Traitements statistiques et programmation avec R

Gilles Hunault
Maître de conférences à l'université d'Angers

DUNOD

Conception de la couverture : Hokus Pokus

Illustration de couverture : © mk. / Fotolia

© Dunod, Paris, 2017
11, rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-075889-0

Table des matières

Remerciements VII

Avant-propos IX

Chapitre 1 Principes généraux de programmation 1

1. Qu'est-ce que la programmation R ? 1

2. Comment apprendre à programmer ? 2

3. Principes de programmation 3

4. Spécificités du langage R 6

Les points clés 7

Exercices 8

Chapitre 2 Affectations, structures de données et affichages 9

1. Variables simples et affectations 9

2. Autres exemples d'affectation 11

3. Structures de données et affectations en R 15

4. Autres « objets » en R 19

5. Affichage des variables en R 20

6. Spécificités du langage R 21

7. Si vous savez déjà programmer 25

Les points clés 28

Exercices 29

Chapitre 3 Conditions logiques et tests 31

1. Valeurs logiques et tests en R 31

2. Indexation et filtrage vectoriel 36

3. Autres exemples de tests et de filtrage vectoriel 37

4. Applications aux matrices et data frames 44

5. Spécificités du langage R 45

6. Si vous savez déjà programmer 48
Les points clés 50
Exercices 51
Chapitre 4 Boucles et itérations 53
1. Boucles *tant que* 53
2. Boucles *pour* 56
3. Boucles *répéter jusqu'à* 57
4. Imbrications, itérations et sorties de boucles 59
5. Autres exemples d'itérations et de boucles 61
6. Spécificités du langage R 63
7. Si vous savez déjà programmer 64
Les points clés 65
Exercices 66
Chapitre 5 Sous-programmes : les fonctions 67
1. Définition des fonctions nommées et anonymes 67
2. Définition et utilisation des paramètres 72
3. Tests des paramètres 74
4. L'ellipse notée ... 77
5. Autres exemples de fonctions 79
6. Spécificités du langage R 85
7. Si vous savez déjà programmer 86
Les points clés 89
Exercices 90
Chapitre 6 Éviter de programmer en R 95
1. Que faut-il vraiment programmer en R ? 95
2. À l'aide des fonctions classiques de R 98
3. À l'aide des fonctions **`*apply`** 101
4. À l'aide des packages 104
5. Spécificités du langage R 108

6. Si vous savez déjà programmer 109

Les points clés 111

Exercices 112

Chapitre 7 Programmation soutenue 113

1. Structures de données classiques des autres langages 113

2. Problèmes demandant des compétences techniques en algorithmique 114

3. Problèmes de grande taille (*big data volume*) 115

4. Problèmes de grande ampleur (*big data computing*) 115

5. Récursivité 116

6. Spécificités du langage R 118

7. Si vous savez déjà programmer 119

Les points clés 121

Exercices 122

Chapitre 8 Programmation et développement 125

1. Programmation, choix et tests 125

2. Programmation, spécifications et documentations 131

3. Mises à jour, maintenances, objets et packages 132

4. Programmation et interfaçage 132

5. Spécificités du langage R 134

6. Si vous savez déjà programmer 136

Les points clés 137

Exercices 138

Chapitre 9 Déboggage, profilage et optimisation 141

1. Un comportement bizarre de R 142

2. Déboggage 145

3. Écrire `<-` n'est pas une faute de frappe 149

4. Profilage 151

5. Optimisation 157

6. Spécificités du langage R 159
7. Si vous savez déjà programmer 160
Les points clés 162
Exercices 163
Solutions des exercices 165
1. Solutions des exercices du chapitre 1 165
2. Solutions des exercices du chapitre 2 168
3. Solutions des exercices du chapitre 3 179
4. Solutions des exercices du chapitre 4 187
5. Solutions des exercices du chapitre 5 192
6. Solutions des exercices du chapitre 6 208
7. Solutions des exercices du chapitre 7 221
8. Solutions des exercices du chapitre 8 231
9. Solutions des exercices du chapitre 9 247
Conclusion 267
Bibliographie 268
Packages cités et conseillés 269
Sites Web cités et conseillés 270
Index des fonctions et des packages 271

À Paul Calés, Laurence Brunet et Timothée Hamon.
À Séverin et Aude.

Remerciements

L'auteur tient à remercier Benoit Da Mota pour sa relecture attentive de l'ouvrage et ses conseils avisés quant aux rapports entre la programmation R et la programmation en général. Il faudrait aussi remercier toutes les étudiantes et les étudiants, en formation initiale comme en formation continue, pour leurs nombreuses questions, même les plus naïves, parce qu'elles ont permis de cerner les difficultés de comprendre ce qu'est la découverte et l'apprentissage initial de la programmation. Qu'ils et elles soient aussi remerciés pour avoir supporté mes blagues parfois trop nombreuses lorsque je les trouvais trop fatigués par la lecture de mes codes R !

