Les tests automatisés en R

Comment maintenir son code

David Beauchemin, BSc. et
Christopher Blier-Wong, MSc.

.Layer, Université Laval, CRDM, GRAAL

14 mai 2019, R à Québec 2019

Agenda

Qui nous sommes

- Étudiants avec intérêt commun à l'actuariat, l'informatique et l'intelligence artificielle
- Membres de .Layer, est une communauté ayant comme mission de promouvoir la collaboration et le partage de connaissances dans le domaine de la science des données.
- On organise des conférences, des ateliers et autres événements.

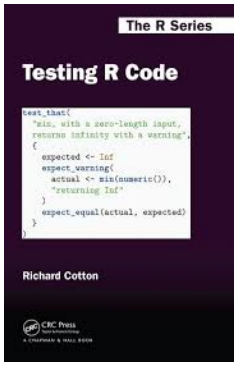
dotlayer.org

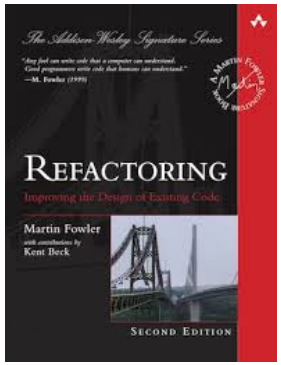
: MeetupMLQuebec

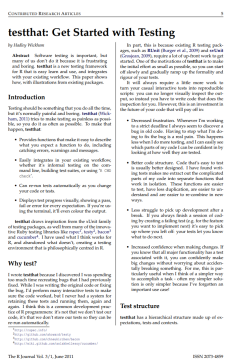
Quelques considérations

- Quand on dit ”les autres” dans cette présentation, ça veut aussi dire vous dans le futur !
- On suit le style de code Google’s R Style Guide. Le livre Advanced R en offre aussi un.
- Il y a différentes philosophies de test pour chaque paradigme de programmation. Dans cet atelier, on considère le R comme un langage procédural contrairement à fonctionnel

Quelques considérations






testthat: Get Started with Testing

Le *clean code*

- Le test unitaire est une pratique du *clean code*
- Le *clean code* est une pratique de génie informatique qui dicte que le code doit être propre.

> “Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”
>
> — Martin Fowler

Le *clean code*

Mais quand on compile, ça n'a pas d'importance si le code est propre
Un code propre permet de collaborer facilement et éviter des bogues.

- Quand on écrit du code, on sait (dans notre tête) ce que les variables représentent
- Ward Cunningham (fondateur, wiki) : on doit passer cette compréhension de la tête au code lui-même.
- Pour que les autres comprennent ce que vous faires sans avoir à lire les assignations.

Le *clean code*

Quelques principes qui s'appliquent en R :

- Nommer une variable est une tâche très importante.
- Une fonction devrait avoir une seule tâche.
- Les commentaires sont dangereux.
  - Si vous en avez besoin, le code n'est pas assez claire.
  - On oublie souvent de les mettre à jour.
- Le contrôle de version est votre ami.

Assurance qualité, R et calcul scientifique, demain @ 13:55

Les tests pour les scientifiques des données

En R, les principales tâches sont :

- D'interagir avec des données afin de découvrir des statistiques ou de l'information intéressante (exploration)
- D'automatiser ces tâches d'exploration sur plusieurs jeux de données ou sur plusieurs tâches (programmation)
- De rassembler une collection de fonctions et publier un paquetage pour partager aux autres (partage)

Les tests pour les scientifiques des données

Exploration

Si les tâches d'exploration

- sont quelques lignes ;
- sont simples ;
- font seulement appel à des fonctions pré-définies ;

les tests ne sont pas nécessaires.

Les tests pour les scientifiques des données

Programmation

On désire des tests dans les tâches de programmation pour

- s'assurer que notre fonction a le comportement voulu
- s'assurer qu'on ne brise pas quelque chose qui fonctionnait dans le passé

Les tests pendant la programmation servent à vérifier que le programmeur n'a pas fait d'erreur.

Les tests pour les scientifiques des données

Partage

On désire des tests dans les tâches de partage pour

- s'assurer que les utilisateurs respectent les conditions d'utilisation de votre fonction.
- important quand on publie des paquetages.

Les tests pendant le partage servent à vérifier que l'utilisateur n'ai pas fait d'erreur

Deux principales philosophies de tests

Le R compte deux grandes philosophies de tests.

- Les tests pour s'aider
- Les tests pour aider les autres

> "The point of development-time testing is to make sure that you haven't done something stupid. By contrast, the point of run-time testing is to make sure that the user hasn't done something stupid."
>
> — Richard Cotton

Deux principales philosophies de tests

Les tests pour s'aider

Les tests lors du développement

- Ce sont les tests unitaires.
- On test le résultat d'une fonction.
- On exécute le test pour s'assurer qu'on a pas introduit de bogues.
- Rendu très simple avec testthat
- C'est l'objectif de l'atelier.

