Outils pour l'estimation de variance dans les enquêtes de l'Insee : le *package* R gustave

Martin Chevalier (DMS)

23 août 2018

Outils pour l'estimation de variance dans les enquêtes de l'Insee : le *package* R gustave

Objectif de la réunion Présenter les outils développés récemment par la Division Sondages pour l'estimation de variance à travers le *package* R gustave.

Outils « prêts-à-estimer » pour les cas les plus simples

Boîte à outils pour les cas plus complexes

Principes et outils de développement

Outils « prêts-à-estimer » pour les cas les plus simples

Retour sur l'existant

Plusieurs outils ont été développés ces dernières années pour estimer la variance d'estimateurs calculés dans des **enquêtes relativement simples** :

- macro SAS `%calker` : sondage aléatoire simple stratifié, correction de la non-réponse par repondération ou par imputation au sein des strates de tirage ;
- macro SAS `%calker_grh` : sondage aléatoire simple stratifié, correction de la non-réponse par repondération au sein de groupes de réponse homogènes quelconques ;
- macro SAS `%everest` : sondage aléatoire simple stratifié, correction de la non-réponse par imputation au sein des strates de tirage ou par repondération au sein de groupes de réponse homogènes quelconques, calage sur marges.

La fonction `qvar()` du *package* gustave

gustave est un ***package* R** fournissant une **boîte à outils pour l'estimation de variance**.

`qvar()` est une *fonction* du *package* gustave, qui **combine les autres fonctions proposées par gustave** dans un outil « prêt-à-estimer » dans les cas les plus simples, à savoir :

- sondage aléatoire simple stratifié ;
- correction de la non-réponse par repondération dans des groupes de réponse homogènes ;
- calage sur marges.

En ce sens, `gustave::qvar()` est **susceptible de remplacer la macro SAS `%everest`**.

Remarque Le nom « qvar » signifie « *quick variance estimation* » et est pensé par analogie avec la fonction `qplot()` du *package* ggplot2.

Démonstration : Installation et chargement du *package* gustave

La version 0.3.0 du *package* gustave est accessible sur le dépôt de *packages* classique de R (le CRAN).

La version 0.4.0 en développement est disponible sur github.com et sur le serveur gitlab de l'Insee.

```
# Installation de la version de développement depuis github
devtools::install_github("martinchevalier/gustave", ref = "dev")

# Chargement du package gustave
library(gustave)

# Affichage de l'aide de la fonction qvar()
? qvar
```

Démonstration : Données d'exemple

Le *package* gustave embarque des **données d'exemple simulées** inspirées de l'enquête Technologies de l'information et de la communication.

```
# Table d'échantillon ict_sample
names(ict_sample)
##  [1] "firm_id"          "division"         "employees"
##  [4] "turnover"         "strata"           "w_sample"
##  [7] "scope"            "resp"             "no_reweighting"
## [10] "nrc"              "hrg"              "response_prob_est"
## [13] "w_nrc"            "calib"            "N_58"
## [16] "N_59"             "N_60"             "N_61"
## [19] "N_62"             "N_63"             "turnover_58"
## [22] "turnover_59"      "turnover_60"      "turnover_61"
## [25] "turnover_62"      "turnover_63"      "w_calib"
## [28] "dissemination"

# Table de variables d'intérêt ict_survey
names(ict_survey)
##  [1] "firm_id"          "division"         "employees"
##  [4] "turnover"         "w_calib"          "speed_quanti"
##  [7] "speed_quanti_NA"  "speed_quali"      "speed_quali_NA"
## [10] "big_data"         "big_data_NA"
```

Démonstration : Plan de sondage (1)

```
# Estimation ne tenant compte que de l'échantillonnage
qvar(

  # Paramètres méthodologiques
  data = ict_sample,
  dissemination_dummy = "dissemination",
  dissemination_weight = "w_sample",
  id = "firm_id",
  sampling_weight = "w_sample",

  # Variable d'intérêt
  turnover

)
## Survey variance estimation with the gustave package
##
## The following features are taken into account:
##   - simple random sampling WITHOUT stratification
## Warning: The following strata contain units whose sampling weights are not e
##                   call   n       est     variance      std      cv
## 1 total(y = turnover) 506 138423007 2.796131e+14 16721637 12.0801
##       lower     upper
## 1 105649202 171196812
```