Avant-propos

Pourquoi lire ce livre sur R ?

La plupart des personnes qui programment en R programment mal en R. Il y a deux causes à cela :

- beaucoup d'utilisateurs de R commencent par écrire quelques commandes ou des petits scripts et du coup ils apprennent la programmation R sur le tas sans savoir ce que cela représente vraiment de programmer vectoriellement ;
- lorsqu'on sait déjà programmer dans un autre langage, sauf peut-être avec le langage APL, on ne sait pas programmer vectoriellement.

De ce fait, la plupart des programmes R qui sont écrits sont inefficaces, surchargés de boucles lentes, de code long et pas toujours facile à maintenir. En conséquence, le but de cet ouvrage est double :

1. aider celles et ceux qui ne savent pas programmer du tout à bien débuter la programmation et en particulier la programmation vectorielle ;
2. fournir à celles et ceux qui savent déjà programmer les moyens de comprendre comment s'adapter à la programmation vectorielle en R.

Il y a d'autres particularités dans la programmation R : les packages de base regorgent de fonctions qui dispensent de reprogrammer des actions usuelles, les autres packages contiennent tout ce qu'il faut pour réaliser efficacement des analyses statistiques et des représentations graphiques de tout ordre et de toute nature, la définition des fonctions avec leurs paramètres nommés et l'ellipse fournit des moyens souples pour programmer, la gestion de la mémoire est spécifique aux langages vectoriels interprétés etc.

Ne pas savoir tout cela rend la programmation R parfois déroutante ou « bizarre » au premier abord et il n'est pas étonnant d'être perplexe devant toutes les structures de données et les fonctions associées, devant la difficulté de comprendre comment optimiser du code R... et avec les données dont on peut disposer aujourd'hui, ces fameux *Big Data*, il est important de savoir optimiser du code R.

À l'inverse, savoir tout cela rend la programmation R plus concise, plus efficace, même si parfois le code est – un peu – moins lisible qu'avec la programmation itérative classique.

Ce que contient cet ouvrage

Les cinq chapitres qui suivent sont toujours structurés de la même manière : d'abord le but du chapitre, puis des notions algorithmiques et leur traduction dans le cadre de la

programmation avec de nombreux exemples puis les spécificités de R liées à ces notions et enfin des remarques pour celles et ceux qui savent déjà programmer.

Une programmeuse, un programmeur classique pourra ainsi survoler les notions (mais pas trop vite, parce que R est « spécial ») pour se concentrer sur les spécificités et sur ces remarques, de façon à pouvoir désapprendre ce qui était considéré comme des bonnes pratiques dans un autre langage et qui se révèle souvent inadapté ou inefficace en R.

Pour les débutantes et les débutants, cela permettra ainsi de séparer ce qui tient de la programmation classique de ce qui ne ressort que de la programmation vectorielle, fournissant ainsi la possibilité d'utiliser plus tard d'autres langages de programmation.

L'enchainement des chapitres est très traditionnel, même si le chapitre 1 ne fait pas toujours partie des manuels classiques de programmation. Dans ce chapitre figurent les principes généraux de programmation avec une insistance particulière sur la rigueur et l'effort soutenu que demande la programmation. Le chapitre 2 présente les affectations, les structures de données et explique comment réaliser de « beaux » affichages. Le cœur du chapitre 3 est constitué par les conditions logiques et les tests, complété au chapitre 4 par les boucles et itérations. Ensuite le chapitre 5 détaille la définition des fonctions et la notion de script. Les bases de la programmation sont alors posées.

Au chapitre 6, le lecteur apprendra comment éviter de programmer des actions usuelles, comment se passer des boucles « POUR » dans pratiquement tous les cas. Le but du chapitre 7 qui est intitulé « Programmation soutenue » est de faire passer le lecteur du statut de programmeur débutant à celui de programmeur averti, *via* des exemples plus approfondis alors qu'au chapitre 8 on viendra détailler les nombreuses différences entre programmation et développement. Enfin, le chapitre 9 fournira quelques repères sur le déboggage et l'optimisation d'applications écrites en R. Les solutions détaillées des exercices sont en fin d'ouvrage, juste avant la conclusion et les annexes (liste de livres, packages et sites conseillés).

Les exercices sont une partie essentielle de ce livre. C'est pourquoi ils représentent avec leurs corrigés pratiquement la moitié de l'ouvrage. Il est important de les résoudre, ou tout au moins d'essayer de les résoudre pour vérifier si les concepts ont été compris. Il y a souvent en programmation plusieurs façons de faire. Nous avons essayé dans les solutions de fournir des exemples de code lisible, robuste et portable.