Deux principales philosophies de tests

Les tests pour aider les autres

Les tests lors du partage

- Quand on écrit une fonction, on a une utilité en tête
- D'autres personnes vont peut-être vouloir l'utiliser pour une tâche différente : les gens sont créatifs !
- Paquetage assertive.

Quand est-ce que je dois écrire mon test ?

À quel moment est-ce que le test devrait être écrit ?

1. Après avoir écrit un script
2. Avant avoir écrit la fonction (partie 3)

Quand est-ce que je dois écrire mon test ?

Situation 1

On considère la situation suivante :

1. On définit une tâche
2. On se donne des paramètres jouets
3. On constuit un script pour donner la réponse voulue
4. On extrait une fonction du script
5. On efface le script

On perd une information importante : une condition que la fonction doit respecter !

Quand est-ce que je dois écrire mon test ?

Situation 1

À la place, on

1. Conserve le résultat attendu dans un test
2. C'est le cas le plus facile, on a déjà tous les ingrédients !

Quand est-ce que je dois écrire mon test ?

Situation 2

On considère la situation suivante :

1. On regarde un script qu'on a fait il y a longtemps
2. On veut utiliser le code dans une autre tâche

Quand est-ce que je dois écrire mon test ?

Situation 2

Alors, on

1. On se définit des conditions que la fonction doit respecter (tests)
2. On écrit la fonction de telle sorte que les tests passent

On s'assure de ne rien briser !

Quand est-ce que je dois écrire mon test ?

Situation 3

On considère la situation suivante :

1. On regarde une fonction
2. Elle contient plusieurs comportements, cas et conditions
3. On désire ajouter un comportement

La fonction est possiblement utilisée à d'autres endroits, il ne faut pas changer ce qu'elle fait, seulement comment elle le fait.

Quand est-ce que je dois écrire mon test ?

Situation 3

Alors, on

1. Écrit des tests pour valider le comportement du code total
2. On extrait des fonctions individuelles pour chaque comportement, cas et conditions
3. On exécute les tests à chaque changement

Quand est-ce que je dois écrire mon test ?

Après avoir écrit un script

Situation 3 : quand on a mal programmé

Fail fast, fail often

- Plus souvent on test, plus il est façile de diagnostiquer le problème
-

Agenda

Un premier test unitaire

Le compromis complexité-rapidité

Faire des exemples de tests dans un atelier est difficile.

- Si les programmes vallent la peine de faire des tests, ils sont trop long à expliquer.
- Si les programmes sont trop simples, ça ne vaut pas la peine de faire tests.

Tester son code est simple. La difficulté est développer l'habitude.

Un premier test unitaire

Avant d'utiliser Testthat, on écrit notre propre test à la main.

La moyenne géométrique

`partie_0/exemple_1.R`

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}$$

Agenda

L'interface générale d'un test

Un test *TestThat* se divise en deux parties :

- Le nom du comportement tester
- Le code du test qui se sous-divise en deux parties :
  - La valeur retournée par le comportement tester (*optionnel selon la situation*)
  - L'évaluation du comportement tester

Exemple *TestThat*

Scénario : Calculer une valeur logarithmique

Une valeur positive,
application de la transformation logarithmique naturel,
retourne la valeur transformer.

Exemple *TestThat*

Scénario : Calculer une valeur logarithmique

```
1 UNE_VALEUR_POSITIVE <- 1
2 VALEUR_ATTENDU <- 0
3
4 test_that("Une valeur positive,
5 transformation ln,
6 valeur logarithmique naturel",
7 {actual <- log(UNE_VALEUR_POSITIVE)
8 expect_equal(expected = VALEUR_ATTENDU, actual)
9 })
```

Exercices *TestThat*

Scénario : Moyenne .....

`introduction.R`

Pause

10 minutes

Le workflow Testthat

Rappel : un test avec Testthat est composé

1. d'un entête en caractères pour décrire le test
2. du test, qui est composé
    1. d'un appel à une fonction
    2. d'un résultat désiré
    3. d'une comparaison entre l'appel et le résultat.

Les tests automatisés en R

- Exécuter les tests doit être facile et rapide
- On doit pouvour exécuter tous les tests quand on applique un changement
- testthat facilite cette tâche mais on doit suivre un schéma particulier

La structure de tests

La structure de fichiers

- Créer un répertoire tests/
- Créer un répertoire tests/testthat
- Créer un fichier tests/testthat.R

Les tests automatisés en R

- Un fichier de tests est placé dans tests/ et débute par test
- Les tests sont organisés par hierarchie : les résultats attendus sont regroupés dans les tests, qui sont regroupés dans des fichiers.

Les tests automatisés en R

- Un résultat attendu décrit le résultat d'une computation. Il vérifie que le comportement est approprié
- Un test regroupe plusieurs résultats attendus pour une seule fonction / une seule fonctionnalité. Parfois appelé test unitaire.
- Un fichier regroupe plusieurs tests similaires ou reliés.

