une valeur négative créée par la fonction dans une série réconciliée (en sortie) est inférieure au seuil spécifié avec l'argument `tolN`.
La valeur par défaut est `warnNegResult = TRUE`.

`tolN` (optionnel)
: Nombre réel négatif spécifiant le seuil pour l'identification des valeurs négatives. Une valeur est considérée négative lorsqu'elle est inférieure à ce seuil.
La valeur par défaut est `tolN = -0.001`.

`id` (optionnel)
: Vecteur de chaînes de caractère (longueur minimale de 1), ou `NULL`, spécifiant le nom des variables additionnelles à transférer du *data frame* d'entrée (argument `data_df`) au *data frame* de sortie, c.-à-d., l'objet renvoyé par la fonction (voir la section **Valeur de retour**). Par défaut, le *data frame* de sortie ne contient que les variables énumérées dans le *data frame* des métadonnées de ratissage (argument `metadata_df`).
La valeur par défaut est `id = NULL`.

`verbose` (optionnel)
: Argument logique (*logical*) spécifiant si les informations sur les étapes intermédiaires avec le temps d'exécution (temps réel et non le temps CPU) doivent être affichées. Notez que spécifier l'argument `quiet = TRUE` annulerait l'argument `verbose`.
La valeur par défaut est `verbose = FALSE`.

`Vmat_option` (optionnel)
: Spécification de l'option pour les matrices de variance (V_e et V_ϵ; voir la section **Détails**) :

Valeur	Description
1	Utiliser les vecteurs x et g dans les matrices de variance.
2	Utiliser les vecteurs $\|x\|$ et $\|g\|$ dans les matrices de variance.

Voir Ferland (2016) et la sous-section **Arguments** `Vmat_option` **et** `warnNegInput` dans la section **Détails** pour plus d'informations.
La valeur par défaut est `Vmat_option = 1`.

`warnNegInput` (optionnel)
: Argument logique (*logical*) spécifiant si un message d'avertissement doit être affiché lorsqu'une valeur négative plus petite que le seuil spécifié par l'argument `tolN` est trouvée dans le *data frame* des données d'entrée (argument `data_df`).
La valeur par défaut est `warnNegInput = TRUE`.

`quiet` (optionnel)
: Argument logique (*logical*) spécifiant s'il faut ou non afficher uniquement les informations essentielles telles que les messages d'avertissements et d'erreurs. Spécifier `quiet = TRUE` annulera également l'argument `verbose` et est équivalent à *envelopper* votre appel à `tsraking()` avec `suppressMessages()`.
La valeur par défaut est `quiet = FALSE`.

`temporal_grp_periodicity`
: (optionnel)

Nombre entier positif définissant le nombre de périodes dans les groupes temporels pour lesquels les totaux doivent être préservés. Par exemple, spécifiez `temporal_grp_periodicity = 3` avec des séries chronologiques mensuelles pour la préservation des totaux trimestriels et `temporal_grp_periodicity = 12` (ou `temporal_grp_periodicity = frequency(in_ts)`) pour la préservation des totaux annuels. Spécifier `temporal_grp_periodicity = 1` (*défaut*) correspond à un traitement période par période sans préservation des totaux temporels.

La valeur par défaut est `temporal_grp_periodicity = 1` (traitement période par période sans préservation des totaux temporels).

`temporal_grp_start`
: (optionnel)

Entier dans l'intervalle [1 .. `temporal_grp_periodicity`] spécifiant la période (cycle) de départ pour la préservation des totaux temporels. Par exemple, des totaux annuels correspondant aux années financières définies d'avril à mars de l'année suivante seraient spécifiés avec `temporal_grp_start = 4` pour des séries chronologiques mensuelles (`frequency(in_ts) = 12`) et `temporal_grp_start = 2` pour des séries chronologiques trimestrielles (`frequency(in_ts) = 4`). Cet argument n'a pas d'effet pour un traitement période par période sans préservation des totaux temporels (`temporal_grp_periodicity = 1`).