Démonstration : Plan de sondage (2)

```
# Ajout de la strate de tirage
qvar(

  # Paramètres méthodologiques
  data = ict_sample,
  dissemination_dummy = "dissemination",
  dissemination_weight = "w_sample",
  id = "firm_id",
  sampling_weight = "w_sample", strata = "strata",

  # Variable d'intérêt
  turnover

)
## Survey variance estimation with the gustave package
##
## The following features are taken into account:
##   - stratified simple random sampling
## Note: The strata variable (strata) is of type character. It is automatically
##                   call   n       est     variance      std       cv
## 1 total(y = turnover) 506 138423007 2.349875e+14 15329303 11.07425
##       lower     upper
## 1 108378125 168467890
```

Démonstration : Fixation des paramètres méthodologiques

```
# Définition d'une fonction avec les paramètres méthodologiques de TIC
precisionTic <- qvar(
  data = ict_sample,
  dissemination_dummy = "dissemination",
  dissemination_weight = "w_sample",
  id = "firm_id",
  sampling_weight = "w_sample", strata = "strata",
  define = TRUE
)
## Survey variance estimation with the gustave package
##
## The following features are taken into account:
##   - stratified simple random sampling
## Note: The strata variable (strata) is of type character. It is automatically
## Note: As define = TRUE, a ready-to-use variance wrapper is (invisibly) retur

# Utilisation de la fonction
precisionTic(ict_survey, turnover)
##                   call   n       est     variance      std       cv
## 1 total(y = turnover) 506 138423007 2.349875e+14 15329303 11.07425
##       lower     upper
## 1 108378125 168467890
```

Démonstration : Ergonomie (1)

```
# Estimation de moyennes et de ratio
precisionTic(ict_survey, mean(turnover))
##                  call   n      est variance      std       cv    lower
## 1 mean(y = turnover) 506 22536.97  6016444 2452.844 10.88365 17729.48
##      upper
## 1 27344.45
precisionTic(ict_survey, ratio(turnover, employees))
##                                        call   n      est variance      std
## 1 ratio(num = turnover, denom = employees) 506 251.7953 842.8196 29.03136
##         cv    lower    upper
## 1 11.52974 194.8949 308.6957

# Plusieurs estimateurs en un seul appel
precisionTic(ict_survey,
  "CA moyen" = mean(turnover),
  "CA moyen par salarié" = ratio(turnover, employees)
)
##                   label                                    call   n
## 1             CA moyen                      mean(y = turnover) 506
## 2 CA moyen par salarié ratio(num = turnover, denom = employees) 506
##          est     variance        std       cv      lower      upper
## 1 22536.9667 6016443.9018 2452.84404 10.88365 17729.4807 27344.4527
## 2   251.7953     842.8196   29.03136 11.52974   194.8949   308.6957
```

Démonstration : Ergonomie (2)

```
# Estimation sur un domaine
precisionTic(ict_survey, mean(speed_quanti), where = employees >= 50)[, 1:4]
##                                                call   n      est variance
## 1 mean(y = speed_quanti, where = employees >= 50) 193 42.50389 12.86715

# Estimation sur plusieurs domaines
precisionTic(ict_survey, mean(speed_quanti), by = division)[, 1:5]
##                                        call by   n      est variance
## 1 mean(y = speed_quanti, by = division) 58 152 36.17213 10.41510
## 2 mean(y = speed_quanti, by = division) 59  69 19.75255 10.97779
## 3 mean(y = speed_quanti, by = division) 60  35 54.18026 45.29168
## 4 mean(y = speed_quanti, by = division) 61  41 84.49828 71.62625
## 5 mean(y = speed_quanti, by = division) 62 170 35.73460  9.98174
## 6 mean(y = speed_quanti, by = division) 63  39 36.11887 40.89093

# Domaines différents par estimateur
precisionTic(ict_survey,
  "CA moyen des 50 et plus" = mean(turnover, where = employees >= 50),
  "CA moyen des 100 et plus" = mean(turnover, where = employees >= 100)
)[, c(1, 3:5)]
##                       label   n      est variance
## 1  CA moyen des 50 et plus 193 27980.13 38815787
## 2 CA moyen des 100 et plus 137 32736.42 81050421
```