Les solutions des exercices sont souvent très détaillées de façon à bien montrer comment on trouve une première solution, comment on l'améliore pour obtenir une solution complète et générale car il est rare qu'en programmation on pense à tout dès le premier jet. Il nous semble important de faire passer ce message qu'un programme ne s'écrit pas d'une seule traite et en une seule fois.

À défaut de résoudre explicitement tous les exercices, il est conseillé de lire au moins les solutions, pour profiter de la progression pédagogique (les exercices sont la plupart du temps de difficulté croissante pour chaque chapitre) et pour voir, souvent après des essais corrects mais incomplets ou trop « itératifs », des « vraies » solutions R.

Ce qu'on ne trouvera pas dans cet ouvrage

Pour que cet ouvrage reste d'une taille raisonnable et joue son rôle d'initiation et de sensibilisation à la programmation vectorielle, nous avons décidé de ne présenter ni la création de classes d'objets, ni la réalisation de packages sauf sous forme de mini-exemples dans les exercices du chapitre 8. Car si les concepts sous-jacents sont relativement simples, leur exécution en pratique requiert une technicité qui demande une forte expérience de la programmation. De plus R utilise trois systèmes de classe d'objets (objets R5, S3 et S4), ce qui complique leur présentation.

On ne trouvera pas non plus beaucoup de calculs statistiques compliqués ou de réalisations graphiques sophistiquées car la plupart des autres livres sur R traitent ces sujets. Nous utiliserons de temps en temps des calculs statistiques simples comme la moyenne, la médiane, nous ferons de temps en temps références à des tracés graphiques comme les *boxplots* (ou *boites à moustaches* en français), mais aucune compétence particulière en statistique ou en mathématique n'est requise pour lire cet ouvrage.

L'idée centrale pour comprendre ce que contient ce livre, pour reprendre une analogie classique, consiste à dire que nous voulons apprendrons au lecteur comment tenir sur un vélo, ou sur des skis, comment prendre de la vitesse, tenir la distance, mais pas comment faire le tour de France ou participer à une épreuve des jeux olympiques. Après tout, il ne s'agit que d'une introduction à la programmation R.

Prérequis

Pour profiter pleinement de cet ouvrage, il y a quelques prérequis. Tout d'abord il faut disposer de temps libre et être reposé(e) car la programmation est une activité intellectuelle qui demande de la concentration.

Ensuite il faut de la patience et accepter de travailler de temps en temps avec un papier et un crayon pour réfléchir aux solutions possibles avant d'utiliser l'ordinateur. Programmer demande du recul et de l'expérience, il ne faut pas être dans le feu de l'action et du Web si on veut programmer sereinement.

Il faut aussi avoir envie plutôt qu'avoir besoin de programmer. Lire cet ouvrage en dilettante ne vous apprendra pas grand chose ou ne vous sera pas très utile par ce que vous n'aurez rien intériorisé. Si vous êtes forcé(e) de programmer, vous programmerez mal par ce que vous ne mettrez pas assez de *cœur à l'ouvrage*.

Enfin, bien sûr, comme cet ouvrage n'est pas une initiation au logiciel R, il faut connaitre *un peu* R et avoir déjà utilisé R. Par exemple, n'avoir jamais utilisé au moins une fois les fonctions **read.table()**, **cat()**, **paste()** ou **names()** signifie sans doute que vous n'avez pas assez manipulé R en ligne de commande et vous aurez du mal à suivre les exemples utilisés, non pas à cause de la programmation mais à cause des fonctions de R utilisées.

Donc si vous n'avez pas beaucoup de pratique de R ou si vous l'avez utilisé seulement *via* une interface comme Rcmdr (« R commandeur »), Rattle ou rkward, il vaut mieux commencer par vous entrainer à écrire des calculs simples en R et réaliser de petits scripts en ligne de commande avant de vous « attaquer » à cet ouvrage.

Il existe de nombreux cours et pages d'initiation à R sur Internet, sans compter les ouvrages en français qui traitent de R. Vous trouverez quelques titres d'ouvrages que nous recommandons dans la bibliographie située dans l'annexe en fin de ce livre et qui peuvent vous servir de compléments – et non pas de préambule – à cet ouvrage.

