Vers une recherche reproductible

Faire évoluer ses pratiques

Loïc Desquilbet, Sabrina Granger, Boris Hejblum,
Arnaud Legrand, Pascal Pernot, Nicolas Rougier Facilitatrice :
Elisa de Castro Guerra

2019-05-06

Table des matières

Préambule

Ce livre s'adresse à tout acteur de la recherche scientifique qui :

— se pose des questions sur la recherche reproductible ou
— souhaite améliorer ses pratiques.

L'objectif de cet ouvrage est de donner les outils pour comprendre les enjeux de la recherche reproductible et permettre à un chercheur de vérifier ou d'obtenir de nouveau les résultats d'un autre chercheur. Cet autre chercheur peut être lui-même car ce qui était clair à un moment donné ne l'est plus quelques semaines plus tard. Notre ambition est d'apporter des solutions avec un niveau de technicité varié, permettant à chacun d'améliorer sa pratique dans son domaine et selon ses moyens.

Le livre se structure en trois grandes parties :

— la première partie traite de la notion même de recherche reproductible selon différentes perspectives. Il s'agit de prendre pour point de départ des expériences "traumatiques" engendrées par des pratiques plus ou moins (moins en fait) propices à une recherche reproductible.
— la deuxième partie explicite les causes et les origines des différents problèmes auxquels chaque chercheur, quel que soit son degré d'expertise dans son domaine, peut être confronté un jour ou l'autre.
— la troisième partie présente enfin une sélection de solutions offertes par la recherche reproductible pour résoudre ou prévenir ces problèmes à l'échelle de chacun.

Première partie

Chroniques de la non reproductibilité

1

État des lieux

1.1 La crise de la reproductibilité de la science

La crise de la reproductibilité de la science est aujourd'hui un phénomène mondial et largement transdiciplinaire qui concourt à la défiance de la société à l'égard du monde de la recherche (**?**). Le sujet est ancien, mais la situation semble avoir atteint un point critique. Des études ont par exemple démontré qu'il n'était pas possible d'obtenir de nouveau les résultats d'études pré-cliniques ou cliniques (**?**) (**?**). Si la reproductibilité des résultats ne peut être considérée comme seul critère de la scientificité d'une recherche, cette crise suscite des interrogations au sein même de la communauté scientifique.

1.2 Définition(s) de "recherche reproductible" ?

Si la communauté scientifique perçoit ce que peut être une recherche reproductible dans son propre domaine, il s'avère *a priori* difficile de fournir une définition standard satisfaisante pour toutes les disciplines. De fait, parce que la notion de "même résultat" dépend fortement du domaine de recherche. Pour les uns, il suffira de confirmer la signifiance d'un effet, pour les autres, il s'agira d'obtenir le même résultat numérique bit pour bit. L'expression "recherche reproductible" apparaît dès 1992, lors du congrès de la "*Society of Exploration Geophysics*" : "*The first appearance of the phrase "reproducible research" in a scholarly publication appears to be an invited paper presented at the 1992 meeting of the Society of Exploration Geophysics (SEG), from the group of Jon Claerbout at*

Stanford (Claerbout & Karrenbach, 1992). [...] His idea of reproducible research was to leave finished work (an article or a thesis) in a state that allowed colleagues to reproduce the calculation, analysis and final figures by executing a single command. The goal was to merge a publication with its underlying computational analysis" (**?**). De nombreuses définitions de "recherche reproductible" sont fournies par Barba (**?**). Parmi celles-ci, nous retiendrons la suivante, issue de l'article de Vandewalle *et al*.(**?**) : ***"A research work is called reproducible if all information relevant to the work, including, but not limited to, text, data and code, is made available, such that an independant researcher can reproduce the results"*** (**?**). (Notre traduction : "Un travail de recherche est dit reproductible si toutes les informations qui concernent ce travail incluant, sans s'y limiter, le texte, les données, et le code de programmation, sont rendues disponibles de telle sorte que n'importe quel chercheur indépendant peut reproduire les résultats.")

1.3 Pourquoi la question de la reproductibilité est-elle devenue centrale dans les débats actuels ?

Le fait que les problèmes de reproductibilité occupent une telle place dans les débats actuels n'est pas tout à fait un hasard. Le numérique, sous des formes multiples, a largement investi tous les champs de la science et l'outil informatique occupe une place incontournable : stockage, formatage, archivage, indexation, analyse, modélisation, statistiques, environnements, précision, *etc*. Or, peu de chercheurs ont été formés (ou se forment) aux fondamentaux et aux bonnes pratiques liés aux outils informatiques. Cela peut amener à la publication de résultats fragiles (dans le sens "peu robustes") dans le meilleur des cas, et faux dans le pire des cas - mais ce n'est pourtant pas là que se situent les plus graves dangers pour la recherche.