La valeur par défaut est `temporal_grp_start = 1`.

Details

Cette fonction résout un problème de ratissage avec `tsraking()` par groupe de traitement (voir la section **Groupes de traitement** pour plus de détails). L'expression mathématique de ces problèmes de ratissage peut être trouvée dans la section **Détails** de la documentation de `tsraking()`.

Le *data frame* des coefficients d'altérabilité (argument `alterability_df`) spécifié avec `tsraking_driver()` peut soit contenir :

- Un seul enregistrement : les coefficients spécifiés seront utilisés pour toutes les périodes de l'objet d'entrée de type série chronologique (argument `in_ts`).
- Un nombre d'enregistrements égal à `frequency(in_ts)` : les coefficients spécifiés seront utilisés pour le *cycle* correspondant aux périodes de l'objet d'entrée de type série chronologique (argument `in_ts`). Exemple pour des données mensuelles : 1er enregistrement pour janvier, 2ème enregistrement pour février, etc.)
- Un nombre d'enregistrements égal à `nrow(in_ts)` : les coefficients spécifiés seront utilisés pour les périodes correspondantes de l'objet d'entrée de type série chronologique (argument `in_ts`), c.-à-d., 1er enregistrement pour la 1ère période, 2ème enregistrement pour la 2ème période, etc.

Spécifier `quiet = TRUE` supprimera les messages de `tsraking()` (ex., l'en-tête de la fonction) et n'affichera que les informations essentielles telles que les avertissements, les erreurs et la période (ou l'ensemble des périodes) en cours de traitement. Nous déconseillons d'*envelopper* l'appel à la fonction `tsraking_driver()` avec `suppressMessages()` pour supprimer l'affichage des informations relatives à la (aux) période(s) en cours de traitement, car cela rendrait difficile le dépannage de problèmes de ratissage individuels.

Bien que tsraking() puisse être appelée avec *apply() pour réconcilier successivement toutes les périodes de l'objet d'entrée de type série chronologique (argument in_ts), l'utilisation de tsraking_driver() présente quelques avantages, notamment :

- la préservation des totaux temporels (seul un traitement période par période, sans préservation des totaux temporels, serait possible avec *apply());
- une plus grande flexibilité dans la spécification des coefficients d'altérabilité définis par l'utilisateur (ex., des valeurs spécifiques aux périodes);
- affichage de la période en cours de traitement dans la console, ce qui est utile pour dépanner les problèmes de ratissage individuels;
- amélioration de la gestion des erreurs, c.-à-d., une meilleure gestion des avertissements ou des erreurs s'ils ne se produisent que pour certains problèmes de ratissage (périodes);
- renvoi automatique d'un objet de type « ts » (« mts »).

Value

La fonction renvoie un objet de type série chronologique (« ts » ou « mts ») contenant les séries composantes réconciliées, les totaux de contrôle transversaux réconciliés et d'autres séries spécifiées avec l'argument id de tsraking(). Il peut être explicitement converti en un autre type d'objet avec la fonction as*() appropriée (ex., tsibble::as_tsibble() le convertirait en tsibble).

Notez qu'un objet NULL est renvoyé si une erreur survient avant que le traitement des données ne puisse commencer. Dans le cas contraire, si l'exécution est suffisamment avancée pour que le traitement des données puisse commencer, alors un objet incomplet (avec des valeurs NA) sera renvoyé en cas d'erreur.

Groupes de traitement

L'ensemble des périodes d'un problème de réconciliation (ratissage ou équilibrage) donné est appelé *groupe de traitement* et correspond soit :

- à une **période unique** lors d'un traitement période par période ou, lorsque les totaux temporels sont préservés, pour les périodes individuelles d'un groupe temporel incomplet (ex., une année incomplète)
- ou à l'**ensemble des périodes d'un groupe temporel complet** (ex., une année complète) lorsque les totaux temporels sont préservés.