Démonstration : Ergonomie (3)

```
# Dichotomisation à la volée
precisionTic(ict_survey, mean(speed_quanti > 100))[, 1:4]
##                            call   n       est     variance
## 1 mean(y = speed_quanti > 100) 506 0.1590424 0.0003121758

# Dichotomisation automatique des variables qualitatives
precisionTic(ict_survey, mean(speed_quali))[, 1:4]
##                     call                     mod   n        est
## 1 mean(y = speed_quali)        Less than 2 Mbs 506 0.03352201
## 2 mean(y = speed_quali)   Between 2 and 10 Mbs 506 0.33934964
## 3 mean(y = speed_quali)  Between 10 and 30 Mbs 506 0.32479578
## 4 mean(y = speed_quali) Between 30 and 100 Mbs 506 0.14329014
## 5 mean(y = speed_quali)          Above 100 Mbs 506 0.15904243

# Intégration avec dplyr et %>%
library(dplyr)
ict_survey %>%
  precisionTic("Connexion supérieure à 100 Mbs" = mean(speed_quanti > 100)) %>%
  select(label, est, lower, upper)
##                            label       est     lower     upper
## 1 Connexion supérieure à 100 Mbs 0.1590424 0.1244128 0.1936721
```

Démonstration : Prise en compte du champ

```
# Ajout de l'indicatrice d'appartenance au champ
precisionTic <-  qvar(
  data = ict_sample,
  dissemination_dummy = "dissemination",
  dissemination_weight = "w_sample",
  id = "firm_id",
  sampling_weight = "w_sample", strata = "strata",
  scope_dummy = "scope"
)
## Survey variance estimation with the gustave package
##
## The following features are taken into account:
##   - stratified simple random sampling
##   - out-of-scope units
## Note: The strata variable (strata) is of type character. It is automatically
## Note: No variable to perform variance estimation on are specified. A ready-t
```

Démonstration : Correction de la non-réponse (1)

```
# Ajout de la correction de la non-réponse
precisionTic <- qvar(
  data = ict_sample,
  dissemination_dummy = "dissemination",
  dissemination_weight = "w_nrc",
  id = "firm_id",
  sampling_weight = "w_sample", strata = "strata",
  scope_dummy = "scope",
  nrc_weight = "w_nrc", response_dummy = "resp", hrg = "hrg",
  define = TRUE
)
## Survey variance estimation with the gustave package
##
## The following features are taken into account:
##   - stratified simple random sampling
##   - out-of-scope units
##   - non-response correction through reweighting
## Note: The strata variable (strata) is of type character. It is automatically
## Note: As define = TRUE, a ready-to-use variance wrapper is (invisibly) retur
```

Démonstration : Correction de la non-réponse (2)

```
# Cas d'unités exclues de la repondération
precisionTic <- qvar(
  data = ict_sample,
  dissemination_dummy = "dissemination",
  dissemination_weight = "w_nrc",
  id = "firm_id",
  sampling_weight = "w_sample", strata = "strata",
  scope_dummy = "scope",
  nrc_weight = "w_nrc", response_dummy = "resp", hrg = "hrg",
  nrc_dummy = "nrc",
  define = TRUE
)
## Survey variance estimation with the gustave package
##
## The following features are taken into account:
##   - stratified simple random sampling
##   - out-of-scope units
##   - non-response correction through reweighting
## Note: The strata variable (strata) is of type character. It is automatically
## Note: As define = TRUE, a ready-to-use variance wrapper is (invisibly) retur
```

