Méthodologie statistique

M2016/04

Savoir compter, savoir coder

Emmanuel L'Hour – Ronan Le Saout
Benoît Rouppert

Document de travail

Institut National de la Statistique et des Études Économiques

INSTITUT NATIONAL DE LA STATISTIQUE ET DES ÉTUDES ÉCONOMIQUES

Série des documents de travail « Méthodologie Statistique »

de la Direction de la Méthodologie et de la Coordination Statistique et Internationale

M 2016/04

Savoir compter, savoir coder
Bonnes pratiques du statisticien en programmation

Emmanuel L'Hour* - Ronan Le Saout**
Benoît Rouppert***

Les auteurs remercient pour leurs conseils et relectures Jean-William Angel, Maxime Bergeat, Pauline Givord, Jean-Michel Goillot, Marine Guillerm, Heidi Koumarianos, Alexandre Lebrère, Fabienne Perray-Gibert, Simon Quantin, Layla Ricroch, Olivier Sautory.

* Insee – CNI
15, bd Gabriel-Péri – 92245 MALAKOFF CEDEX
** Insee – DMCSI – Division des méthodes appliquées de l'économétrie et de l'évaluation
18, bd Adolphe Pinard – 75675 PARIS CEDEX 14
*** Au moment de la rédaction de ce document de travail, Insee – Unité Qualité
18, bd Adolphe Pinard – 75675 PARIS CEDEX 14

Direction de la méthodologie et de la coordination statistique et internationale -Département des Méthodes Statistiques - Timbre L101
18, bd Adolphe Pinard - 75675 PARIS CEDEX - France -
Tél. : 33 (1) 41 17 66 33 - Fax : 33 (1) 41 17 61 97 - CEDEX - E-mail :-DG75-L001@insee.fr - Site Web Insee : http://www.insee.fr

Savoir compter, savoir coder
Bonnes pratiques du statisticien en programmation

Emmanuel L'Hour * – Ronan Le Saout **
Benoît Rouppert ***

Résumé

Les statistiques sont le résultat de programmes informatiques. Lors de la publication d'une étude, ces chiffres gagnent en lisibilité avec un bon usage des techniques rédactionnelles. Peu de règles régissent par contre la programmation informatique en amont. Exécuter ou faire évoluer un programme écrit par un autre se révèle alors une tâche difficile. Or écrire un programme lisible, c'est gagner en productivité individuelle et collective, en limitant les erreurs, en facilitant la relecture et en améliorant l'évolutivité du code.

Lors de la rédaction d'une étude, l'étape de préparation, à travers la recherche documentaire, l'identification de messages clés ou l'élaboration du plan, est essentielle. De même, exprimer le besoin, identifier les moyens et planifier les actions permettent de gagner un temps précieux lors de la rédaction d'un programme informatique. Par ailleurs, construire des phrases courtes avec « une idée - une phrase » et éviter le jargon inutile clarifient un message. Dans les programmes, cela se traduira par des programmes courts et organisés et des noms expressifs. Enfin, de même qu'une publication fait l'objet de multiples relectures, faire relire le code et la définition de tests apparaissent essentiels.

Abstract

Statistics come from computer programs. In a published study, these figures are more readable when editorial techniques are well used. But few rules govern the upstream coding tasks. As a consequence, to run or to change code written by someone else can prove to be a strenuous burden. A readable code increases individual and collective work efficiency by limiting errors, by making review easier, and by making the code more sustainable.

For someone preparing a study, both conducting a literature search and identifying key messages are crucial tasks. Similarly, expressing the relevant business need, identifying the available means and planning adequate actions can save valuable time when coding. Moreover, short sentences built around “one idea – one sentence” and no useless jargon make a message clearer. Likewise, readable programs are short, well structured, and use meaningful names. Finally, in the same manner as a published study has been reviewed several times, code reviews and defining tests seem critical.

* INSEE, Centre national informatique de Paris
emmanuel.l'hour@insee.fr
** INSEE, Département des méthodes statistiques
ronan.le-saout@insee.fr
*** INSEE, Unité qualité (au moment de la rédaction)
benoit.rouppert@insee.fr

Le concept central des quelques règles présentées ici est la **lisibilité**, dans les attendus de la programmation, dans l'organisation générale des programmes et dans le code lui-même. Elles peuvent de ce point de vue se rapprocher des bonnes pratiques des **techniques rédactionnelles**[1].

Ce document s'adresse particulièrement aux statisticiens et chargés d'études, non informaticiens, écrivant des programmes associés tant à la production statistique qu'aux études socio-économiques. Il vise également à attirer l'attention des managers sur les bienfaits de ces bonnes pratiques et sur les risques à ne pas les appliquer, tant pour la gestion de court terme que pour la transmission des connaissances à plus long terme.

Les deux premières règles invitent à exprimer son besoin et à concevoir plusieurs solutions, avant de procéder à la réalisation d'une solution choisie. Les règles suivantes abordent la programmation proprement dite.

Sommaire

[1] Courrier des statistiques - Numéro Hors-série 2009 : Savoir compter, savoir conter

1. Pourquoi chercher à être lisible ?

Début 2013, une controverse est intervenue concernant une étude médiatisée de 2010 des économistes Carmen REINHART et Kenneth ROGOFF liant taux d'endettement et croissance. Les données comportaient des erreurs et ont été sélectionnées de manière contestable. Ces erreurs ont été mises à jour par un étudiant ayant reproduit les analyses de ces deux économistes reconnus.