Le nombre total de groupes de traitement (nombre total de problèmes de réconciliation) dépend de l'ensemble de périodes des séries chronologiques d'entrée (objet de type série chronologique spécifié avec l'argument in_ts) et de la valeur des arguments temporal_grp_periodicity et temporal_grp_start.

Les scénarios courants incluent temporal_grp_periodicity = 1 (par défaut) pour un traitement période par période sans préservation des totaux temporels et temporal_grp_periodicity = frequency(in_ts) pour la préservation des totaux annuels (années civiles par défaut). L'argument temporal_grp_start permet de spécifier d'autres types d'années (*non civile*). Par exemple, des années financières commençant en avril correspondent à temporal_grp_start = 4 avec des données mensuelles et à temporal_grp_start = 2 avec des données trimestrielles. La préservation des totaux trimestriels avec des données mensuelles correspondrait à temporal_grp_periodicity = 3.

Par défaut, les groupes temporels convrant plus d'une année (c.-à-d., correspondant à `temporal_grp_periodicity > frequency(in_ts)`) débutent avec une année qui est un multiple de `ceiling(temporal_grp_periodicity / frequency(in_ts))`. Par exemple, les groupes bisannuels correspondant à `temporal_grp_periodicity = 2 * frequency(in_ts)` débutent avec une *année paire* par défaut. Ce comportement peut être modifié avec l'argument `temporal_grp_start`. Par exemple, la préservation des totaux bisannuels débutant avec une *année impaire* au lieu d'une *année paire* (par défaut) correspond à `temporal_grp_start = frequency(in_ts) + 1` (avec `temporal_grp_periodicity = 2 * frequency(in_ts)`).

Voir les **Exemples** de `gs.build_proc_grps()` pour des scénarios courants de groupes de traitements.

References

Statistique Canada (2018). "Chapitre : Sujets avancés", **Théorie et application de la réconciliation (Code du cours 0437)**, Statistique Canada, Ottawa, Canada.