1.4 Où l'on parle de recherche reproductible de manière pragmatique

Quel peut être le point commun entre : un archéologue en train d'effectuer une campagne de fouille, un biologiste préparant une nouvelle expérience dans son laboratoire, un numéricien finalisant une simulation de grande ampleur ? Tous sont exposés aux risques ~~drames~~ suivants, indépendamment de leur volonté de contribuer à l'accroissement des connaissances dans leurs domaines respectifs :

— envoyer à des collègues des données qui ne pourront pas être lues pour des raisons d'incompatibilité de formats,
— réaliser une simulation effectuée sur deux machines différentes et obtenir des résultats radicalement différents,
— se rendre compte qu'une donnée essentielle était stockée sur feu le disque dur (*requiescat in pace*),
— renoncer à une hypothèse prometteuse faute de pouvoir reproduire une de ses propres expériences

La liste pourrait s'allonger. Ne vous êtes-vous jamais posé les questions suivantes : "Suis-je vraiment sûr de mon analyse statistique ?", "Suis-je capable de recréer une figure conçue il y a 6 mois ?" Outre votre équipe de recherche, votre communauté scientifique et *in fine* le monde non académique, le premier bénéficiaire d'une recherche reproductible, c'est d'abord *vous*. Une recherche reproductible facilite en effet les tâches les plus quotidiennes, permet de garantir l'exactitude des méthodes et de documenter l'ensemble de la pratique scientifique. Cela peut-il constituer un gage de qualité de la recherche ? Non. Cela y participe, mais ne suffit pas. En effet, une recherche reproductible (au sens de l'ouvrage, "une recherche dont les résultats publiés peuvent être reproduits") n'est pas synonyme de "bonne" recherche : une mauvaise recherche peut tout à fait être reproductible (*spoiler alert* : *don't try at work* !).

1.5 Aperçu (très rapide) des causes d'une recherche non reproductible

Comme nous le verrons dans ce livre, les causes d'une recherche non reproductible sont très nombreuses. Le suspect habituel est la perte de données. D'autres causes s'avèrent plus difficiles à détecter : par exemple, le chaos numérique, aussi subtil à identifier que vecteur de troubles majeurs. Une fois de plus, il ne s'agit pas de développer une vision accusatoire des pratiques de recherche. L'impossibilité même de reproduire des résultats n'est pas engendrée par la malhonnêteté scientifique, mais s'avère bien plus souvent le fruit d'une forme de méconnaissance, de pratiques plus ou moins hasardeuses. Sous des dehors souvent anodins, les petits "braconnages" du quotidien ("Ça va passer") et autres rustines font le lit de la "dette technique" qui à terme, peut devenir insurmontable et peut condamner un laboratoire. Au travers de situations fictives mais hélas réalistes, nous verrons qu'à l'origine des problèmes de reproductibilité se trouve un ensemble de concepts fondamentaux qu'il est nécessaire de connaître. L'objectif n'est pas de les maîtriser totalement. La recherche reproductible n'exige pas d'adopter une logique du "tout ou rien". Il existe des solutions très simples à mettre en œuvre que tout un chacun peut s'approprier. D'autres solutions demanderont un peu plus de temps et d'énergie.

1.6 La minute théologie négative : ce que vous ne trouverez pas dans cet ouvrage

Il n'est pas question dans cet ouvrage de traiter toutes les solutions pour garantir la reproductibilité de la recherche au sens de Randall et Welser (**?**) : ainsi, la question de la qualité de la recherche est hors périmètre du présent ouvrage. En effet, nous vous proposons plutôt de nous focaliser sur les solutions qui permettent communiquer des ré-

sultats pouvant être reproduits de façon exhaustive. Ainsi, nous n'allons pas traiter des solutions à des problèmes qui nuisent à la qualité de la recherche, et en particulier :

— aller à la "pêche" aux résultats significatifs parmi tous les tests statistiques réalisés ("p-hacking") (**?**), (**?**), (**?**)
— générer une hypothèse de recherche *a posteriori*, c'est-à-dire après avoir obtenu un résultat significatif (« harking ») (**?**)
— sur-interpréter le résultat statistique qui est significatif (« Probability That a Positive Report is False ») (**?**), (**?**), (**?**), (**?**), (**?**)

Pour tous ces sujets cités précédemment, nous invitons le lecteur à se référer à la littérature :

The Seven Deadly Sins of Psychology : A Manifesto for Reforming the Culture of Scientific Practice (**?**)

"Why Most Published Research Findings Are False" (**?**)

"A manifesto for reproducible science" (**?**)

Statistics Done Wrong (**?**)

"A Guide to Robust Statistical Methods in Neuroscience" (**?**)

2

Retours d'expériences

Nous avons recueilli ci-dessous plusieurs témoignages fictifs de *personas* (**?**) incarnant différents acteurs de la recherche. Il s'agit de personnages inventés mais néanmoins vraisemblables car inspirés d'expériences réelles. Parce qu'ils se présentent sous une forme narrativisée, les *personas* permettent :

— à des chercheurs, d'appréhender concrètement différentes questions de recherche reproductible, de contextualiser ces enjeux dans un cadre quotidien ;
— à des personnels de soutien à la recherche, de mieux comprendre les questions auxquelles sont confrontés les chercheurs.

2.1 *Charles P., doctorant en sociologie*

"J'interroge des personnes en situation de précarité économique afin de recueillir leurs témoignages. Je consigne toutes mes notes dans un carnet relié et je retranscris les informations démographiques dans un tableur Calc afin de compiler quelques statistiques : pyramide des âges, répartition des sexes, *etc*. Dans le cadre de ce projet, je collabore étroitement avec un autre doctorant qui a le même directeur de thèse que moi. Il alimente lui aussi ce fichier Calc, que nous nous échangeons régulièrement par clé USB."

2.2 *Jeanne A., jeune méthodologiste en biostatistique*

"J'ai terminé mon Master 2 l'année dernière. Dans ce cadre, j'ai été formée à conduire des analyses statistiques sous R. J'analyse régulièrement des jeux de données fournis par des cliniciens qui travaillent au CHU. Les données sont au format Excel et me sont fournies directement par mail. Je travaille de manière relativement isolée pour effectuer ma tâche d'analyse de données : mes collègues sont en effet tous médecins ou biologistes.