Reproduire une analyse peut permettre d'**identifier des erreurs dans le traitement des données ou dans l'analyse méthodologique**. De nombreuses revues académiques demandent ainsi la diffusion des bases de données et des programmes lorsqu'un article est publié. La mise en forme de ces programmes n'obéit pas pour l'heure à des règles strictes, ce qui nuit à leur caractère reproductible. En plus d'une description claire et exacte de la méthodologie suivie ou de la rédaction d'une documentation, rendre son code lisible, c'est en premier lieu **améliorer son caractère reproductible par un tiers**.

Mettre en forme les programmes à la fin du projet, par exemple lorsqu'une étude économique est publiée, ne suffit pas. **Tout au long de la vie du projet** et des programmes associés, la lisibilité **améliore :**

- **le travail collaboratif** : Le code est-il lisible par un co-auteur ou mon successeur sur mon poste ? Serait-il compréhensible par quelqu'un d'extérieur ou de nouveau dans le projet ? ;

- **l'évolutivité** : Puis-je reprendre mon code plusieurs mois plus tard ? ;

- **la documentation** : Le code est-il auto-documenté (i.e. lisible sans commentaires ou documentation supplémentaire) ? ;

- et plus généralement, **toute tâche de programmation**. On passe en effet plus de temps à lire son code qu'à en écrire**.**

2. Expression du besoin et conception de solutions

Règle n°1 - Avant de programmer, fixer un cap

Il est important de **clarifier le problème** que l'on cherche à résoudre, le « **besoin** », en collaboration avec ceux qui l'ont posé (hiérarchie, client, co-auteur… voire soi-même !). En effet, des attentes peu claires sont souvent sources de travaux inutiles.

La tentation est toujours grande de commencer la programmation le plus rapidement possible. C'est rarement la solution la plus efficace. Il convient en effet d'identifier l'existant (N'existe-t-il pas déjà un programme qui fait ce que je veux ?) et les contraintes (Pour quand est attendu le résultat ? Ce travail devra-t-il être reconduit par la suite ?). Cela doit amener à réfléchir l'automatisation des traitements et le choix du (des) logiciel(s).

Dans la construction d'une solution à un problème, on gagne à séparer la conception (ce que l'on veut faire) de la réalisation (ce que l'on fait). Les travaux exploratoires peuvent être considérés comme des tâches de conception. Leur transposition dans la solution nécessite, sinon une réécriture intégrale, du moins une mise au propre.

Règle n°2 - Identifier les moyens et planifier

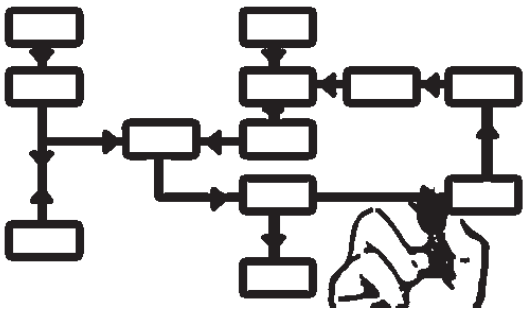
Une fois ce « besoin » exprimé, et avant de passer à la réalisation proprement dite, il faut **choisir une solution**. Ce choix peut s'appuyer sur des variantes de conception, qui modulent notamment, si des marges de manœuvre existent, le **temps** de réalisation, la **pérennité** de la solution et la **qualité** de la solution. Ensuite, pour chacune de ces variantes envisageables, on définit les moyens alloués à la réalisation de la solution : quelles personnes, avec quels savoir-faire, quels outils.

Prenons un exemple. Il nous est demandé de calculer l'évolution de la taille d'une population donnée. On se positionne entre deux extrêmes :

- Supposons que l'on doive donner la réponse dans la journée, que cette demande n'est pas vouée à être renouvelée et qu'une vision approximative suffit. On peut alors se contenter de faire les calculs dans un tableur, et sur des sources existantes et directement mobilisables ;

- À l'inverse, supposons que l'on ait du temps et que ce chiffre doive être produit régulièrement avec une qualité maximale. On choisira alors une solution automatisée, avec de nombreux contrôles automatiques et manuels, et sur une source *ad hoc*.

Entre ces deux extrêmes, **on se posera la question de la pertinence de l'automatisation de certains éléments de la solution et des outils utilisés. On réfléchira à la quantité et au mode de contrôles réalisés, ainsi qu'aux sources mobilisables ou à créer.**



Encadré n°1 - Bien se renseigner, bien s'outiller

Pour identifier l'existant et donc les contraintes d'un projet, le réflexe d'une recherche Internet est indispensable. Quelques mots clés (le nom d'une procédure, d'une méthode statistique…) suffisent bien souvent à trouver l'aide pertinente et à résoudre un problème.

Certains outils nous facilitent grandement la tâche. Ce sont par exemple des gestionnaires de versions ou des éditeurs améliorant l'ergonomie des logiciels ou la lisibilité des programmes, Il n'est bien sûr pas possible d'en fournir ici une liste exhaustive et à jour. Quelques programmes sont mentionnés en annexe à la date d'écriture de ce document. Pour disposer d'une liste à jour ou plus en adéquation avec les besoins d'un projet, il ne faut pas hésiter à effectuer une recherche Internet !

3. Construire un code lisible

Règle n°3 – Choisir des noms expressifs : renommer la table `CAA5012` pour la rendre plus explicite

Au-delà des aspects syntaxiques, il ne faut pas négliger la **sémantique**. Qu'il y ait des noms expressifs (de programmes, de variables, de tables) ou non, la machine ne voit pas la différence. Mais la personne qui lit le programme, elle, la voit ! Une table `CA_Annuel_1950_2012` est plus facile à appréhender que `CAA5012`. De même, on préférera parler de la variable `revenus` que de `var1234`.