See Also

`tsraking()` `tsbalancing()` `rkMeta_to_blSpecs()` `gs.build_proc_grps()`

Examples

```
# Problème de ratissage à 1 dimension où les ventes trimestrielles de voitures
# dans les 3 provinces des Prairies (Alb., Sask. et Man.) pour 8 trimestres,
# de 2019 T2 à 2021 T1, doivent être égales au total (`cars_tot`).

# Métadonnées du problème
mes_meta <- data.frame(series = c("autos_alb", "autos_sask", "autos_man"),
                       total1 = rep("autos_tot", 3))
mes_meta

# Données du problème
mes_series <- ts(matrix(c(14, 18, 14, 58,
                          17, 14, 16, 44,
                          14, 19, 18, 58,
                          20, 18, 12, 53,
                          16, 16, 19, 44,
                          14, 15, 16, 50,
                          19, 20, 14, 52,
                          16, 15, 19, 51),
                        ncol = 4,
                        byrow = TRUE,
                        dimnames = list(NULL, c("autos_alb", "autos_sask",
                                                "autos_man", "autos_tot"))),
                 start = c(2019, 2),
                 frequency = 4)


############
# Exemple 1 : Traitement période-par-période sans préservation des totaux annuels.
```

```
# Réconcilier les données
res_ratis1 <- tsraking_driver(mes_series, mes_meta)

# Données initiales
mes_series

# Données réconciliées
res_ratis1

# Vérifier les contraintes transversales en sortie
all.equal(rowSums(res_ratis1[, mes_meta$series]), as.vector(res_ratis1[, "autos_tot"]))

# Vérifier le total de contrôle (fixe)
all.equal(mes_series[, "autos_tot"], res_ratis1[, "autos_tot"])


###########
# Exemple 2 : Préservation des totaux annuels de 2020 (traitement période-par-période
#             pour les années incomplètes 2019 et 2021), avec `quiet = TRUE` pour
#            éviter d'afficher l'en-tête de la fonction pour chaque groupe de traitement.

# Vérifions tout d'abord que le total annuel de 2020 de la série totale (`autos_tot`)
# et de la somme des séries composantes (`autos_alb`, `autos_sask` et `autos_man`)
# concordent. Dans le cas contraire, il faudrait d'abord résoudre cet écart avant
# d'exécuter `tsraking_driver()`.
tot2020 <- aggregate.ts(window(mes_series, start = c(2020, 1), end = c(2020, 4)))
all.equal(as.numeric(tot2020[, "autos_tot"]), sum(tot2020[, mes_meta$series]))

# Réconcilier les données
res_ratis2 <- tsraking_driver(in_ts = mes_series,
                              metadata_df = mes_meta,
                              quiet = TRUE,
                              temporal_grp_periodicity = frequency(mes_series))

# Données initiales
mes_series

# Données réconciliées
res_ratis2

# Vérifier les contraintes transversales en sortie
all.equal(rowSums(res_ratis2[, mes_meta$series]), as.vector(res_ratis2[, "autos_tot"]))

# Vérifier les contraintes temporelles en sortie (total annuel de 2020 pour chaque série)
all.equal(tot2020,
          aggregate.ts(window(res_ratis2, start = c(2020, 1), end = c(2020, 4))))

# Vérifier le total de contrôle (fixe)
all.equal(mes_series[, "autos_tot"], res_ratis2[, "autos_tot"])


###########
# Exemple 3 : Préservation des totaux annuels pour les années financières allant
```

```
#             d'avril à mars (2019T2-2020T1 et 2020T2-2021T1).

# Calculer les deux totaux d'années financières (objet « ts » annuel)
tot_annFisc <- ts(rbind(aggregate.ts(window(mes_series,
                                            start = c(2019, 2),
                                            end = c(2020, 1))),
                        aggregate.ts(window(mes_series,
                                            start = c(2020, 2),
                                            end = c(2021, 1)))),
                  start = 2019,
                  frequency = 1)

# Écarts dans les totaux d'années financières (série totale contre la somme des
# séries composantes)
as.numeric(tot_annFisc[, "autos_tot"]) - rowSums(tot_annFisc[, mes_meta$series])


# 3a) Réconcilier les totaux d'années financières (ratisser les totaux d'années
#     financières des séries composantes à ceux de la série totale).
tot_annFisc_ratis <- tsraking_driver(in_ts = tot_annFisc,
                                     metadata_df = mes_meta,
                                     quiet = TRUE)

# Confirmer que les écarts précédents ont disparu (ils sont tous les deux nuls).
as.numeric(tot_annFisc_ratis[, "autos_tot"]) - rowSums(tot_annFisc_ratis[, mes_meta$series])

# 3b) Étalonner les séries composantes trimestrielles à ces nouveaux totaux (cohérents)
#     d'années financières.
res_eta <- benchmarking(series_df = ts_to_tsDF(mes_series[, mes_meta$series]),
                        benchmarks_df = ts_to_bmkDF(
                          tot_annFisc_ratis[, mes_meta$series],
                          ind_frequency = frequency(mes_series),

                          # Années financières d'avril à mars (T2 à T1)
                          bmk_interval_start = 2),

                        rho = 0.729,
                        lambda = 1,
                        biasOption = 2,
                        allCols = TRUE,
                        quiet = TRUE)
mes_series_eta <- tsDF_to_ts(cbind(res_eta$series, autos_tot = mes_series[, "autos_tot"]),
                             frequency = frequency(mes_series))

# 3c) Réconcilier les données trimestrielles en préservant les totaux d'années finiacières.
res_ratis3 <- tsraking_driver(in_ts = mes_series_eta,
                              metadata_df = mes_meta,
                              temporal_grp_periodicity = frequency(mes_series),

                              # Années financières d'avril à mars (T2 à T1)
                              temporal_grp_start = 2,

                              quiet = TRUE)
```

```
# Données initiales
mes_series

# Avec totaux d'années finiacières cohérents
mes_series_eta

# Données réconciliées
res_ratis3

# Vérifier les contraintes transversales en sortie
all.equal(rowSums(res_ratis3[, mes_meta$series]), as.vector(res_ratis3[, "autos_tot"]))

# Vérifier les contraintes temporelles en sortie (totaux des deux années financières pour
# chaque série)
all.equal(rbind(aggregate.ts(window(mes_series_eta, start = c(2019, 2), end = c(2020, 1))),
             aggregate.ts(window(mes_series_eta, start = c(2020, 2), end = c(2021, 1)))),
          rbind(aggregate.ts(window(res_ratis3, start = c(2019, 2), end = c(2020, 1))),
                aggregate.ts(window(res_ratis3, start = c(2020, 2), end = c(2021, 1)))))

# Vérifier le total de contrôle (fixe)
all.equal(mes_series[, "autos_tot"], res_ratis3[, "autos_tot"])
```