Nous avons décidé de rédiger ce livre car, malheureusement, très peu d'ouvrages généraux présentent R comme un « vrai » langage de programmation et encore moins exploitent explicitement le fait que R est vectoriel. Ils se contentent souvent d'une présentation rapide de la syntaxe de R, pas de sa « philosophie ». Vous verrez souvent cette partie de phrase dans cet ouvrage : « *Comme R est vectoriel...* » car c'est un point clé de la programmation en R. Et lorsque vos collègues vous demanderont pourquoi vos programmes à vous sont si courts à écrire et si rapides à exécuter, vous pourrez leur répondre : « *C'est parce que R est vectoriel !* ».

Vous trouverez également en annexe les différents packages que nous considérons comme importants pour R sous l'angle de la programmation et une liste de quelques sites Web utiles pour bien progresser en R et se tenir au courant des évolutions de R.

Nous avons bien conscience qu'on n'apprend pas l'anglais, l'allemand ou le chinois en vingt-quatre heures. On n'apprend pas non plus à programmer en quelques dizaines d'heures juste en lisant des scripts ou des solutions d'exercices ou en « bricolant » quelques petits programmes. C'est à force de pratique que l'on apprend, parfois en commettant des erreurs, ou en lisant les erreurs des autres et rien ne remplace donc vos essais personnels de programmation. De plus il faut souvent plusieurs essais, plusieurs versions progressives avant que le script ne soit complet, testé, aussi bien dans les bons cas que dans la gestion des erreurs les plus courantes, avec au passage quelques idées sur le « profil » du code, sa rapidité, sans oublier sa documentation.

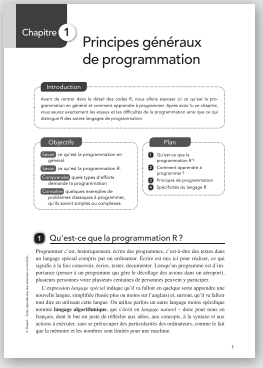
Pour conclure cette introduction, je vous souhaite autant de plaisir que possible à lire ce livre et à programmer. Un programme est un petit peu une œuvre d'art, alors soyez artistes, soyez heureuses et soyez heureux.

À la découverte de votre livre

1 Ouverture de chapitre

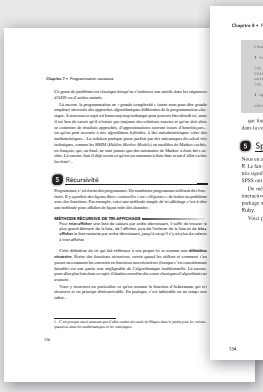
Elle donne :

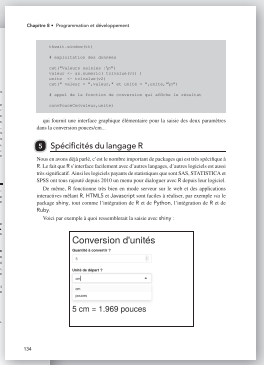
- une **introduction** aux sujets et aux problématiques abordés dans le chapitre
- un rappel des **objectifs** pédagogiques
- le **plan** du chapitre


Chapitre 1
Principes généraux de programmation
Introduction
Objectifs
Plan
1 Qu'est-ce que la programmation R ?

2 Le cours

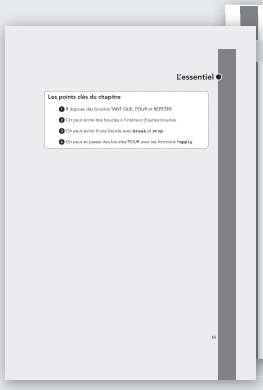
Un cours concis et structuré.

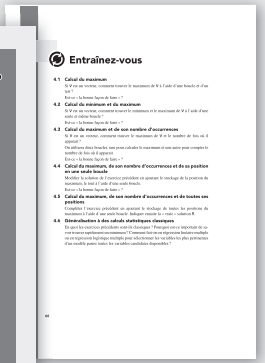

5 Récursivité


5 Spécificités du langage R

3 En fin de chapitre

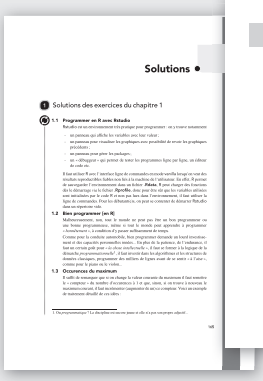
- **L'essentiel** : les **points clés** pour réviser les connaissances essentielles
- Des **exercices** pour tester ses connaissances

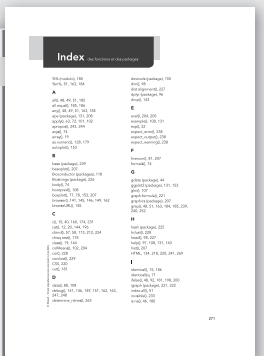

L'essentiel
Les points clés du chapitre


Entraînez-vous

4 En fin d'ouvrage

- Les solutions des exercices
- Une bibliographie
- Les packages cités et conseillés
- Les sites web cités et conseillés
- Un index des fonctions et des packages


Solutions


Index des fonctions et des packages

5 Ressources numériques

Retrouvez sur www.dunod.com l'ensemble des lignes de code des exercices.