J'ai récemment participé à la rédaction d'un article scientifique et nous venons de recevoir les commentaires des relecteurs : je dois modifier les couleurs d'une figure afin que celle-ci soit lisible en noir et blanc. Comme je n'arrivais pas à remettre la main sur mon script R ayant généré la figure en question, j'ai ré-écrit le programme correspondant. Le seul problème, c'est que cette nouvelle figure est un peu différente de la précédente et remet en cause les conclusions de l'article. Je ne comprends pas ce qui a pu se passer. J'aimerais changer mes pratiques pour rendre ma recherche plus reproductible, mais je ne sais pas par où commencer."

2.3 *Cindy D., stagiaire de Master en physique des matériaux*

"Il y a 3 semaines, j'ai commencé mon stage de Master 2. En fait, j'ai plutôt l'impression d'avoir pris l'autoroute de la souffrance. Ma principale occupation a été d'extraire des données à partir d'une série d'articles, qui donnait les points de caractéristique dans les PDF en fichiers supplémentaires : du *fun* à tous les étages. Je les copie-colle directement dans Excel. *Fun*². Dans le tableur, je dois ensuite transformer les"." en ",". Comme je ne dispose pas des incertitudes d'estimation pour tous les articles

(seuls certains articles les incluent dans leurs résultats), je n'en tiens pas compte et ne rentre que les estimations.

C'est un travail assez rébarbatif (faites un stage qu'ils disaient), mais c'est le seul moyen de pouvoir ensuite analyser les données de la littérature. Lorsque ma collecte sera terminée,~~je fête ça avec des amis~~ je calculerai le coefficient R^2 pour mes données et mon directeur espère pouvoir publier nos résultats."

2.4 *Long N., enseignant-chercheur en biologie*

"Pour interpréter mes données expérimentales sur les protéines fluorescentes, je me suis lancé dans la simulation moléculaire et j'ai installé un code réputé dans la littérature. La fluorescence est en compétition avec des mouvements intramoléculaires assez rares. Pour avoir une chance de les observer et de faire une statistique, je dois faire des simulations avec des pas de temps assez longs.

En analysant les résultats, j'ai détecté un mouvement intéressant, mais le pas de temps est trop long pour en observer les détails. J'ai donc repris les données stockées par le programme juste avant cet événement et j'ai relancé la simulation avec un pas de temps plus fin. Je n'ai jamais pu reproduire cet événement. En faisant de la bibliographie, j'ai réalisé que ce type de simulation était affecté par du "chaos numérique". Je n'ai plus confiance dans mes résultats."

2.5 *Mohammed B., ingénieur d'étude en calcul scientifique*

"En tant qu'ingénieur du labo, je suis reponsable de la maintenance du logiciel *pytR*, développé il y a 15 ans par un postdoctorant très doué. Il a depuis quitté le labo et personne ne sait

vraiment comment modifier le code de ce logiciel alors qu'une bonne partie de notre activité de recherche repose sur son utilisation. J'ai collé pas mal de rustines qui nous ont permis de tenir un certain temps face à l'évolution de nos infrastructures et de nos systèmes. Mais depuis la mise à jour de notre parc informatique il y a 6 mois, ce n'est plus possible. J'ai été obligé de garder une machine sous l'ancien OS pour pouvoir continuer à faire tourner le logiciel. Mes collègues n'ont pas l'air trop inquiets, mais ça m'angoisse parce que nous avons cumulé une grosse dette technique. Je ne sais pas ce qui va se passer quand cette vieille bécane va nous lâcher !"

2.6 *Christina Z., directrice de recherche*

"Je dirige une équipe de recherche depuis 3 ans et j'encadre actuellement 2 doctorants et 3 post-doctorants. Je suis un peu inquiète car l'un de mes doctorants a soutenu sa thèse la semaine dernière et part en post-doc aux États-Unis à la fin du mois. Ses derniers écrits sont très prometteurs, mais il reste 50% du travail à faire et l'article n'est pas encore rédigé. J'ai prévu de recruter un stagiaire pour prendre la suite mais cela risque de prendre énormément de temps : alors que l'exploitation des résultats est particulièrement délicate sur ce projet, ce doctorant documente très peu les étapes de son travail en dehors de ses manips. Pour aggraver la situation, tout doit être bouclé d'ici la fin de l'année car le financement du projet arrive à échéance.

Récemment, j'ai aussi aussi reçu des nouvelles d'un ancien camarade de thèse dont un article important s'est fait rétracter. Certaines conclusions de son papier ont été attaquées. Il n'a pas pu fournir les données expérimentales qu'il avait utilisées : l'article date d'il y a 5 ans et il a perdu toute trace du doctorant qui avait conduit les manips. L'éditeur de la revue a rétracté l'article, faute d'éléments tangibles pour faire valoir un éventuel droit de

réponse. La réputation de mon ancien camarade en a pris un coup.”

3

Symptômes d'une recherche non reproductible

Il existe de nombreuses situations où l'on souhaite reproduire des résultats de recherche :

— on peut vouloir vérifier que la méthode mise en œuvre il y a quelques mois par un stagiaire, par un doctorant, ou par soi-même, donne les mêmes résultats avant de poursuivre l'étude ;
— on doit obéir (hé oui, c'est bien la dure vie de chercheur) aux demandes d'un relecteur de modifier une figure ou de tester l'impact de paramètres non envisagés dans l'étude initiale ;
— on souhaite vérifier que les méthodes "maison" font aussi bien, voire mieux, que celles des équipes concurrentes.

Nous sommes persuadés que vous avez d'autres situations en tête. Le point commun de toutes ces situations est qu'elles peuvent être l'occasion de surprises (très) désagréables que nous appellerons les "symptômes d'une recherche non reproductible". En complément des *personas*, nous vous livrons ci-dessous quelques témoignages, certes fictifs mais non moins vraisemblables, qui illustrent les dits symptômes.