Il faut éviter les noms imprononçables, dont la prononciation est proche, ou seulement différenciés par la casse (avec ou sans majuscules), et ne pas indexer arbitrairement les variables par des chiffres. Des labels (et des formats via la `PROC FORMAT` sous SAS) peuvent également être associés aux variables pour préciser la signification d'une variable et de ses modalités et améliorer la lisibilité des résultats statistiques.

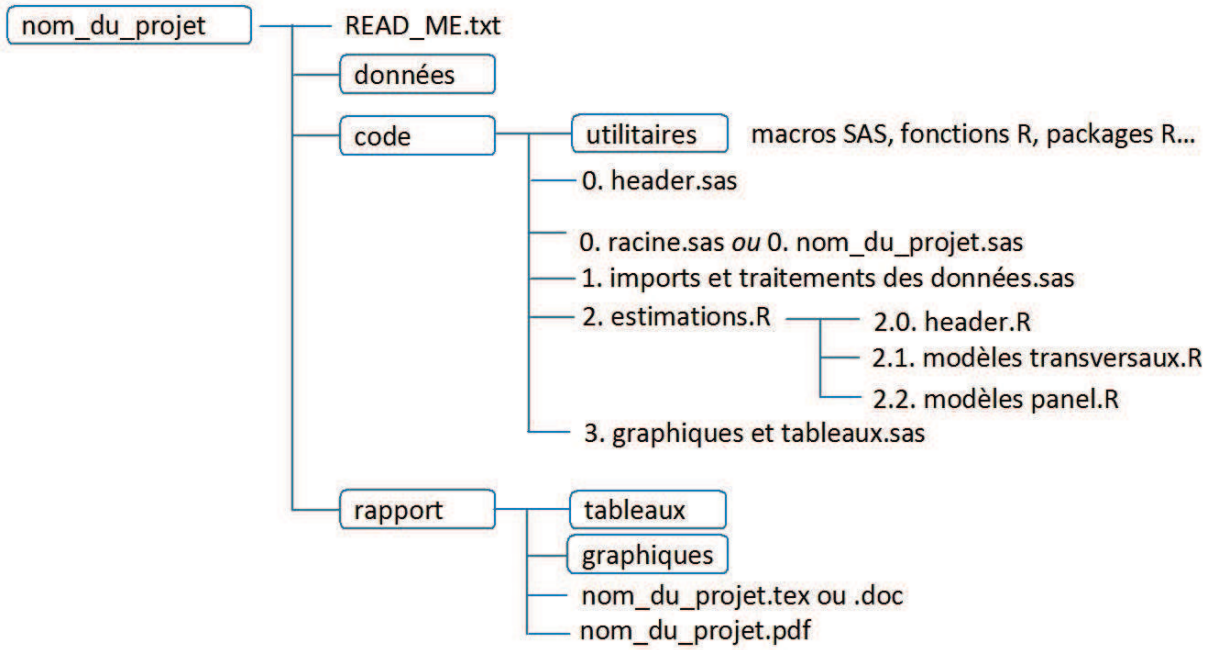
Règle n°4 - Des programmes courts et organisés : entrée-traitement-sortie

Lors d'un projet à orientation statistique ou économétrique, on manipule plusieurs types de fichiers :

- Des bases de données ;
- Des programmes ;
- Des sorties (tableaux, graphiques) ;
- Des documents mis en forme (.odt, .rtf, .doc, .pdf,...)

Il est nécessaire d'organiser ces différents fichiers, pour améliorer le travail collaboratif et pouvoir aisément retrouver un fichier plusieurs jours, voire plusieurs mois, après l'avoir créé. Il est généralement pertinent d'adopter une approche entrée-traitement-sortie. Plutôt que de longs programmes compliqués, **les différentes étapes sont alors scindées en sous-étapes de manière lisible** (cf. exemple du schéma 1)[2].

Schéma 1 : Exemple d'arborescence des fichiers



Un fichier racine lance l'ensemble des programmes à la volée[3]. Il permet de certifier la cohérence d'ensemble des programmes. En l'exécutant plusieurs fois de suite, les résultats, les tableaux et les graphiques en sortie, doivent rester identiques. Ainsi les tables initiales ne sont pas modifiées (des tables intermédiaires sont créées), les générateurs de nombres aléatoires prennent des paramètres fixes, etc.

[2]On présente ici un exemple assez simple et courant. Pour plus de généralité, on peut regarder le modèle GSBPM (General Statistical Business Process Model,). On peut aussi s'intéresser à un package de R qui propose de générer une arborescence en fonction de ses besoins : http://projecttemplate.net/architecture.html

[3] Pour lancer l'ensemble des programmes et générer l'intégralité des résultats, voir par exemple l'instruction `%INCLUDE` dans SAS, la fonction `source()` en R, et plus généralement les fichiers .bat. sous Windows et .sh sous Unix.

Le *header* (ou *en-tête* en français) peut servir à préciser les chemins d'accès et les variables globales (par exemple, le chemin du répertoire du projet).

La définition d'un sommaire (READ_ME.txt) détaillant l'**ordre chronologique des traitements** permet de garder une visibilité sur leur déroulement. Ce sommaire peut prendre une forme traditionnelle, avec des étapes numérotées. On peut aussi le représenter sous forme de schémas (cf. schéma 2 ci-dessous) qui permettent de visualiser l'enchaînement des programmes, ce qui peut être particulièrement utile pour des projets complexes de production statistique. Pour les études économiques, un schéma peut être réalisé en fin de projet dans une optique de faciliter la reproduction des analyses.