Chapitre 1

Principes généraux de programmation

Introduction

Avant de rentrer dans le détail des codes R, nous allons exposer ici ce qu'est la programmation en général et comment apprendre à programmer. Après avoir lu ce chapitre, vous saurez exactement les enjeux et les difficultés de la programmation ainsi que ce qui distingue R des autres langages de programmation.

Objectifs

- **Savoir** ce qu'est la programmation en général.
- **Savoir** ce qu'est la programmation R.
- **Comprendre** quels types d'efforts demande la programmation.
- **Connaître** quelques exemples de problèmes classiques à programmer, qu'ils soient simples ou complexes.

Plan

1. Qu'est-ce que la programmation R ?
2. Comment apprendre à programmer ?
3. Principes de programmation
4. Spécificités du langage R

1 Qu'est-ce que la programmation R ?

Programmer c'est, historiquement, écrire des programmes, c'est-à-dire des textes dans un langage spécial compris par un ordinateur. Écrire est mis ici pour réaliser, ce qui signifie à la fois concevoir, écrire, tester, documenter. Lorsqu'un programme est d'importance (penser à un programme qui gère le décollage des avions dans un aéroport), plusieurs personnes voire plusieurs centaines de personnes peuvent y participer.

L'expression *langage spécial* indique qu'il va falloir en quelque sorte apprendre une nouvelle langue, simplifiée (basée plus ou moins sur l'anglais) et, surtout, qu'il va falloir tout dire en utilisant cette langue. On utilise parfois un autre langage moins spécifique nommé **langage algorithmique**, qui s'écrit en *langage naturel* – donc pour nous en français, dont le but est juste de réfléchir aux idées, aux concepts, à la syntaxe et aux actions à exécuter, sans se préoccuper des particularités des ordinateurs, comme le fait que la mémoire et les nombres sont limités pour une machine.

La programmation en R est un peu spéciale dans la mesure où on n'écrit pas vraiment des programmes mais plutôt au départ des scripts dans un environnement. Nous reviendrons régulièrement sur cette différence au travers d'exemples afin que vous puissiez aussi programmer dans d'autres langages. Dans un langage de programmation, on trouve en général cinq actions principales :

1. effectuer un calcul directement, plus ou moins complexe ;
2. lire une ou plusieurs valeurs, au clavier, dans un fichier, en mémoire... ;
3. écrire une ou plusieurs valeurs, à l'écran, dans un fichier, en mémoire... ;
4. appeler un sous-programme, avec éventuellement des paramètres ;
5. gérer le flux des instructions, faire des tests et des boucles.

Savoir programmer, c'est maitriser les concepts et les techniques liés à chacune de ces actions, ce qui demande surtout de la pratique au début, puis de la théorie des structures de données et des paradigmes de programmation dès qu'on s'attaque à des problèmes d'importance ou complexes.

2 Comment apprendre à programmer ?

Pour apprendre à programmer, il faut utiliser les mêmes techniques que pour apprendre une nouvelle langue. Cela inclut donc beaucoup de pratique, de répétition, d'entrainement. Après avoir maitrisé le vocabulaire de base, on passe à la syntaxe élémentaire puis au bout d'un certain temps on passe à des phrases plus longues, on apprend la grammaire et surtout, on continue encore et encore...

L'avantage avec l'ordinateur, c'est qu'il est infiniment patient et qu'on ne risque pas de l'énerver en commettant toujours la même erreur, qu'on peut essayer et réessayer de nombreuses fois, jusqu'à temps qu'on arrive à se faire comprendre. Par contre, le gros défaut, c'est qu'il est bête ou plus exactement mécanique et qu'il est incapable d'accepter l'à peu près.

Par exemple si un étranger vous dit « *je voudrais une papier blanche* » ou « *je voudrais une blanche papier* » vous corrigerez de vous-mêmes en « *papier blanc* » et vous comprendrez ce que veut la personne. Si par contre vous demandez à l'ordinateur d'appeler le sous-programme **`CALCUL(x,)y`** au lieu du sous-programme **`CALCUL(x,y)`**, il ne saura que vous répondre « `erreur de syntaxe` » et peut-être, dans le meilleur des cas, « `variable manquante après la virgule` », sans se rendre compte et corriger de lui-même la position du **`y`** dans la phrase.

Nous allons donc progressivement apprendre des mots et les utiliser dans des phrases courtes, puis réfléchir pour savoir comment construire des phrases longues, produire automatiquement des séries de phrases, automatiser des comportements...