3.1 J'ai perdu mes données ou mon code de programmation !

Après la publication d'un des mes articles, un collègue souhaite collaborer avec moi pour tester de nouvelles hypothèses sur le jeu de données que j'ai présenté. Malheureusement :

— le disque dur sur lequel j'archivais les données brutes a crashé, ou bien,
— j'ai effacé (vous repassez par la case départ) les données qui pre-

naient beaucoup de place sur mon ordinateur, puisque l'article était publié.

Que pourrais-je répondre à ce collègue tout en gardant un semblant de dignité ? Non reproductibilité : 1 / Tranquillité d'esprit : 0

3.2 Mes résultats ont changé !

Il y a quelques mois, j'ai soumis un article au journal. Depuis, j'ai continué de travailler sur mon code de traitement des données. Un *reviewer* me demande de compléter quelques figures, ce qui nécessite pour moi de faire de nouveau l'analyse des données. Malheureusement, la version actuelle du code, dans laquelle j'ai amélioré les algorithmes, ne donne plus tout à fait les mêmes résultats que ceux de l'article.

Dois-je annoncer au *reviewer* qu'une partie des résultats a changé ?

3.3 Mon code ne marche plus !

Ayant réussi à décrocher une ANR, je m'offre un nouvel ordinateur, avec la version la plus récente du système et des logiciels. Pour profiter au mieux des performances de cette machine, je recompile avec enthousiasme mon code de simulation : ma simulation prendra dix fois moins de temps qu'avec ma précédente machine, me dis-je. Mon excitation retombe subitement en voyant que :

— le compilateur génère des erreurs, ou bien
— le code recompilé démarre et se plante après quelques secondes.

Ma première idée (un peu la mort dans l'âme) est de récupérer mon vieil ordinateur, mais l'informaticienne du labo l'a déjà reformaté pour le passer à un stagiaire. Et là je me dis : "Mais pourquoi moi ?"

3.4 Mon nouveau doctorant n'observe pas les mêmes effets que son prédécesseur !

L'an passé, un de mes doctorants a soutenu brillament sa thèse après avoir obtenu des résultats remarquables, que nous avons publiés dans un excellent journal. Il a trouvé un post-doctorat à l'étranger sur un sujet sensiblement différent pour élargir le spectre de ses compétences. Mon nouveau doctorant doit repartir de ces résultats pour améliorer l'efficacité de notre processus. Cela fait maintenant un an qu'il échoue à reproduire les observations de son prédécesseur, alors qu'il suit scrupuleusement (et j'ai vérifié) le protocole établi. Dois-je mettre fin à sa thèse pour incompétence, le lancer sur un autre sujet et abandonner cet axe de recherche, ou envisager de rétracter l'article de son prédécesseur car tout d'un coup pris d'un certain doute ?

3.5 Mais fallait-il vraiment écrire toute cette tartine ?

Pour mon stage de M2, je souhaite montrer un lien entre la délétion d'un gène chez la souris et la survenue de problèmes neurolocomoteurs. J'ai vu de nombreux articles évaluer la présence de ces problèmes chez la souris, mais aucun ne m'a particulièrement attiré. En revanche, toute cette revue de la littérature m'a donné une super idée pour une telle évaluation comportementale, reposant sur un protocole d'observations certes complexe mais que je jugeais génial. Je n'ai pas voulu perdre de temps (nous ne pouvions bénéficier de ce lot de souris que pendant un mois). J'ai obtenu d'excellents résultats et les ai présentés en réunion d'équipe, tout excité à l'idée de rédiger mon premier article que j'allais soumettre dans une excellente revue. Après ma présentation, mon chef d'équipe m'a demandé de lui montrer le descriptif rédigé de mon protocole si "génial". Je lui ai répondu que je n'avais pas voulu prendre le temps de rédiger quelque chose qui était dans ma tête : quel intérêt d'écrire toute cette tartine pour soi, quand tout est

si clair dans sa tête ? Il m'a répondu "Ok, donc tu peux oublier ton premier papier sur tes souris".

Deuxième partie

Sources de non reproductibilité

4

Acquisition de l'information

Dans une démarche de recherche, la première étape est bien souvent l'aquisition d'information, que ce soit à partir de la collecte de nouvelles mesures expérimentales ou à partir de données déjà publiées.

4.1 Absence de standardisation de la collecte des informations

Dans la grande majorité des cas, la production de résultats issus d'une recherche passe par la collecte d'informations. Ces informations sont recueillies sur des "unités" (une unité pouvant être une pièce mécanique, un être vivant, *etc*.). Ce que l'on entend par "informations" sont les caractéristiques de chaque unité qui fait l'objet de la recherche ; par exemple : la résistance à la traction d'un matériau, la concentration en glucose d'une personne atteinte de diabète, ou bien encore son âge, son poids, *etc*.

Si cette collecte des informations n'est pas standardisée, la personne qui collecte des informations sur un individu/unité un jour n°1 ne le fera potentiellement pas de la même façon le jour n°2 si elle doit réitérer l'opération. Or, si elle ne procède pas à l'identique, la valeur de l'information du jour n°2 sera différente de celle du jour n°1, non pas parce que l'information a changé au cours du temps (ce qui est possible, et éventuellement acceptable – *cf*. ci-dessous), mais parce que la méthode a changé.