Démonstration : Calage sur marges (1)

```
# Ajout du calage sur marges
precisionTic <- qvar(
  data = ict_sample,
  dissemination_dummy = "dissemination",
  dissemination_weight = "w_calib",
  id = "firm_id",
  sampling_weight = "w_sample", strata = "strata",
  scope_dummy = "scope",
  nrc_weight = "w_nrc", response_dummy = "resp", hrg = "hrg",
  nrc_dummy = "nrc",
  calibration_weight = "w_calib",
  calibration_var = c(paste0("N_", 58:63), paste0("turnover_", 58:63)),
  define = TRUE
)
## Survey variance estimation with the gustave package
##
## The following features are taken into account:
##   - stratified simple random sampling
##   - out-of-scope units
##   - non-response correction through reweighting
##   - calibration on margins
## Note: The strata variable (strata) is of type character. It is automatically
## Note: As define = TRUE, a ready-to-use variance wrapper is (invisibly) retur
```

Démonstration : Calage sur marges (2)

```
# Cas d'unités exclues du calage
precisionTic <- qvar(
  data = ict_sample,
  dissemination_dummy = "dissemination",
  dissemination_weight = "w_calib",
  id = "firm_id",
  sampling_weight = "w_sample", strata = "strata",
  scope_dummy = "scope",
  nrc_weight = "w_nrc", response_dummy = "resp", hrg = "hrg",
  nrc_dummy = "nrc",
  calibration_weight = "w_calib",
  calibration_var = c(paste0("N_", 58:63), paste0("turnover_", 58:63)),
  calibration_dummy = "calib",
  define = TRUE
)
## Survey variance estimation with the gustave package
##
## The following features are taken into account:
##   - stratified simple random sampling
##   - out-of-scope units
##   - non-response correction through reweighting
##   - calibration on margins
## Note: The strata variable (strata) is of type character. It is automatically
## Note: As define = TRUE, a ready-to-use variance wrapper is (invisibly) retur
```

Pour synthétiser

`gustave::qvar()` facilite la production de programmes d'estimation de variance relativement ergonomiques **dans les cas les plus simples**.

Cette fonction opère par ailleurs un **grand nombre de contrôles techniques et méthodologiques**.

Plusieurs **limitations** :

- elle ne prend pas en compte la correction de la non-réponse par imputation (contrairement à `%everest`) ;
- elle ne permet pas de tenir compte des plans de sondage complexes : degrés multiples, partage des poids.

Les **autres fonctionnalités du *package* gustave** apportent des éléments de réponse à ces questions.

Boîte à outils pour les cas plus complexes

Principes du *package* gustave (1)

Gustave : a **U**ser-oriented **S**tatistical **T**oolkit for **A**nalytical **V**ariance **E**stimation

gustave est avant tout une « boîte à outils » (*toolkit*), i.e. une **collection de fonctions utiles au ou à la méthodologue** pour estimer la variance d'estimateurs issus d'une enquête par sondage.

En ce sens, son approche diffère sensiblement de celle proposée par la macro SAS `%poulpe` (*cf.* les actes des JMS 1998) :

- `%poulpe` : macro complète qui met en œuvre un cadre méthodologique à portée universelle ;
- gustave : ensemble de « petites » fonctions qui simplifient le travail du ou de la méthodologue.

Principes du *package* gustave (2)

D'un point de vue organisationnel, gustave s'inscrit dans une organisation du travail à **deux niveaux** :

1. **Méthodologue** : analyse méthodologique, mobilisation de l'information auxiliaire, construction d'un **programme d'estimation de variance** raisonnablement exact ;
2. **Responsable d'enquête, chargé(e) d'étude** : utilisation du programme d'estimation dans le cadre d'études ou pour répondre à des obligations réglementaires.