Une des difficultés sera de comprendre comment l'ordinateur comprend nos phrases, de mettre en place des automatismes, des habitudes de programmation.

Il faut d'abord se convaincre des qualités qu'il faut développer pour savoir programmer avant de se lancer dans l'écriture de programmes. Même un exercice qui parait *a priori* simple se révèle parfois technique à réaliser. Et une fois que l'on sait à peu près programmer, il reste encore beaucoup à à faire pour « bien » programmer en R.

Si vous ne savez pas programmer, certains exercices vous paraitront difficiles. Si vous savez déjà programmer, certains exercices, mais sans doute pas les mêmes, vous paraitront difficiles. C'est normal : la programmation est un art et R ne ressemble pas aux autres langages. Par contre la programmation reste la programmation, quelque soit le langage et il faut savoir ce que cela implique de programmer. Tout le reste de ce chapitre est dédié à cette notion, ses prérequis et ses contraintes.

Les exercices en fin de chapitre sont gradués. Pour reprendre l'analogie avec les langues, si dans le corps du chapitre nous parlons de mot, d'adjectif comme « *le chat* », « *la petite souris* », les exercices vont jusqu'à la production de grandes phrases comme « *le gros chat roux de Chateauroux a mangé la petite souris grise et maline dans le beau jardin de ma tante Éloïse* ». Ils sont donc obligatoires si vous voulez parler couramment en R.

3 Principes de programmation

On peut certainement considérer qu'il y a quatre grands principes de programmation :

- programmer, c'est réfléchir et organiser ;
- programmer, c'est choisir et expliciter ses choix ;
- programmer, c'est tester, prévoir et valider ;
- programmer, c'est être précis, endurant et avec beaucoup de rigueur.

Avant de rentrer dans le détail des instructions de programmation, voici quelques exemples de situation pour expliciter ces grands principes de la programmation. Vous en déduirez facilement les qualités et compétences requises pour savoir programmer en espérant que cela ne vous fera pas fuir et que vous y survivrez !

3.1 Programmer, c'est réfléchir et organiser

Imaginons que nous cherchions ce qu'on nomme maximum d'une série de valeurs, c'est-à-dire la plus grande valeur.

Prenons par exemple les valeurs 1 8 2 5 8 7 8. Je suis sûr que vous avez trouvé le maximum qui est 8. Mais de quel 8 s'agit-il ? Est-ce le premier ? Le second ? Le dernier ?

Et comment avez-vous fait ? Avez-vous passé en revue toutes les valeurs à partir du début ? Ou à partir de la fin ? Ou était-ce évident ? Dans ce cas, comment expliquer à l'ordinateur comment le trouver ?

Et s'il s'agissait de pourcentages, vu que le maximum est dans l'absolu 100 %, est-ce que cela changerait quelque chose à votre façon de chercher le maximum ?

Imaginons maintenant que nous voulions le maximum, le nombre de fois où il apparait et la (ou les) positions où il apparait. Sachant que la liste des valeurs est longue pour un humain, disons une centaine de valeurs, sauriez-vous résoudre ce problème en ne passant en revue la liste qu'une seule fois ?

Programmer, c'est aussi se poser ce genre de questions. En cas de très grandes listes pour l'ordinateur (par exemple si chaque valeur est obtenue au bout d'un long calcul), il faut réfléchir pour trouver une méthode rapide. Passer en revue la liste une première fois pour trouver le maximum et la passer en revue une seconde fois pour calculer le nombre d'occurences et les positions du maximum est une méthode simple mais non rapide.

L'usage veut donc qu'avant de commencer à programmer on réfléchisse au problème, aux entrées et aux sorties, aux fonctionnalités de ce qu'on veut faire, aux durées possibles et prévoir en retour ce qu'on va programmer...

3.2 Programmer, c'est choisir et expliciter ses choix

Voici un exemple classique qui fait partie de la programmation traditionnelle : construire le nom d'un fichier de sortie à partir d'un fichier d'entrée. Par exemple on veut, à partir du fichier nommé `serie35.xls` construire le fichier `serie35.txt` ou encore, à partir du fichier `serie_1.manip2.rep036.xls` construire le fichier `serie_1.manip2.rep036.txt`.

Vous voyez certainement ce qu'il faut faire : repérer le « bon » point qui indique la fin du nom de fichier (ce n'est donc pas le premier point qu'on rencontre dans le nom de fichier, mais plutôt le dernier), puis extraire la partie avant ce point et rajouter `txt` ou `.txt` suivant qu'on a déjà extrait le point ou non.