`ts_to_bmkDF` *Convertir un objet « ts » en* data frame *d'étalons*

Description

Convertir un objet « ts » (ou « mts ») en un *data frame* d'étalons pour les fonctions d'étalonnage avec cinq variables (colonnes) ou plus :

- quatre (4) pour la converture de l'étalon
- une (1) pour chaque série chronologique d'étalons

Pour des étalons discrets (points d'ancrage couvrant une seule période de la série indicatrice, par exemple, des stocks de fin d'année), spécifiez `discrete_flag = TRUE` et `alignment = "b"`, `"e"` ou `"m"`.

Usage

```
ts_to_bmkDF(
  in_ts,
  ind_frequency,
  discrete_flag = FALSE,
  alignment = "b",
  bmk_interval_start = 1,
  startYr_cName = "startYear",
  startPer_cName = "startPeriod",
  endYr_cName = "endYear",
```

```
  endPer_cName = "endPeriod",
  val_cName = "value"
)
```

Arguments

`in_ts`	(obligatoire) Objet de type série chronologique (« ts » ou « mts »), ou objet compatible, à convertir.
`ind_frequency`	(obligatoire) Entier spécifiant la fréquence de la série indicatrice (haute fréquence) à laquelle les étalons (séries de basse fréquence) sont liés. La fréquence d'une série chronologique correspond au nombre maximum de périodes dans une année (par exemple, 12 pour des données mensuelles, 4 pour des données trimestrielles, 1 pour des données annuelles).
`discrete_flag`	(optionnel) Argument logique (*logical*) précisant si les étalons correspondent à des valeurs discrètes (points d'ancrage couvrant une seule période de la série indicatrice, par exemple des stocks de fin d'année) ou non. `discrete_flag = FALSE` définit des étalons non discrets, c'est-à-dire des étalons qui couvrent plusieurs périodes de la série indicatrice (par exemple, des étalons annuels couvrent 4 trimestres ou 12 mois, des étalons trimestriels couvrent 3 mois, etc.). **La valeur par défaut** est `discrete_flag = FALSE`.
`alignment`	(optionnel) Caractère identifiant l'alignement des étalons discrets (argument `discrete_flag = TRUE`) dans la fenêtre de couverture de l'intervalle de l'étalon (série de basse fréquence) : • `alignment = "b"` : début de la fenêtre de l'intervalle de l'étalon (première période) • `alignment = "e"` : fin de la fenêtre de l'intervalle de l'étalon (dernière période) • `alignment = "m"` : milieu de la fenêtre de l'intervalle de l'étalon (période du milieu) Cet argument n'a pas d'effet pour les étalons non discrets (`discrete_flag = FALSE`). **La valeur par défaut** est `alignment = "b"`.
`bmk_interval_start`	(optionnel) Entier dans l'intervalle [1 .. `ind_frequency`] spécifiant la période (cycle) de la série indicatrice (haute fréquence) à laquelle commence la fenêtre de l'intervalle de l'étalon (série de basse fréquence). Par exemple, des étalons annuels correspondant à des années financières définies d'avril à mars de l'année suivante seraient spécifiés avec `bmk_interval_start = 4` pour une série indicatrice mensuelle (`ind_frequency = 12`) et `bmk_interval_start = 2` pour une série indicatrice trimestrielle (`ind_frequency = 4`). **La valeur par défaut** est `bmk_interval_start = 1`.