Encadré n°2 – Faire des schémas

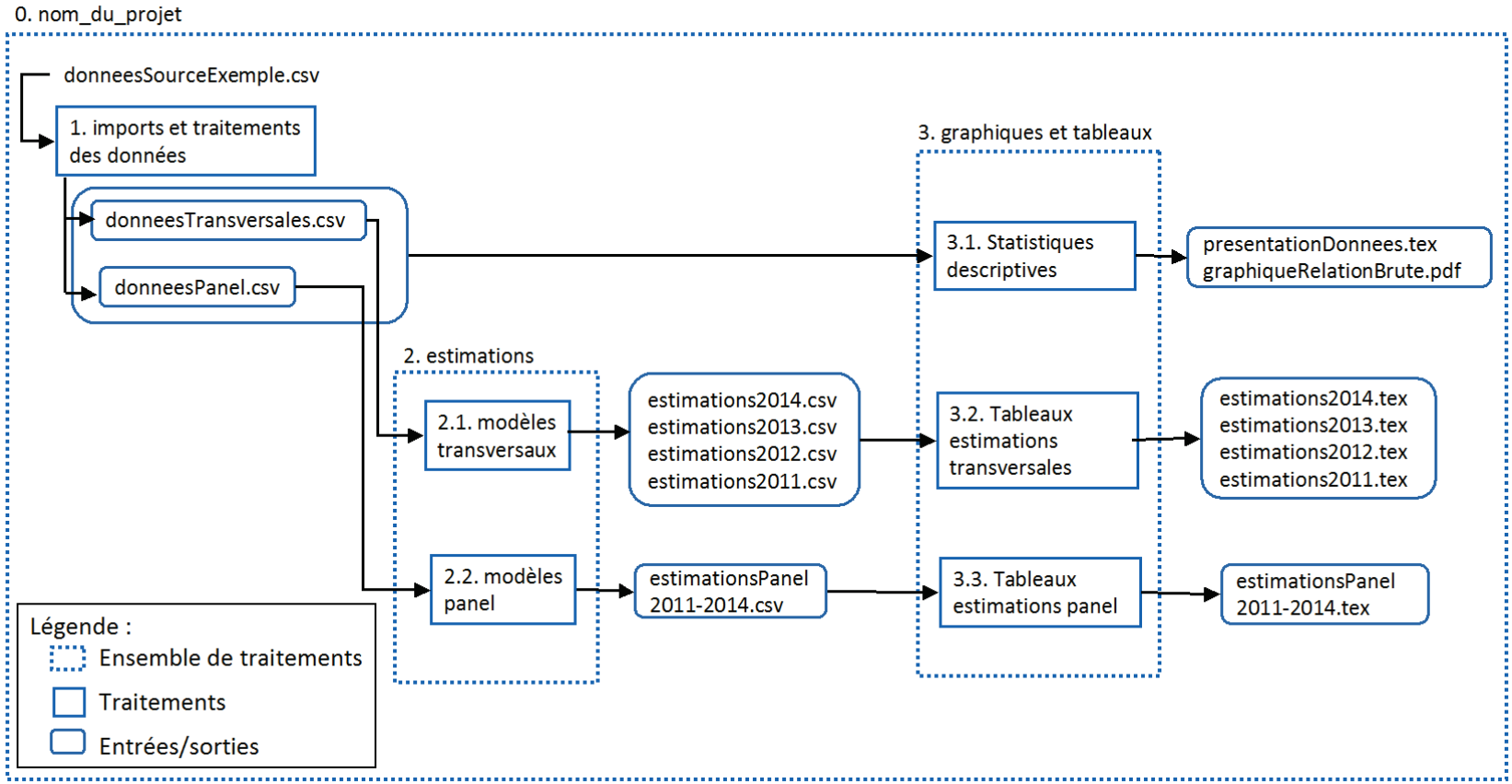
Pour **réaliser des schémas de processus**, plusieurs outils sont possibles. Ces outils permettent de manipuler les formes (flèches, cadres, textes…) plus facilement que sous un logiciel de dessin :

- Un **logiciel de présentation** (par exemple *LibreOffice Impress* ou *Microsoft PowerPoint*) est très facile d'utilisation. De plus, il contraint le schéma à être de taille réduite, donc synthétique.

- Si l'on cherche absolument à présenter de nombreuses informations sur un seul schéma, et donc à ne pas être limité sur la taille du schéma, on peut utiliser par exemple un **tableur**. On gagne néanmoins souvent à structurer et scinder de tels schémas, pour qu'ils ne soient pas trop complexes à appréhender.

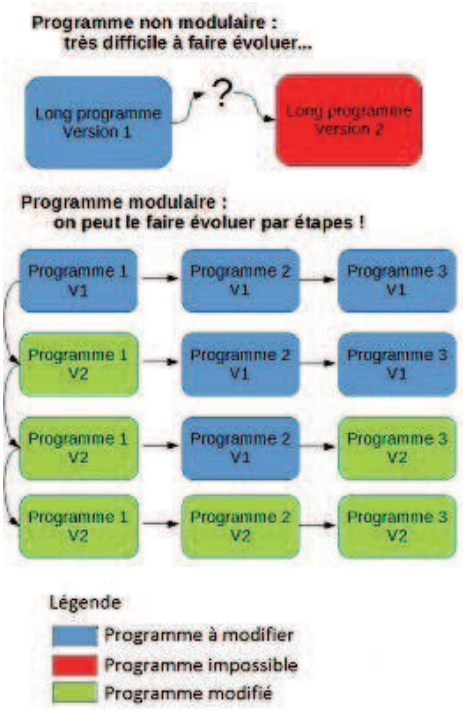
- Des outils **UML** (*Unified Modeling Language*), utilisés en informatique pour le développement logiciel, sont également dédiés à la réalisation de schémas.