Les résultats attendus

- Un résultat attendu est le plus petit niveau de test.
- Il commence par expect_
- Il contient deux arguments : le résultat actuel, et le résultat attendu
- Les deux arguments doivent correspondre.

Les résultats attendus

Les expect_

- expect_equal
- expect_identical
- expect_match
- expect_output
- expect_error
- expect_warning
- expect_is
- expect_true et expect_false

Les résultats attendus

Les expect_

```
ls("package:testthat", pattern = "^expect")
```

Les résultats attendus

Qu'est-ce qui est égal ?

- 1e-50 = 0 ?
- Vrai / Attendu < ε ?
- |Vrai - Attendu| < ε ?

Les tests

- Chaque test devrait avoir un nom informatif qui explique ce qui se passe
- Quand le test échoue, c'est un des messages qu'on affiche
- Le message doit permettre d'identifier l'erreur rapidement
- Pensez : Tester que donné ..., quand ..., alors ...

Les tests

- On définit un nouveau test avec test_that
- Un test devrait tester un comportement

Organisation des tests

Les contextes

- On peut avoir un fichier par test
- On peut avoir un fichier avec tous les tests
- On cherche un compromis
- On peut diviser un fichier en contextes.

Exemple de contexte

Exécuter les tests

- `test file` exécute tous les tests dans un fichier
- `test_dir` exécute tous les fichiers dans un répertoire
- `test check` si vous développez des paquetages

Exécuter les tests

Exemple de output

Exécuter les tests

Changer les rapports

- `test_file("test-file.R", reporter = "summary")`
- `"minimal"` : une ligne
- `"stop"` : arrêt à l'échec
- `"silent"` : aucun, mais retourne une
- `"rstudio"` : Entre summary et minimal : une ligne par échec
- `"tap"` : "Test Anything Protocol"
- `"check"` : Pour les paquetages

Quoi tester

Quand vous êtes temptés d'écrire quelquechose sous un print, écrire un test à la place. – Martin Fowler

- Comportements attendus.
- Exemples négatifs.
- Fail fast, fail often : c'est façile faire des erreurs. Les tests sont des garanties et des

Plus d'information sur les tests

- Quelle valeur a fait échouer la fonction ?
- Quel itération de la boucle a échoué ?

```
test_that(
  "temp, with a NULL input, returns NULL",
  {
    actual <- temp(3, NULL)
    info <- paste("temp(3, NULL) =", deparse(actual))
    expect_null(actual, info = info)
  }
)
```

Pause

Pause de 30 minutes

Le workflow Testthat

Rappel : un test avec Testthat est composé

1. d'un entête en caractères pour décrire le test
2. du test, qui est composé
    1. d'un appel à une fonction
    2. d'un résultat désiré
    3. d'une comparaison entre l'appel et le résultat.

Le workflow Testthat

La hiérarchie de tests

1. Résultats attendus dans les tests
2. Tests dans les contextes
3. Contextes dans un fichier
4. Fichiers de tests
5. Répertoires de fichiers

Agenda

Bien écrire nos tests

Un test se divise en trois parties :

- Étant donner (*given*)
- Quand (*when*)
- Alors (*then*)

Cette segmentation permet une approche par exemple de comportement.

Given

Il s'agit de notre état avant le début du comportement du scénario que l'on test.

Scénario : Calculer une valeur logarithmique

Given Une valeur positive.

When

Il s'agit du comportement à tester de notre scénario que l'on test.

Scénario : Calculer une valeur logarithmique

When Application de la transformation logarithmique naturel.

Then

Il s'agit de la résultante de la transformation sur notre état initial.

Scénario : Calculer une valeur logarithmique

Then Retourne la valeur transformer.

Bien utiliser nos tests

- Documentation par comportement.
- Meilleure confiance lors d'intégration des composantes dans une architecture plus complexe.
- Rapidité de développement.

Bien utiliser nos tests

- Documentation par comportement.
- Meilleure confiance lors d'intégration des composantes dans une architecture plus complexe.
- Rapidité de développement.

Testability

> “Ideally, development-time tests should be written once and run lots of times.”
>
> — Richard Cotton

Tell don't ask en R

Laisser la propriété des données aux objets propriétaire.

> "[...] rather than asking an object for data and acting on that data, we should instead tell an object what to do. This encourages to move behavior into an object to go with the data."
>
> — Martin Fowler

Refactoring

Il s'agit d'un processus en continu !

> "A change made to the internal structure of [your code] to make it easier to understand and cheaper to modify without changing its observable behavior."
>
> — Martin Fowler

Boy scout rule

Nous sommes mutuellement responsable de la qualité du code.

> “Always leave the code you’re editing a little better than you found it.”
>
> — Robert C. Martin