Conséquence : les programmes d'estimation de variance doivent donc

- être autonomes et aussi simples d'utilisation que possible ;
- prendre en compte l'ensemble des éléments relatifs au calcul de précision ;
- ne pas être trop complexes à développer ni à maintenir.

Principes du *package* gustave (3)

Idée centrale « Emballer » la fonction d'estimation de variance complexe dans une autre fonction (appelée « *wrapper* ») plus simple d'utilisation :

- **fonction d'estimation de la variance** : fonction spécifique à chaque enquête développée par le ou la méthodologue ;
  $\rightarrow$ **complexité méthodologique**

- ***wrapper* d'estimation de variance** : fonction générique qui prend en charge des opérations systématiques (calcul des statistiques, domaines), appelle la fonction de variance et affiche les résultats.
  $\rightarrow$ **complexité informatique**

Contenu du *package* gustave

De ce fait, les fonctions du *package* gustave sont de deux ordres :

1. **Fonctions méthodologiques** : estimateurs de variance usuels (Sen-Yates-Grundy, Deville-Tillé pour le sondage équilibré, etc.), fonctions outils (calcul des résidus de calage, somme par groupe, etc.) ;
   $\rightarrow$ faciliter l'écriture de la **fonction de variance**

2. **Fonctions techniques** : fonctions qui simplifient la production de programmes d'estimation de variance qui ne nécessitent pas d'expertise méthodologique particulière.
   $\rightarrow$ produire le ***wrapper* de variance**

Remarque `qvar()` n'est qu'un **appel pré-paramétré** produisant des ***wrappers* de variance**.

Démonstration : Précision de l'EEC

La production des programmes de calcul de précision pour l'enquête Emploi en continu (EEC) est désormais **directement intégrée à la chaîne de production trimestrielle** de l'enquête.

Le module spécifique de la chaîne de production est une **mise en œuvre des fonctions du *package* gustave**.

Conceptuellement, le production du programme de calcul de précision pour un trimestre de l'EEC comporte **deux étapes** :

1. Ecriture de la fonction de variance ;
2. Création du *wrapper* de variance.

Démonstration : Fonction de variance (1)

L'écriture de la fonction de variance spécifique à une enquête découle de l'**analyse de sa méthodologie**, notamment pour déterminer l'**information auxiliaire** nécessaire.

Ici la fonction de variance prend les arguments suivants :

- `y` : variables d'intérêt (sous la forme d'une matrice) sur lesquelles faire porter l'estimation de variance ;
- `up` : information auxiliaire relative aux unités primaires (« bisecteurs ») ;
- `log` : information auxiliaire relative aux logements ;
- `ind` : information auxiliaire relative aux individus.

```
varEec <- function(y, up, log, ind){

  variance <- list()

  # Etape 0 : Agrégation par logement
  y <- sum_by(y, by = ind$idlog)

  # Etape 1 : Prise en compte du calage
  y <- add_zero(y, log$id[log$cal])
  y <- res_cal(y, precalc = log$res_cal_precalc)
```

Démonstration : Fonction de variance (2)

```
  (...)

  # Etape 2 : Prise en compte de la non-réponse
  variance[["nr"]] <- colSums(
    (1/log$pilog[log$cal]^2 - log$qlog[log$cal]) *
      (1 - log$pinr[log$cal]) * (y/log$pinr[log$cal])^2
  )
  y <- add_zero(y / log$pinr[log$cal], log$id)

  # Etape 3 : Sélection des logements dans les up
  variance[["log"]] <- varDT(
    y, w = 1/(log$piup^2) - log$qup,
    precalc = log$varDT_precalc
  )

  # Etape 4 : Sélection des up
  y <- sum_by(y, by = log$idup, w = 1/log$pilog_up)
  y <- add_zero(y, up$id)
  variance[["up"]] <- varDT(y, precalc = up$precalc)

  colSums(do.call(rbind, variance))

}
```