Schéma 2 : Exemple de schéma d'enchaînement des traitements (même projet que le schéma 1)



Les programmes ne doivent pas être trop longs. **L'organisation du code doit avoir du sens** (enchaînement des procédures, types de calculs, types de sorties…). Se contraindre à faire des programmes courts compense la faible ergonomie des logiciels (naviguer dans un code SAS ou R de plusieurs centaines de lignes est laborieux). De plus, cela rend les traitements "modulaires", et donc plus faciles à maintenir (cf. schéma 3 ci-contre).

Schéma 3 : Pérennité d'un code modulaire



Règle n°5 - Trouver un style, c'est important, et le garder, c'est primordial

Dans la plupart des langages informatiques répandus, les éditeurs offrent des outils avancés de mise en forme automatique. Les éditeurs de logiciels statistiques sont souvent assez pauvres de ce point de vue. Il faut donc se contraindre à respecter certaines normes. Il n'y a néanmoins pas de normes universelles. Il existe plusieurs styles possibles, mais il faut en choisir un, et le suivre.

Mieux vaut un code plus long mais explicite

Le code est sous forme électronique, et n'est pas destiné à être imprimé. On peut donc aérer et structurer le code par de l'**indentation** (tabulations et retour à la ligne), il n'y aura pas de gaspillage de papier ! Il existe même des utilitaires qui reformatent le code.

MAJUSCULES, minuscules ou autresChoix ?

Quand on utilise le macro langage SAS, indispensable pour automatiser des traitements SAS, on perd l'affichage coloré du code, pourtant bien pratique pour différencier les mots de codes des variables. Il faut donc s'imposer d'écrire les mots de code SAS en majuscules : `PROC`, `DATA`, `BY`, `VAR`, `IF`…

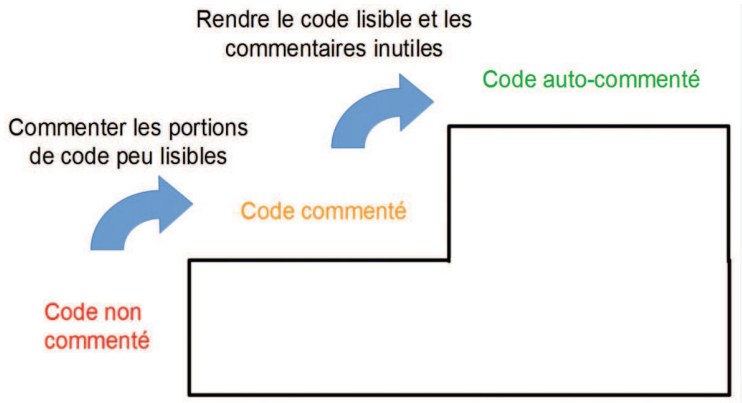
Une convention courante est alors d'écrire les variables en minuscules, par exemple en `lowerCamelCase` (majuscule pour la première lettre des mots, sauf le premier mot) ou en `snake_case` (mots en minuscules séparés par des tirets bas). Les constantes peuvent être écrites en `SCREAMING_SNAKE_CASE` (mots en majuscules séparés par des tirets bas).

Règle n°6 - Mon commentaire est-il utile ?

Ah vous commentez votre code ! C'est déjà mieux que rien… Il est en effet souvent utile d'inclure un commentaire minimal, qui explique à quoi sert le programme, quelle table il appelle et quelle table il donne en sortie. **Mais on peut généralement faire mieux que laisser un commentaire.** Quand on cherche à commenter un programme,

on veut apporter une information importante qui n'est pas présente dans le code. Mais a-t-on convenablement structuré son code, ou bien a-t-on utilisé des noms de variables/fonctions assez expressifs ?

Schéma 4 : Utiliser les commentaires pour clarifier le code



On ne peut que louer la volonté du programmeur d'éviter toute ambiguïté dans son code. Mais essayer d'être clair uniquement avec des commentaires est une mauvaise solution. En effet, un code est destiné à « vivre », les commentaires deviendront donc faux. Il est en effet difficile de maintenir à la fois le code lui-même et ses commentaires.

On peut commencer par écrire le commentaire, puis voir si l'information qu'il comporte ne pourrait pas être portée par la structuration du code ou la sémantique (cf. schéma 4).

Le commentaire ne se substitue pas à une bonne documentation. La base de données ne doit ainsi pas être décrite dans les programmes mais à l'aide d'un dictionnaire spécifique. Pour des fonctions complexes avec de nombreux arguments, il est nécessaire de rédiger une documentation qui détaille chaque paramètre et d'illustrer son fonctionnement à l'aide d'exemples (c'est le cas des macros SAS INSEE d'analyse de données, mais également des packages complémentaires de R ou Stata ; ce ne sera pas le cas pour les programmes d'une étude économique).

Parfois il est difficile de faire autrement que de laisser un commentaire. C'est en particulier le cas lorsqu'une **instruction**, voire une série d'instructions :

- est **complexe et difficile à simplifier** par exemple :
    - parce qu'on n'a pas le temps ;
    - ou bien parce qu'elle a été optimisée pour des raisons motivées de performance (voir encadré n°3);
- ou bien est **peu intuitive** et difficile à clarifier directement dans le code.

D'autres fois cependant, on a l'impression qu'un commentaire est judicieux, alors qu'il est possible de s'en passer : il vaut mieux décomposer les différentes étapes d'un programme, en utilisant des fonctions dont les noms sont judicieusement choisis, afin d'obtenir un code auto-commenté, comme illustré dans l'exemple suivant.

Exemple de code auto-commenté *vs* commentaire

Dans le code suivant :

```
/*Édition du tableau de synthèse des estimations*/
concatenerLignes(lignes) ;
```

le commentaire précise l'**intention**, le « pourquoi », ce que le nom actuel de la fonction ne peut pas faire, parce qu'il se focalise sur le « comment ».