Si les méthodes *extraire la première partie avec le point* et *sans le point* sont sans doutes équivalentes en terme de simplicité et de vitesse, quelle est la meilleure méthode ? Et selon quels critères ? Une fois que vous aurez trouvé ce qui est la meilleure méthode pour vous (et vos fichiers) – ce qui n'est peut-être pas la même meilleure méthode que pour vos collègues – il faut la noter, documenter ce choix pour éviter de se poser la question à nouveau et toujours s'y tenir, ce qui peut se faire en écrivant un sous-programme qui réalise ce traitement.

3.3 Programmer, c'est tester, prévoir et valider

Reprenons l'exemple précédent de construction d'un nom de fichier en sortie. Nous avions basé notre analyse sur le fait qu'il y avait un point en fin de nom de fichier d'entrée. Et si ce n'était pas le cas ? C'est ce qui risque d'arriver non pas en ligne de commandes, si on tape le nom du fichier, mais si le nom du fichier d'entrée a par exemple été mal construit par un autre programme. Que fait la méthode précédente dans ce cas ?

La réponse dépend de la façon dont vous avez détecté le point. Certaines fonctions renvoient, lorsque le point n'est pas trouvé, la valeur −1, d'autres une valeur égale à *1 de plus que la longueur du nom de fichier*. Ces deux choix se valent, le plus important est d'y avoir pensé car ce qu'il faut faire dans ces cas dépend de la valeur renvoyée.

Nous essaierons, dans le cadre des exercices présentés, de prévoir les cas usuels classiques d'erreur (fichier non présent ou faute de frappe, liste vide de valeurs...) afin d'avoir des programmes dits **robustes** (ils ne se « plantent » pas) et **fiables** (leur résultat est conforme à ce qu'on attendait).

Cela signifie qu'en conséquence il faudra tester différents cas classiques afin de valider le « bon » comportement de nos programmes dans les conditions normales d'application et qu'il faudra aussi penser à gérer les « mauvais » cas et les « cas exceptionnels ».

En particulier, il faudra réfléchir, organiser et prévoir ce qu'on fait par exemple si un fichier n'est pas vu alors qu'on traite une liste de fichiers. Pensez-y. Vous, que feriez-vous dans ce cas-là ?

Interrompre le programme pour demander un nouveau nom de fichier est peut-être inadapté, tout arrêter est sans doute trop brutal, se contenter d'afficher un message d'erreur qui risque de disparaitre de l'écran si la liste des fichiers à traiter est longue est risqué...

Donc la solution qui gère l'absence d'un fichier n'est finalement pas si simple que cela, en fonction du nombre de fichiers à traiter. Il faut être capable de disposer de tous les moyens pour programmer soit un arrêt du script si chaque fichier est vital, soit savoir « stocker » les erreurs de noms de fichier pour les afficher en fin de traitement.

3.4 Programmer, c'est être précis, endurant et avoir beaucoup de rigueur

Au vu des situations présentées ci-dessus, il est clair que la programmation n'est pas au départ une partie de plaisir puisqu'il faut réfléchir beaucoup, essayer de penser à tout. Et trouver une « bonne » solution consiste souvent à réaliser un compromis entre simplicité et vitesse d'exécution. Si on ajoute qu'il faut penser aux erreurs possibles, qu'il faut tester qu'on a bien tout prévu, qu'il faut aussi gérer les erreurs prévisibles, documenter ce qu'on choisit, la programmation se révèle vite une entreprise technique et non immédiate.

Par contre, c'est au final une réelle joie que d'avoir un programme qui « tourne » sans « bugger » ou « boucler », que d'obtenir presque automatiquement toute une série de résultats et de fichiers sans avoir à exécuter de nombreux copier/coller ou de fastidieuses manipulations à faire et à refaire, de fournir à la communauté un programme qui peut aider d'autres collègues et faire gagner ainsi beaucoup de temps, cette denrée rare...

4 Spécificités du langage R

R est un langage interprété (non compilé), non typé explicitement et avec un environnement d'exécution. La programmation en R ressemble au départ et approximativement à la programmation traditionnelle. Ce qui change beaucoup, c'est principalement le fait que R est vectoriel, qu'on utilise un environnement qui sauvegarde les variables et qu'il y a des milliers de fonctions de base et des milliers de fonctions complémentaires disponibles dans des packages. Du coup, de nombreuses actions (trier, calculer, tracer...) sont soit élémentaires soit déjà programmées.