Ainsi, comment s'assurer d'une recherche reproductible si celle-ci se fonde sur des informations dont la valeur varie en fonction des modalités de collecte appliquées ? Quelle peut-être la valeur, voire la fia-

bilité, des données issues d'un mode de collecte qui n'est pas stabilisé pendant toute la durée de l'étude ?

4.2 Modification des données après une première collecte

Un autre problème conduisant à une recherche non reproductible se produit lorsque des informations recueillies sur une unité sont modifiées après une première collecte, sans que ces modifications ne soient tracées. Dans ce cas-là, les analyses statistiques qui seront conduites sur les informations modifiées ne fourniront évidemment pas les mêmes résultats que celles conduites sur les informations initiales. S'il n'y a aucun moyen de revenir aux informations initiales et/ou de savoir quelles sont les informations qui ont été modifiées, votre recherche devient par conséquent non reproductible.

Les solutions pour éviter de perdre ainsi la trace de la modification sont évoquées dans le chapitre 9.

4.3 Une collecte d'informations non répétable ou non reproductible

Dans cette section, en vue d'examiner les impacts de l'étape de la collecte de données, nous allons utiliser une définition particulièrement précise des termes "reproductibilité" et "répétabilité", en utilisant leur définition métrologique.

4.3.1 Quelques définitions issues du *Vocabulaire International de Métrologie*

Nous avons décidé de choisir les définitions proposées en 2012 dans la dernière version du *Vocabulaire International de Métrologie (VIM)* (**?**) car

elles représentent l'effort le plus récent de normalisation dans ce domaine ; document téléchargeable ici : (**?**).

La **fidélité** de mesure s'entend comme : "l'étroitesse de l'accord entre les indications ou les valeurs mesurées obtenues par des mesurages répétés du même objet ou d'objets similaires dans des conditions spécifiées."

La **répétabilité** est la fidélité de mesure dans les conditions de mesures suivantes : "conditions qui comprennent la même procédure de mesure, les mêmes opérateurs, le même système de mesure, les mêmes conditions de fonctionnement et le même lieu, ainsi que des mesurages répétés sur le même objet ou des objets similaires pendant une courte période de temps."

La **reproductibilité** est la fidélité de mesure dans les conditions de mesures suivantes : "conditions qui comprennent des lieux, des opérateurs et des systèmes de mesure différents, ainsi que des mesurages répétés sur le même objet ou des objets similaires."

4.3.2 Impact d'une absence de répétabilité ou de reproductibilité dans la collecte des informations

Si la collecte d'une information n'est pas "répétable" au sens du *VIM* défini ci-dessus (**?**), les conditions d'une recherche reproductible ne peuvent alors pas être remplies : vous n'obtiendriez pas les mêmes résultats à partir d'informations collectées sur des unités identiques, évalués dans les mêmes conditions par un même opérateur. Si la collecte des données n'est pas "reproductible" d'un opérateur à un autre au sens du *VIM*, les conditions d'une recherche reproductible ne sont pas non plus remplies : personne d'autre que vous ne pourrait obtenir les mêmes résultats sur des unités identiques évaluées dans les mêmes conditions.

4.4 Collecte des données à plusieurs

Supposons que vous ne soyez pas la seule ou le seul à collecter les informations pour votre étude. Deux questions se posent alors :

— la première, déjà abordée ci-dessus, concerne la standardisation de la collecte des informations : si cette collecte n'est pas standardisée, votre collègue et vous n'obtiendrez potentiellement pas les mêmes valeurs des informations collectées lorsque vous évaluez pourtant les mêmes unités.
— la seconde concerne l'outils de partage de l'information : dans quel document, sur quel support, allez-vous collecter les données, pour garantir que vous et votre collègue n'allez pas effacer ou affecter les informations collectées par l'autre ?

4.5 Collecte des données de la littérature

Nous envisageons maintenant le cas d'une étude qui dépend d'informations collectées dans la littérature. Dans ce genre de cas, une intervention manuelle est souvent nécessaire pour constituer la base de données.

Considérons d'abord le cas, fréquent dans certains domaines (et *a priori* favorable), où les données d'intérêt sont dans le fichier PDF d'un article ou de son supplément. Lorsqu'on effectue un copier/coller d'une partie de fichier PDF vers un éditeur de texte, les sources de contrariété sont multiples et dépendent largement du logiciel utilisé pour afficher le fichier PDF. Les désagréments les plus courants sont :

— une impossibilité éventuelle de gérer correctement des tables complexes avec des cellules vides ou une table pivotée ;
— la présence de renvois bibliographiques sur certains éléments du tableau ;

— la gestion du signe moins (" - "), qui est souvent récupéré comme un tiret (" — ") ou demi-tiret, ne pouvant alors pas être interprété par les codes de calcul.

Après extraction des données, une étape de correction manuelle est donc souvent indispensable et constitue en elle-même une source potentielle d'erreur, en plus de ne pas toujours être effectuée de façon traçable. La récupération de données à partir d'images (OCR) présente des problèmes similaires.

Et pour le chercheur aventureux, copier/coller les données collectées dans un tableur peut introduire une couche supplémentaire de surprises (transformation de nombres ou d'identifiants en dates, par exemple) (**?**).

4.6 Que faire?

Les solutions pour faire face aux problèmes évoqués dans ce chapitre, dépendent du collecteur de données, mais également de l'émetteur.

Le collecteur de données pourra se reporter aux solutions présentées dans les chapitres 9 et 13 pour automatiser et tracer au maximum le processus de collecte, d'autant plus que le volume de données est important et/ou si la tâche est répétitive.

L'émetteur de données pourra se reporter aux solutions présentées dans les chapitres 10 et 11 dédiés aux bonnes pratiques d'archivage de données dans des formats ouverts et lisibles par la machine.