Dans ce cas, on gagnerait à introduire une nouvelle fonction `editionTableauDeSynthese`, qui se contenterait d'appeler `concatenerLignes`, et à renommer `lignes` en `estimations`[4] :

```
editionTableauDeSynthese(estimations) ;
```

[4] Les aficionados de SAS auraient plutôt réécrit l'appel de `%concatenerLignes(&lignes)` en un appel de `%editionTableauDeSynthese(&estimations)`

Avec l'usage, la fonction `editionTableauDeSynthese` pourra ainsi s'étoffer et inclure d'autres fonctions que `concatenerLignes`.

Un autre exemple où l'utilisation de commentaires peut se discuter est lorsque l'on subit un **héritage** (un programme ou des données élaborées par une autre personne), que l'on ne cherche pas à masquer. Par exemple, si l'on récupère des variables aux **noms imprononçables** d'une base de données existante, il pourrait paraître légitime de conserver ce nommage, pour être transparent sur les données que l'on manipule. Cependant, on gagnera à les renommer dans son code, par exemple par l'intermédiaire d'une table de passage qui pourra aussi servir de dictionnaire des variables.

Règle n°7 - Se faire relire

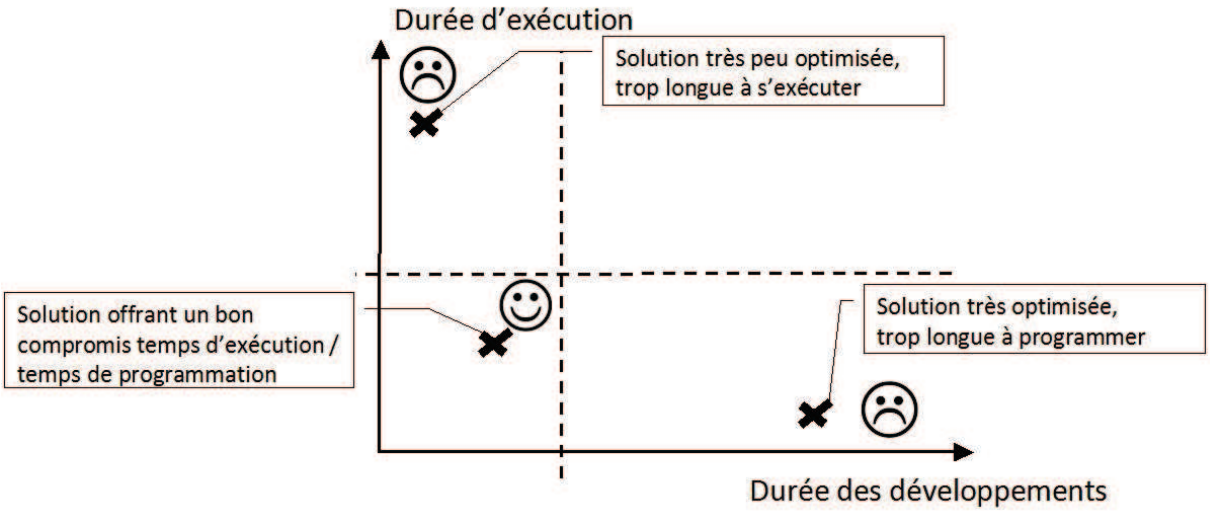
Tout le monde fait des erreurs ou a des mauvaises habitudes, même parmi les programmeurs les plus habiles. Quoi de mieux, pour tester la lisibilité d'un programme, que de le faire relire par un tiers ?

Une étude économique ou méthodologique fera l'objet de nombreuses relectures, notamment par la hiérarchie. C'est très rarement le cas des programmes qui ont servi à réaliser l'étude. Les études économiques se font souvent à plusieurs. Même si une répartition s'est faite entre celui qui programme et celui qui conçoit (ou selon les parties de l'étude), il est indispensable que chacun des auteurs s'assure par exemple du caractère reproductible de l'ensemble des analyses. De surcroît, cela permet souvent d'identifier des erreurs.

Encadré n°3 - Optimiser, est-ce bien nécessaire ?

Optimiser son code, c'est modifier un programme qui fonctionne pour qu'il s'exécute plus rapidement. **L'optimisation est la plupart du temps inutile**. En effet, la quête d'un code optimisé peut surtout se révéler être chronophage, sans réels gains (cf. schéma 5). Si les règles présentées ici sont déjà prises en compte, le code sera à la fois simple, lisible et évolutif.

Schéma 5 : Motiver l'optimisation du code par une baisse sensible de la durée d'exécution



Un programme mis à disposition de manière large, et qui sera donc potentiellement exécuté de très nombreuses fois, pourra justifier une plus longue durée des développements à des fins d'optimisation.

Pour des simulations ou l'utilisation de données massives, les temps d'exécution peuvent être très longs. Des algorithmes plus performants permettent alors des gains sensibles de temps de calcul. Mais une manière simple de gagner de nombreuses (et précieuses) heures est également de tester son programme sur un échantillon de taille réduite, plutôt que de constater son échec après plusieurs heures d'exécution.

Règle n°8 - Débugger c’est bien, tester c’est encore mieux

Débugger un programme, c’est obtenir un programme qui s’exécute sans erreurs. Mais cela ne veut pas forcément **dire que le programme fait ce qui est attendu de lui**. Le vérifier correspond à l’étape de tests. Mettre en place des tests de vérifications intermédiaires évite de rechercher la source de l’erreur tardivement, ce qui peut être très coûteux.