Démonstration : *Wrapper* de variance

À partir de la fonction de variance et de l'information auxiliaire nécessaire, la fonction `define_variance_wrapper()` crée un *wrapper* de variance simple d'utilisation.

```
# Création du wrapper de variance avec define_variance_wrapper()
precisionEec <- define_variance_wrapper(
  variance_function = varEec,
  technical_data = list(up = up, log = log, ind = ind),
  reference_id = technical_data$ind$id,
  reference_weight = technical_data$ind$w,
  default_id = quote(paste0(ident, noi))
)

# Utilisation du wrapper de variance (données du T4 2014)
precisionEec(z, acteu %in% 2)
```

```
##                       call     est   variance      std       cv   lower
## 1 total(y = acteu %in% 2) 3001046 2158830156 46463.21 1.548234 2909980
##     upper
## 1 3092112
```

Remarque Le *wrapper* de variance est une fonction complètement autonome : toute l'information auxiliaire spécifiée au paramètre `technical_data` est intégrée *dans* la fonction (il s'agit d'une *closure*).

Un *package* pensé pour être extensible (1)

La fonction `define_variance_wrapper()` accepte **n'importe quelle fonction de variance en entrée** :

- autant d'information auxiliaire que nécessaire ;
- utilisation des fonctions d'autres *packages* pour coder la fonction de variance (utiliser `require()` dans la fonction de variance) ;
- prise en compte de paramètres affectant l'estimation de variance (méthodologies alternatives, etc.).

Large éventail de méthodologies couvert à ce jour :

- échantillons tirés dans l'échantillon-maître Octopusse (formule spécifique dérivée de la formule de Sen-Yates-Grundy) ;
- degrés multiples (CVS) ;
- partage des poids complexes (SRCV).

Un *package* pensé pour être extensible (2)

La fonction de variance peut exporter, en plus des variances estimées, des **résultats intermédiaires** de l'estimation de variance.

Cette fonctionnalité facilite la création de **surcouches** à partir des *wrappers* de variance produits par le *package* gustave.

Exemple Dans l'EEC, l'estimation de variance pour des indicateurs faisant intervenir plusieurs trimestres (évolution d'un trimestre à l'autre, moyennes annuelles, etc.) s'appuie sur la récupération des résultats intermédiaires des *wrappers* de variance de chaque trimestre concerné (estimation des covariances trimestrielles).

Un *package* pensé pour être extensible (3)

Il est également possible de définir de nouvelles fonctions pour estimer la précision de **statistiques complexes** grâce à la fonction `define_statistic_wrapper()` :

```
# Définition du coefficient de gini à partir du package vardpoor
gini <- define_statistic_wrapper(
  statistic_function = function(y, weight){
    require(vardpoor)
    result <- lingini(Y = y, weight = weight)
    list(point = result$value$Gini, lin = result$lin$lin_gini)
  },
  arg_type = list(data = "y", weight = "weight", param = NULL)
)

# Utilisation pour calculer la précision dans l'enquête SRCV en 2014
precisionSrcv(r, gini(HX090))
```

```
##              call      est  variance       std      cv    lower    upper
## 1 gini(y = HX090) 29.21328 0.1013441 0.3183458 1.08973 28.58933 29.83722
```

Pour synthétiser

Au-delà de la fonction « prête-à-estimer » `qvar()`, le *package* gustave est pensé comme une **boîte à outils pour le ou la méthodologue** en charge de l'estimation de variance.

Il rassemble des fonctions méthodologiques et techniques qui **facilitent la création de programmes d'estimation de variance simples d'utilisation et autonomes**.

Sa conception est pensée pour que ses fonctionnalités soient **le plus extensibles possibles**.

Ce souci est également présent dans ses **principes de développement**.

Principes et outils de développement

Objectif : maximiser la maintenabilité

Le développement d'un outil de calcul de précision conduit à affronter **deux sources de complexité** :

- complexité **méthodologique** : méthodes souvent ardues et peu enseignées en formation initiale ;
- complexité **informatique** : outils complexes pour obtenir des programmes ergonomiques pour le non-spécialiste (gestion fine des environnements, évaluation non-standard, etc.).