`startYr_cName`, `startPer_cName`, `endYr_cName`, `endPer_cName`
: (optionnel)

Chaînes de caractères spécifiant le nom des variables (colonnes) numériques dans le *data frame* de sortie qui définiront la couverture des étalons, c'est-à-dire les identificateurs de l'année et de la période de début et de fin des étalons.

Les valeurs par défaut sont `startYr_cName = "startYear"`, `startPer_cName = "startPeriod"` `endYr_cName = "endYear"` et `endPer_Name = "endPeriod"`.

`val_cName`
: (optionnel)

Chaîne de caractères spécifiant le nom de la variable (colonne) dans le *data frame* de sortie qui contiendra les valeurs des étalons. Cet argument n'a aucun effet pour les objets « mts » (les noms des variables d'étalons sont automatiquement hérités de l'objet « mts »).

La valeur par défaut est `val_cName = "value"`.

Value

La fonction renvoie un *data frame* avec cinq variables ou plus :

- Année de début de la couverture de l'étalon, type numérique (voir argument `startYr_cName`)
- Période de début de la couverture de l'étalon, type numérique (voir argument `startPer_cName`)
- Année de fin de la couverture de l'étalon, type numérique (voir argument `endYr_cName`)
- Période de fin de la couverture de l'étalon, type numérique (voir argument `endPer_cName`)
- Une (objet « ts ») ou plusieurs (objet « mts ») variable(s) de données d'étalons, type numérique (voir argument `val_cName`)

Note : la fonction renvoie un objet « data.frame » qui peut être explicitement converti en un autre type d'objet avec la fonction `as*()` appropriée (ex., `tibble::as_tibble()` le convertirait en tibble).

See Also

`ts_to_tsDF()` `stack_bmkDF()` `benchmarking()` `stock_benchmarking()` time_values_conv

Examples

```
# Séries chronologiques annuelle et trimestrielle
ma_sc_ann <- ts(1:5 * 100, start = 2019, frequency = 1)
ma_sc_ann
ma_sc_tri <- ts(1:5 * 10, start = c(2019, 1), frequency = 4)
ma_sc_tri


# Étalons annuels pour des séries indicatrices mensuelles
ts_to_bmkDF(ma_sc_ann, ind_frequency = 12)

# Étalons annuels pour des série indicatrices trimestrielles
ts_to_bmkDF(ma_sc_ann, ind_frequency = 4)

# Étalons trimestriels pour des séries indicatrices mensuelles
```

```
ts_to_bmkDF(ma_sc_tri, ind_frequency = 12)

# Stocks de début d'année pour des séries indicatrices trimestrielles
ts_to_bmkDF(ma_sc_ann, ind_frequency = 4,
            discrete_flag = TRUE)

# Stocks de fin de trimestre pour des séries indicatrices mensuelles
ts_to_bmkDF(ma_sc_tri, ind_frequency = 12,
            discrete_flag = TRUE, alignment = "e")

# Étalons annuels (avril à mars) pour des séries indicatrices ...
# ... mensuelles
ts_to_bmkDF(ma_sc_ann, ind_frequency = 12,
            bmk_interval_start = 4)
# ... trimestrielles
ts_to_bmkDF(ma_sc_ann, ind_frequency = 4,
            bmk_interval_start = 2)

# Stocks de fin d'année (avril à mars) pour des séries indicatrices ...
# ... mensuelles
ts_to_bmkDF(ma_sc_ann, ind_frequency = 12,
            discrete_flag = TRUE, alignment = "e", bmk_interval_start = 4)
# ... trimestrielles
ts_to_bmkDF(ma_sc_ann, ind_frequency = 4,
            discrete_flag = TRUE, alignment = "e", bmk_interval_start = 2)

# Nom personnalisé pour la variable (colonne) des étalons
ts_to_bmkDF(ma_sc_ann, ind_frequency = 12,
            val_cName = "eta_val")

# Séries chronologiques multiples: argument `val_cName` ignoré
# (les noms des colonnes de l'object « mts » sont toujours utilisés)
ts_to_bmkDF(ts.union(ser1 = ma_sc_ann, ser2 = ma_sc_ann / 10), ind_frequency = 12,
            val_cName = "nom_de_colonne_inutile")
```