NB : en tant que chercheurs, nous sommes souvent émetteurs de données pour d'autres chercheurs. De fait, nous devrions intégrer autant que possible cet aspect dans nos bonnes pratiques de partage de résultats.

5

Gestion des données

La perte de données à tous les niveaux d'un processus de recherche est une cause majeure de non reproductibilité. Cela peut aller du simple accident matériel, comme par exemple le *crash* d'un disque, au problème de méthode, comme l'absence d'une politique de sauvegarde ou de règles élementaires de documentation (métadonnées).

5.1 Intégrité et curation des données

Voici un *scenario* catastrophe classique quand il est question d'intégrité des données : alors qu'un éditeur vous demande de mettre à disposition les données brutes sous peine de ne pas publier votre article pourtant accepté, les données associées ont été effacées ou égarées. Quel que soit le degré d'intégrité scientifique du chercheur, si des doutes sur la validité des données émergent, ne pas être en mesure de fournir les données constitue pour lui un handicap difficilement surmontable. Par ailleurs, l'absence de sauvegarde des données est considérée comme une négligence professionnelle.

Il existe des *scenari* encore plus insidieux où l'intégrité des données peut être compromise sans que vous vous en rendiez compte. Par exemple : vous recevez vos données avec une certaine précision mais vous sauvegardez ces données avec une précision moindre. Vous serez alors confronté à une perte d'information irréversible : une partie de l'information s'est littéralement évaporée. De même, dans le cas de résultats produisant un déluge de données (comme par exemple le *Large Hydron Collider*) et devant l'impossibilité de tout sauvegarder, il faut sélectionner les données à sauvegarder, sachant que les autres seront

irrémédiablement perdues. Une mauvaise décision initiale peut se révéler catastrophique pour peu que vous ayez besoin de ces données à une étape ultérieure.

Enfin, si vous ne vous êtes pas assuré du contrôle d'accès sur vos données, quelqu'un peut venir les modifier par inadvertance et à votre insu, changeant ainsi les conclusions de vos analyses.

5.2 Traçabilité de la source des données

Quand bien même l'intégrité des données aurait été assurée, l'absence d'information descriptive sur la source des données (métadonnées) peut causer de nombreux problèmes. Vos données sont disponibles mais impossible de comprendre ce qu'elles représentent exactement. Par exemple : des données sont collectées dans la littérature, mais les références bibliographiques ne sont pas mentionnées ou s'avèrent lacunaires. Un problème pouvant être perçu comme formel constitue en réalité un manque de traçabilité portant atteinte à la reproductibilité. Le bibliothécaire avait donc raison.

5.3 Indexation des données

Lorsque vous manipulez de très larges volumes de données (en termes de nombre d'échantillons) il devient tout à fait possible de perdre, non pas les données, mais l'accès à ces données. Imaginez : vous avez utilisé un nommage particulier des fichiers pour indiquer par exemple la nature de la donnée (*well done*!) mais vous avez égaré le fichier expliquant les règles de nommages (*too bad*). Alors que vous possédez l'intégralité de vos données, vous vous trouvez incapable les utiliser.

5.4 Codages et unités

Lorsque vous sauvegardez des données sur un support informatique, il est important de comprendre qu'un certain nombre de choix sont effectués de façon automatique et sans possibilité de contrôle de votre part. Ces choix dépendent étroitement de l'architecture matérielle de votre ordinateur. Par exemple, en ce qui concerne la représentation des nombres en virgule flottante, certaines machines vont lire la représentation binaire de gauche à droite alors que d'autres le feront de droite à gauche (*endianess*). Si vous travaillez toujours avec le même type de machine, vous n'aurez pas de problème jusqu'au jour où vous changerez de machine et observerez alors des valeurs complètement erratiques, vous laissant à penser que vos données auront été compromises.

Plus généralement, stocker ou transmettre des données numériques sans en préciser les unités ni les conventions de codages associées constitue un vecteur important de risques, notamment si un tiers désire les réutiliser. Cela fut le cas pour la sonde "*Mars Climate Orbiter*" qui s'est désintégrée à la surface de Mars en raison d'une communication entre un système de mesure anglo-saxon (émission) et un système métrique (réception) (**?**).

5.5 Obsolescence des données

Dans certains cas, les données ont été sauvegardées, leur intégrité est parfaite, on peut les retrouver très facilement et pourtant, elles s'avèrent inutilisables. Comment expliquer ce paradoxe ? Les données sont généralement sauvegardées dans un format pouvant être ouvert ou fermé (propriétaire). Or si le format est fermé, vous ne pouvez pas contrôler l'évolution de ce format. Prenez par exemple un fichier Word créé il y a une vingtaine d'années, pouvez-vous encore le lire aujour-

d'hui ? Votre version de Word vous assure-t-elle une compatibilité avec ce format obsolète ? Vous avez répondu par la négative à ces questions ? Considérez alors les données comme inutilisables.

5.6 Que faire ?

Privilégier des formats ouverts (chapitre 10), assurer un archivage pérenne des données et leur associer des métadonnées pertinentes sur des serveurs institutionnels ou publics (chapitres 11 et 17) constitue actuellement l'une des meilleures manières de se prémunir contre la perte de données.

6

Programmation et calcul

Les problèmes inhérents au calcul et aux codes associés partagent des similarités avec les difficultés liées aux données, par exemple la non disponibilité. Toutefois, les questions de calcul ont leurs spécificités du fait de leur nature opératoire : il s'agit d'exécuter ce code afin d'obtenir un résultat. Or, c'est lors de cette étape d'exécution que vont surgir un certain nombre de complications que l'on peut classer en deux grandes catégories :

— d'une part, celles qui empêchent d'obtenir un résultat
— d'autre part, celles qui rendent un résultat différent voire faux.