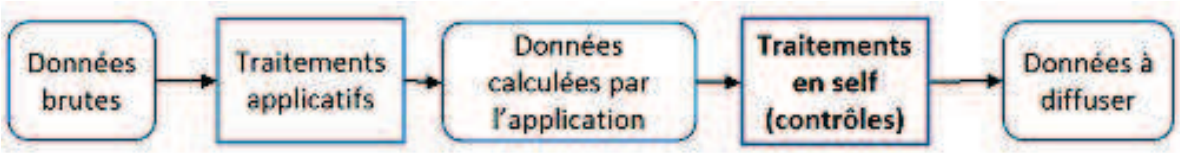
Pour débugger et tester un programme, on peut établir la chronologie suivante :

- Après exécution, regarder le journal pour identifier les éventuelles erreurs.
- Après correction de ces erreurs à l’exécution, vérifier les résultats pour savoir s’ils ne sont pas aberrants. Ces tests gagnent souvent à être automatisés.

Un exemple particulier est le cas du transfert de données entre différents logiciels. Une vérification exhaustive du résultat n’est pas possible. Néanmoins, il est possible de vérifier différents types de variables et plusieurs observations. La gestion des données manquantes, parfois notées par une lettre, peut par exemple produire à tort de mauvais formats de variables.

Un autre exemple est celui du **contrôle des données** issues de traitements applicatifs, avant diffusion des résultats (cf. schéma 6 ci-dessous). Les traitements applicatifs sont spécifiés par un statisticien mais programmés par un informaticien. Dans le cadre d’une enquête statistique, cela pourra par exemple correspondre à des redressements automatisables ou la gestion de la codification des variables. La vérification de la cohérence entre les spécifications (i.e. le cahier des charges) et le résultat est une phase de tests spécifiques, nommée *recette*. L’ensemble des fonctionnalités sont alors testées une à une. Les données issues de tels traitements ne sont souvent pas directement diffusables. D’autres contrôles sont spécifiés et programmés par le statisticien (traitements dits en self), par exemple pour la gestion du secret statistique, la cohérence des résultats d’une enquête répétée dans le temps ou à partir des questions des diffuseurs ou d’autres utilisateurs. Ils **peuvent s’accumuler** et **renforcent alors la robustesse du processus**.

Schéma 6 : Exemple d'automatisation de contrôles



Références indicatives

McConnell Steve (2005), *Tout sur le Code*, Microsoft Press

Martin Robert C. (2009), *Coder proprement*, Pearson

Courrier des statistiques (2009), *Numéro Hors-série : Savoir compter, savoir conter*

Annexe : Quelques outils utiles (Janvier 2016)

Gestion des versions

Il est très utile de versionner ses fichiers. On peut ainsi le faire manuellement, par numéro (0.1,0.2,1.0, etc.) ou par date. Mais pour des projets avec beaucoup de fichiers, cela peut devenir fastidieux. Il existe des applicatifs dédiés à ces questions, **Git** ou **TortoiseSVN** par exemple. Cette solution sera plus efficace que des en-têtes de programmes précisant la version ou les modifications effectuées, qui sont difficiles à mettre à jour de manière dynamique et peuvent se périmer.

GitHub permet de plus d'effectuer cette gestion des versions pour l'ensemble des documents d'un projet collaboratif, tout en autorisant un travail simultané sur de mêmes fichiers.

Éditeurs

- **RStudio** est un environnement de développement spécialement dédié à R,
- **Notepad++** est un éditeur de code source, plus riche que le simple bloc-notes,
- Pour les plus courageux, un coup d'œil à **emacs** s'impose, en particulier son module *Emacs Speaks Statistics* (ESS).

Outils divers

- **Grep** permet de rechercher une chaîne de caractère dans un ensemble de fichiers, par exemple pour trouver toutes les utilisations qui sont faites d'une fonction R ou d'une macro SAS. Pour aller plus loin on pourra s'intéresser à l'ensemble des outils Unix, en particulier **sed** et **awk**,
- **DocItOut** permet de générer automatiquement de la documentation pour SAS suivant le même principe que la Javadoc,
- Les packages **Sweave** ou **R markdown** permettent d'inclure des tableaux et graphiques interactifs générés sous R dans des présentations ou des rapports. Ceux-ci sont automatiquement mis à jour en cas de modifications du code ou des données.
- **Cacoo** pour faire des schémas.

Série des Documents de Travail « Méthodologie Statistique »

9601 : Une méthode synthétique, robuste et efficace pour réaliser des estimations locales de population.
G. DECAUDIN, J.-C. LABAT

9602 : Estimation de la précision d'un solde dans les enquêtes de conjoncture auprès des entreprises.
N. CARON, P. RAVALET, O. SAUTORY

9603 : La procédure FREQ de SAS - Tests d'indépendance et mesures d'association dans un tableau de contingence.
J. CONFAIS, Y. GRELET, M. LE GUEN

9604 : Les principales techniques de correction de la non-réponse et les modèles associés.
N. CARON

9605 : L'estimation du taux d'évolution des dépenses d'équipement dans l'enquête de conjoncture : analyse et voies d'amélioration.
P. RAVALET

9606 : L'économétrie et l'étude des comportements. Présentation et mise en œuvre de modèles de régression qualitatifs. Les modèles univariés à résidus logistiques ou normaux (LOGIT, PROBIT).
S. LOLLIVIER, M. MARPSAT, D. VERGER

9607 : Enquêtes régionales sur les déplacements des ménages : l'expérience de Rhône-Alpes.
N. CARON, D. LE BLANC

9701 : Une bonne petite enquête vaut-elle mieux qu'un mauvais recensement ?
J.-C. DEVILLE

9702 : Modèles univariés et modèles de durée sur données individuelles.
S. LOLLIVIER

9703 : Comparaison de deux estimateurs par le ratio stratifiés et application aux enquêtes auprès des entreprises.
N. CARON, J.-C. DEVILLE

9704 : La faisabilité d'une enquête auprès des ménages.
1. au mois d'août.
2. à un rythme hebdomadaire

C. LAGARENNE, C. THIESSET

9705 : Méthodologie de l'enquête sur les déplacements dans l'agglomération toulousaine.
P. GIRARD.