Dans ce contexte, le risque est élevé d'aboutir à un **outil très difficilement maintenable dans le temps**, notamment dans le contexte des mobilités à l'Insee.

Solution mise en œuvre Adopter des **principes de développement qui maximisent la maintenabilité du code**, inspirés de l'univers du **logiciel libre**.

Principes (1) : Logiciel libre et *package*

Utilisation de R : logiciel libre conçu pour l'exploitation de données statistiques, R présente davantage de garanties de pérennité (à l'Insee et en général) que des alternatives (SAS notamment).

Plus encore, R offre la possibilité de **structurer un ensemble de codes sous la forme de *packages*** :

- gestion des conflits de noms et des dépendances ;
- documentation de l'ensemble des fonctions accessibles à l'utilisateur ;
- intégration de données d'exemple ;
- diffusion sur le CRAN : validation technique du *package* et installation facile.

Pour en savoir plus Développer un *package* avec RStudio et git

Principes (2) : Suivi de versions

Les évolutions du *package* sont **suivies en version** depuis l'été 2017 :

- **code source librement accessible** sur plusieurs plateformes de développement : github.com, gitlab interne de l'Insee ;
- **conservation de toutes les versions** (plus de 300 *commits* à ce jour) avec leurs métadonnées : une description est associée à chaque ensemble cohérent de modifications ;
- **travail collaboratif facilité**, y compris de façon concomittante : création de branches pour des développements particulier, gestion des conflits ;
- possibilité pour des utilisateurs externes de **proposer efficacement des modifications** (remontée de *bugs*, demandes spécifiques, *pull requests*).

Principes (3) : Tests unitaires et intégration en continu

Le développement sous la forme de *packages* favorise également le développement de **tests unitaires** :

1. À chaque fonctionnalité du *package* est associé un **test** qui vérifie son bon fonctionnement.
2. Au cours du développement, il est très facile de rejouer tous les tests unitaires pour garantir la **non-régression**.

   → gustave comporte plus de **180 tests unitaires**.

Il est possible d'associer suivi de versions et tests unitaires *via* l'**intégration en continu** :

- à chaque nouvelle version du *package*, des tests sont automatiquement réalisés ;
- une notification est envoyée en cas d'échec.

Principes (4) : Formation et organisation du travail

Le maintien d'un programme dans le temps ne dépend pas que de ses caractéristiques techniques, mais aussi du cadre organisationnel :

- **formations au développement avec R** : initiation, perfectionnement, développement de *packages*, gestion de version ;
- participation à l'**amélioration des outils de développement** : utilisation de git dans RStudio sur AUS, investissement dans les outils proposés par la plateforme Innovation ;
- organisation sous forme de **binôme** : passation longue avec Nicolas Paliod en 2017-2018, y compris sur les points les plus techniques du *package* ;
- **impulsion collective** au sein de la section Méthodes d'estimation : investissement de moyen terme dans le développement de *packages* R et les outils collaboratifs.

De nouvelles possibilités de collaboration

Le modèle de développement du *package* gustave est résolument **ouvert et horizontal**. Tout un chacun peut ainsi :

- utiliser la **toute dernière version** en cours de développement ;
- **relire le code** pour comprendre son fonctionnement, détecter d'éventuelles erreurs et **proposer un correctif** ;
- **proposer une nouvelle fonctionnalité** et éventuellement des éléments pour la mettre en œuvre.

Les plateformes de développement comme github.com ou gitlab permettent de facilement communiquer autour du code :

- ticket d'incident (*bug report*) ;
- proposition de nouvelle fonctionnalité (*feature request*) ;
- proposition de code à intégrer dans le *package* (*pull request*).

Ces nouvelles méthodes de travail et ces nouveaux outils ouvrent de **nouvelles possibilités de collaboration** autour des questions méthodologiques.

Merci de votre attention !

Martin Chevalier
martin.chevalier@insee.fr
https://github.com/martinchevalier/gustave