Si le premier type de problème est ennuyeux (euphémisme), le second type de problème est d'autant plus grave qu'il est difficile à détecter (effroi intense).

6.1 Le code n'est pas disponible

En guise de préambule, débutons par une liste non exhaustive des cas où l'on n'a tout simplement pas ou plus accès au programme à exécuter :

— **Les logiciels propriétaires ou la loterie de la licence d'exploitation** : votre équipe/structure a cessé de payer la licence. Variante : ce logiciel est disponible dans l'université d'un collègue mais pas dans l'établissement où vous travaillez actuellement. Autre variante : vous avez accès au logiciel, mais seul un nombre restreint de personnes peut y accéder en même temps,

via un système de jetons. De fait, vous vous retrouvez à devoir attendre un bon moment avant d'y arriver.

— **Un seul code vous manque et tout est dépeuplé** : le code a été développé "en interne". Il arrive trop souvent qu'à la suite d'un *crash* disque, d'un vol d'ordinateur portable, du départ du développeur principal, que l'on n'ait simplement plus accès au logiciel. C'est souvent le résultat d'une politique (ou d'une absence de politique) de sauvegarde ou de partage d'informations au sein d'une équipe.

— **Le numéro que vous avez demandé n'est plus attribué** : assez souvent, il s'agit d'un code développé "en externe" (dans une autre équipe de recherche par exemple) que l'on souhaite réexécuter, par exemple pour avoir un point de comparaison ou bien pour vérifier si on obtient bien des résultats similaires avec une autre méthode. En général, on cherche alors le code sur le web mais il est assez courant que l'URL indiquée dans l'article ne soit plus accessible car le développeur a depuis quitté l'équipe où il travaillait et que sa page web a été supprimée ou complètement restructurée. Ce problème est connu sous le nom d'*URL decay* (**?**) ou de *Link Rot* (**?**).

— **Cachez ce code que je ne saurais voir** : enfin, les auteurs du code peuvent tout simplement ne pas souhaiter le partager, par exemple parce qu'ils jugent qu'il n'est pas montrable en l'état (pas ou peu commentaires, structure horrible cachant des erreurs) ou encore pour conserver ce qu'ils considèrent comme un avantage compétitif.

Si cette question vous intéresse, vous pouvez lire les travaux de Collberg, Proebsting et Warren : (**?**) (**?**). Les auteurs étudient les causes d'incapacité à réexécuter du code dans la communauté de recherche *Computer Systems*, pourtant très au fait des aspects logiciels. Vous y trouverez de nombreux témoignages (assez drôles si c'était sans conséquences !) issus d'une étude de terrain ; vous pourrez notamment lire les excuses les plus couramment utilisées pour justifier une incapacité à donner accès au code derrière une publication. Vous pouvez aussi consulter "Re-run, Repeat, Reproduce, Reuse, Replicate : Transforming Code into Scientific Contributions"(**?**).

6.2 Comment lance-t-on ce code ? ("Allô Houston ?")

Lorsque l'on fait de la recherche, il est courant de devoir développer soi-même un code pour répondre à un besoin spécifique. Que ce soit un "gros" code ou un petit script, on prend rarement le temps de rédiger une documentation "externe" (à destination des utilisateurs) puisque le code est principalement utilisé par les membres de l'équipe que l'on croise tous les jours. Mais lorsque l'on revient quelques mois plus tard, pour ré-exécuter un de ses propres calculs ou bien que l'on essaye de repartir du travail de quelqu'un d'autre (qui a quitté le laboratoire ou n'y a même jamais travaillé), il est courant de ne pas (ou plus) savoir comment il avait été lancé. Avec quels paramètres, quels fichiers d'entrées, quelles variables d'environnement, *etc.* ? La moindre erreur sur les paramètres conduira à des résultats différents voire à un *crash*. Et malheureusement pour vous, le "vous" d'il y a 6 mois ne répond pas au *mail*. Enfin, et comme nous le verrons par la suite, il existe bien d'autres raisons qui peuvent conduire à ces deux symptômes.

6.3 Comment fonctionne ce code ? *Lost in translation*

Si on n'est plus sûr des paramètres utilisés, on peut vouloir chercher à comprendre d'où vient le problème en inspectant le code … si tant est qu'on ait accès au code source bien sûr. Or, les logiciels propriétaires ou les logiciels disponibles uniquement sous forme binaire rendent toute inspection de ce type impossible. Mais admettons que vous ayez réussi à inspecter les sources et que vous ayez les compétences pour le comprendre (*a minima*, un langage de programmation que vous connaissez).

Les codes de recherche, développés pour des besoins spécifiques, sont souvent des prototypes et il est rare de prendre le temps de rédiger une documentation "interne" (à destination des développeurs). Et quand

bien même il y aurait des commentaires, encore faut-il qu'ils soient compréhensibles donc *a minima* en anglais. Par ailleurs, ils doivent aussi correspondre à la réalité : quand un code évolue vite, on ne prend pas toujours le temps d'actualiser au fur et à mesure les commentaires et la documentation. Si ces critères ne sont pas réunis, les commentaires risquent davantage de vous induire en erreur que de vous aider.