Savoir programmer en R signifie savoir tirer partie de ces caractéristiques. Vous aller donc devoir apprendre de nombreuses fonctions, beaucoup plus qu'avec un langage traditionnel, avec leurs paramètres et vous allez écrire de nouvelles fonctions pour compléter toutes celles qui existent ou plutôt pour les mettre en oeuvre ensemble dans un tout cohérent qu'on nomme **script**, de façon à commander R pour obtenir exactement tout ce que vous voulez. De fait, une partie de la programmation R consiste à ne pas programmer des calculs, mais à savoir utiliser des programmes déjà écrits, à juste enchainer des appels de fonctions qui, elles, calculent, tracent, lisent et écrivent des données...

Si vous ne savez pas ce que le mot **vectoriel** signifie, ne vous inquiétez pas, vous allez le découvrir rapidement. En deux mots, cela veut dire qu'on se place à un haut niveau d'abstraction, que la plupart du temps on laisse R passer en revue les structures au lieu de tout détailler.

Si vous ne savez pas ce que le terme **milliers de fonctions** implique, vous allez vous en rendre compte au fur et à mesure des chapitres. Mais là encore en deux mots, cela veut dire que chaque action qui est un peu générale a déjà été programmée. Où et comment, c'est une autre histoire. Heureusement, le site officiel du CRAN et ses nombreux miroirs dans le monde sont là pour vous aider dans cette tâche, de même que les *Task Views* que vous devez certainement un peu connaitre. Sinon, faites une pause dans la lecture de ce livre et allez feuilleter les pages Web associées, cela vous servira très bientôt...

En conclusion, comme pour une langue, il va falloir faire un gros effort de recherche et de mémoire pour bien dire ce que vous voulez, avec les bons termes, techniques et précis. Sinon, de la même façon que dans une langue étrangère vous passerez par une longue périphrase à peine compréhensible comme « *l'objet plastique quasi rectangulaire mais assez plat d'un quinzaine de centimètres pour coiffer les cheveux* » au lieu du simple mot « *peigne* », en R vous utiliserez plusieurs instructions, parfois de façon maladroite là où « la bonne » fonction de R fait tout le travail directement.

Un dernier conseil avant de passer aux exercices du chapitre pour tester si vous avez bien tout compris de ces prérequis et contraintes : ne commencez jamais un chapitre de ce livre le soir avant *Plus belle la vie* ou *Game of Thrones*. Vous risqueriez de rater la fin de l'épisode !

Les points clés du chapitre

1. La programmation est une activité intellectuelle qui demande rigueur et ténacité.
2. Apprendre à programmer ressemble au début à apprendre une nouvelle langue.
3. Pour savoir programmer, il faut beaucoup de pratique.

Entraînez-vous

1.1 Programmer en R avec Rstudio

Pourquoi vaut-il mieux utiliser Rstudio que l'interface standard si on veut programmer en R ? Dans quels cas faut-il utiliser R avec l'interface en ligne de commandes ?

1.2 Bien programmer [en R]

Est-ce que tout le monde peut devenir une bonne programmeuse ou un bon programmeur ?

1.3 Occurences du maximum

Montrer comment on peut trouver « à la main » sur la liste de valeurs 1 8 2 5 8 7 8 le maximum et son nombre d'occurences en un seul passage en revue de la liste.

1.4 Plus grande répétition

Étant donnée une chaine de caractères, expliquer comment trouver la plus grande sous-chaine répétée d'au moins deux caractères, son nombre de répétitions et ses occurences.

Application 1. Quelle est la plus grande sous-chaine répétée pour la chaine de caractères "MON BEAU BATEAU BLEU" ?

Application 2. Quelle est la plus grande sous-chaine répétée d'au moins deux caractères de la séquence d'ADN de *Pseudomonas putida GB-1* qui est un des cinq exemplaires séquencés et complètement assemblés de *Pseudomonas putida*, chaine de caractères disponible au format Fasta à l'adresse

http ://forge.info.univ-angers.fr/~gh/Abdc/Data/putida_gb_1.fasta.zip

Que risque-t-on d'obtenir ?

1.5 Plus grande sous-chaine commune

On voudrait trouver la plus grande sous-chaine de caractères commune d'un ensemble de séquences d'ADN bactérien avec au moins deux caractères, par exemple pour les 10 génomes séquencés de *Helicobacter pylori* disponibles à l'adresse

http ://forge.info.univ-angers.fr/~gh/Abdc/Data/pyloris.zip

Que devrait-on trouver ?

1.6 Programmation d'un aéroport

On voudrait programmer la gestion des vols d'un aéroport. Donner les grandes lignes de la résolution de ce problème.

1.7 Que faut-il conclure de ces exercices ?

Que montrent, *in fine* ces exercices, et surtout les trois derniers ?

Quel impact cela peut-il avoir sur les chapitres qui suivent avec leurs exercices et sur la réponse aux deux questions « *combien de temps faut-il pour savoir bien programmer ?* » et « *peut-on tout programmer ?* » ?