`ts_to_tsDF` *Convertir un objet « ts » en* data frame *de séries chronologiques*

Description

Convertir un objet « ts » (ou « mts ») en un *data frame* de séries chronologiques pour les fonctions d'étalonnage avec trois variables (colonnes) ou plus :

- deux (2) pour l'identification du point de données (année et période)
- une (1) pour chaque série chronologique

Pour des étalons discrets (points d'ancrage couvrant une seule période de la série indicatrice, par exemple, des stocks de fin d'année), spécifiez `discrete_flag = TRUE` et `alignment = "b"`, `"e"` ou `"m"`.

Usage

```
ts_to_tsDF(
  in_ts,
  yr_cName = "year",
  per_cName = "period",
  val_cName = "value"
)
```

Arguments

`in_ts`	(obligatoire) Objet de type série chronologique (« ts » ou « mts »), ou objet compatible, à convertir.
`yr_cName`, `per_cName`	(optionnel) Chaînes de caractères spécifiant le nom des variables (colonnes) numériques dans le *data frame* de sortie qui contiendront les identificateurs d'année et de période du point de données. **Les valeurs par défaut** sont `yr_cName = "year"` et `per_cName = "period"`.
`val_cName`	(optionnel) Chaîne de caractères spécifiant le nom de la variable (colonne) dans le *data frame* de sortie qui contiendra les valeurs des points de données. Cet argument n'a aucun effet pour les objets « mts » (les noms des variables de données des séries chronologiques sont automatiquement hérités de l'objet « mts »). **La valeur par défaut** est `val_cName = "value"`.

Value

La fonction renvoie un *data frame* avec trois variables ou plus :

- Année du point de données, type numérique (voir argument `yr_cName`)
- Période du point de données, type numérique (voir argument `per_cName`)
- Valeur du point de données, type numérique (voir argument `val_cName`)
- Une (objet « ts ») ou plusieurs (objet « mts ») variable(s) de données de série(s) chronologique(s), type numérique (voir argument `val_cName`)

Note : la fonction renvoie un objet « data.frame » qui peut être explicitement converti en un autre type d'objet avec la fonction `as*()` appropriée (ex., `tibble::as_tibble()` le convertirait en tibble).

See Also

`tsDF_to_ts()` `ts_to_bmkDF()` `stack_tsDF()` `benchmarking()` `stock_benchmarking()` time_values_conv

Examples

```
# Série chronologique Quarterly time series
ma_sc <- ts(1:10 * 100, start = 2019, frequency = 4)
ma_sc


# Noms de variables (colonnes) par défaut
ts_to_tsDF(ma_sc)

# Nom personnalisé pour la variable (colonne) des étalons
ts_to_tsDF(ma_sc, val_cName = "ser_val")


# Séries chronologiques multiples: argument `val_cName` ignoré
# (les noms de colonnes de l'object « mts » sont toujours utilisés)
ts_to_tsDF(ts.union(ser1 = ma_sc,
                    ser2 = ma_sc / 10),
           val_cName = "nom_de_colonne_inutile")
```

---

unstack_tsDF *Fonction réciproque de* stack_tsDF()

Description

Convertir un *data frame* empilé (long) de séries chronologiques multivariées (format de données de benchmarking() et stock_benchmarking()) en un *data frame* non empilé (large) de séries chronologiques multivariées.