9801 : Les logiciels de désaisonnalisation TRAMO & SEATS : philosophie, principes et mise en œuvre sous SAS.
K. ATTAL-TOUBERT, D. LADIRAY

9802 : Estimation de variance pour des statistiques complexes : technique des résidus et de linéarisation.
J.-C. DEVILLE

9803 : Pour essayer d'en finir avec l'individu Kish.
J.-C. DEVILLE

9804 : Une nouvelle (encore une !) méthode de tirage à probabilités inégales.
J.-C. DEVILLE

9805 : Variance et estimation de variance en cas d'erreurs de mesure non corrélées ou de l'intrusion d'un individu Kish.
J.-C. DEVILLE

9806 : Estimation de précision de données issues d'enquêtes : document méthodologique sur le logiciel POULPE.
N. CARON, J.-C. DEVILLE, O. SAUTORY

9807 : Estimation de données régionales à l'aide de techniques d'analyse multidimentionnelle.
K. ATTAL-TOUBERT, O. SAUTORY

9808 : Matrices de mobilité et calcul de la précision associée.
N. CARON, C. CHAMBAZ

9809 : Échantillonnage et stratification : une étude empirique des gains de précision.
J. LE GUENNEC

9810 : Le Kish : les problèmes de réalisation du tirage et de son extrapolation.
C. BERTHIER, N. CARON, B. NEROS

9901 : Perte de précision liée au tirage d'un ou plusieurs individus Kish.
N. CARON

9902 : Estimation de variance en présence de données imputées : un exemple à partir de l'enquête Panel Européen.
N. CARON

0001 : L'économétrie et l'étude des comportements. Présentation et mise en œuvre de modèles de régression qualitatifs. Les modèles univariés à résidus logistiques ou normaux (LOGIT, PROBIT) (version actualisée).
S. LOLLIVIER, M. MARPSAT, D. VERGER

0002 : Modèles structurels et variables explicatives endogènes.
J.-M. ROBIN

0003 : L'enquête 1997-1998 sur le devenir des personnes sorties du RMI - Une présentation de son déroulement.
D. ENEAU, D. GUILLEMOT

0004 : Plus d'amis, plus proches ? Essai de comparaison de deux enquêtes peu comparables.
O. GODECHOT

0005 : Estimation dans les enquêtes répétées : application à l'Enquête Emploi en Continu.
N. CARON, P. RAVALET

0006 : Non-parametric approach to the cost-of-living index.
F. MAGNIEN, J. POUGNARD

0101 : Diverses macros SAS : Analyse exploratoire des données, Analyse des séries temporelles.
D. LADIRAY

0102 : Économétrie linéaire des panels : une introduction.
T. MAGNAC

0201 : Application des méthodes de calages à l'enquête EAE-Commerce.
N. CARON

C 0201 : Comportement face au risque et à l'avenir et accumulation patrimoniale - Bilan d'une expérimentation.
L. ARRONDEL, A. MASSON, D. VERGER

C 0202 : Enquête Méthodologique Information et Vie Quotidienne - Tome 1 : bilan du test 1, novembre 2002.
J.-A. VALLET, G. BONNET, J.-C. EMIN, J. LEVASSEUR, T. ROCHER, P. VRIGNAUD, X. D'HAULTFOEUILLE, F. MURAT, D. VERGER, P. ZAMORA

0203 : General principles for data editing in business surveys and how to optimise it.
P. RIVIERE

0301 : Les modèles logit polytomiques non ordonnés : théories et applications.
C. AFSA ESSAFI

0401 : Enquête sur le patrimoine des ménages - Synthèse des entretiens monographiques.
V. COHEN, C. DEMMER

0402 : La macro SAS CUBE d'échantillonnage équilibré
S. ROUSSEAU, F. TARDIEU

0501 : Correction de la non-réponse et calage de l'enquêtes Santé 2002
N. CARON, S. ROUSSEAU

0502 : Correction de la non-réponse par répondération et par imputation
N. CARON

0503 : Introduction à la pratique des indices statistiques - notes de cours
J-P BERTHIER

0601 : La difficile mesure des pratiques dans le domaine du sport et de la culture - bilan d'une opération méthodologique
C. LANDRE, D. VERGER

0801 : Rapport du groupe de réflexion sur la qualité des enquêtes auprès des ménages
D. VERGER

M2013/01 : La régression quantile en pratique
P. GIVORD, X. D'HAULTFOEUILLE

M2014/01 : La microsimulation dynamique : principes généraux et exemples en langage R
D. BLANCHET

M2015/01 : la collecte multimode et le paradigme de l'erreur d'enquête totale
T. RAZAFINDROVONA

M2015/02 : Les méthodes de Pseudo-Panel
M. GUILLERM

M2015/03 : Les méthodes d'estimation de la précision pour les enquêtes ménages de l'Insee tirées dans Octopusse
E. GROS – K.MOUSSALAM

M2016/01 : Le modèle Logit Théorie et application.
C. AFSA

M2016/02 : Les méthodes d'estimation de la précision de l'Enquête Emploi en Continu
E. GROS – K.MOUSSALAM

M2016/03 : Exploitation de l'enquête expérimentale Vols, violence et sécurité.
T. RAZAFINDROVONA

M2016/04 : Savoir compter, savoir coder. Bonnes pratiques du statisticien en programmation.
E. L'HOUR – R. LE SAOUT
B. ROUPPERT