Cette fonction, combinée avec tsDF_to_ts(), est utile pour convertir le *data frame* renvoyé par un appel à benchmarking() ou stock_benchmarking() en un objet « mts », où plusieurs séries ont été étalonnées en mode de traitement *groupes-BY*.

Usage

```
unstack_tsDF(
  ts_df,
  ser_cName = "series",
  yr_cName = "year",
  per_cName = "period",
  val_cName = "value"
)
```

Arguments

ts_df	(obligatoire) *Data frame*, ou objet compatible, contenant les données de séries chronologiques multivariées à *désempiler*.

`ser_cName` (optionnel)

Chaîne de caractères spécifiant le nom de la variable (colonne) dans le *data frame* d'entrée qui contient le nom des séries chronologiques (nom des variables des séries chronologiques dans le *data frame* de sortie).

La valeur par défaut est `ser_cName = "series"`.

`yr_cName, per_cName`

(optionnel)

Chaînes de caractères spécifiant le nom des variables (colonnes) numériques dans le *data frame* d'entrée qui identifient l'année et la période des points de données. Ces variables sont *transférées* dans le *data frame* de sortie avec les mêmes noms de variable.

Les valeurs par défaut sont `yr_cName = "year"` et `per_cName = "period"`.

`val_cName` (optionnel)

Chaîne de caractères spécifiant le nom de la variable (colonne) numérique dans le *data frame* d'entrée qui contient la valeur des points de données.

La valeur par défaut est `val_cName = "value"`.

Value

La fonction renvoie un *data frame* avec trois variables ou plus :

- Année du point de données, type numérique (voir argument `yr_cName`)
- Période du point de données, type numérique (voir argument `per_cName`)
- Une variable de données de série chronologique pour chaque valeur distincte de la variable du *data frame* d'entrée spécifiée avec l'argument `ser_cName`, type numérique (voir arguments `ser_cName` et `val_cName`)

Note : la fonction renvoie un objet « data.frame » qui peut être explicitement converti en un autre type d'objet avec la fonction `as*()` appropriée (ex., `tibble::as_tibble()` le convertirait en tibble).

See Also

`stack_tsDF()` `tsDF_to_ts()` `benchmarking()` `stock_benchmarking()`

Examples

```
# Étalonnage proportionnel pour plusieurs (3) séries trimestrielles traitées avec
# l'argument `by` (en mode groupes-BY)

vec_ind <- c(1.9, 2.4, 3.1, 2.2, 2.0, 2.6, 3.4, 2.4, 2.3)
df_ind <- ts_to_tsDF(ts(data.frame(ser1 = vec_ind,
                                   ser2 = vec_ind * 100,
                                   ser3 = vec_ind * 10),
                        start = c(2015, 1), frequency = 4))

vec_eta <- c(10.3, 10.2)
df_eta <- ts_to_bmkDF(ts(data.frame(ser1 = vec_eta,
                                    ser2 = vec_eta * 100,
```

```
                                    ser3 = vec_eta * 10),
                          start = 2015, frequency = 1),
                        ind_frequency = 4)

res_eta <- benchmarking(stack_tsDF(df_ind),
                        stack_bmkDF(df_eta),
                        rho = 0.729, lambda = 1, biasOption = 3,
                        by = "series",
                        quiet = TRUE)

# « Data frame » des séries chronologiques initiales et finales (étalonnés)
df_ind
unstack_tsDF(res_eta$series)
```

Index