Un célèbre dicton en informatique dit : "*Programs must be written for people to read, and only incidentally for machines to execute*." C'est une citation d'Harold Abelson [1] tirée de son livre *Structure and Interpretation of Computer Programs* publié en 1979 (**?**). Commenter, c'est une chose, mais lorsque l'on cherche à comprendre un programme, on se rend vite compte qu'il est indispensable que les noms de variables et de fonctions aient été bien choisis, que le code ait été été bien structuré avec des fonctions au rôle clairement défini, sans quoi le code devient totalement incompréhensible (ce qui est précisemment l'objet du concours "Obfuscated C" (**?**). De même, lorsque qu'il s'agit d'un code conséquent réparti dans de nombreux fichiers, une mauvaise convention de nommage des fichiers ou bien l'usage d'une structure de fichiers absconse empêchent toute tentative de compréhension.

Enfin, même si le code est relativement compréhensible, il est possible que des *bugs* (des erreurs de programmation) soient à l'origine de vos malheurs mais comment les trouver ?

6.4 Quelle version du code ?

Nul n'est parfait et les *bugs* sont donc courants, même chez les programmeurs les plus chevronnés. Il se peut que le *bug* à l'origine de vos problèmes provienne de la version d'un logiciel actuellement installé sur la machine. Pour corriger ce *bug*, on peut vouloir mettre à jour le logiciel. Mais quelle version a été utilisée dans le passé et quelle est la version actuelle ? Et comment savoir si c'est effectivement la cause de la différence observée ? La mise à jour n'introduirait-elle pas de nou-

1. https://en.wikipedia.org/wiki/Hal_Abelson

veaux *bugs* ? L'idéal serait peut-être de revenir à une version plus ancienne mais comment faire ? Quelle est la version la plus récente que je puisse utiliser ?

Enfin, cette nouvelle version sera-t-elle toujours compatible avec mon ordinateur ? Et si je repars du code source, arriverai-je à le recompiler ?

6.5 L'environnement de calcul ou le paradigme des poupées russes diaboliques

Plus le langage que vous utilisez est de haut niveau, plus il est probable qu'il dissimule une grande complexité. Même le script le plus anodin dépend en général d'une large hiérarchie de bibliothèques que l'on a du mal à imaginer. À titre d'exemple, lorsqu'en Python vous souhaitez faire un petit graphique, il est courant de charger la bibliothèque `matplotlib` avec un simple :

```
import matplotlib
```

Or cette bibliothèque est fournie par un "paquet" qui, sur la machine d'un des auteurs s'appelle, `python3-matplotlib`. Lorsque nous cherchons à en savoir plus sur ce paquet, voilà ce que nous obtenons :

```
Package: python3-matplotlib
Version: 2.1.1-2
Depends: python3-dateutil, python-matplotlib-data (>= 2.1.1-2),
python3-pyparsing (>= 1.5.6), python3-six (>= 1.10), python3-tz,
libjs-jquery, libjs-jquery-ui, python3-numpy (>= 1:1.13.1),
python3-numpy-abi9, python3 (<< 3.7), python3 (>= 3.6~),
python3-cycler (>= 0.10.0), python3:any (>= 3.3.2-2~), libc6 (>=
2.14), libfreetype6 (>= 2.2.1), libgcc1 (>= 1:3.0), libpng16-16 (>=
1.6.2-1), libstdc++6 (>= 5.2), zlib1g (>= 1:1.1.4)
```

C'est ici la version 2.1.1-2 qui est présente et, pour l'installer, il a fallu installer les paquets `python3-dateutil`,

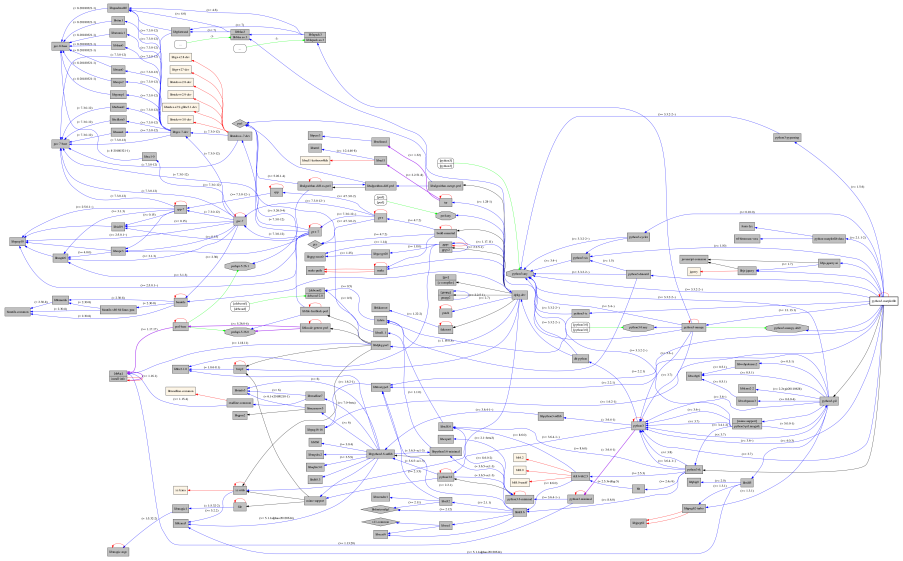

FIGURE 6.1 : Dépendances de Matplotlib sous Debian obtenues avec debtree

`python-matplotlib-data`, `python3-pyparsing`, *etc*. C'est ce qu'on appelle les "dépendances". Mais pour ces paquets dépendent eux-mêmes d'autres paquets. Lorsque l'on récupère l'ensemble des paquets nécessaires avec leurs dépendances, voici le graphe qu'on obtient, *cf*. Fig. 6.1 (alerte paracétamol) :

Vous remarquerez dans les dépendances que la version n'est pas précisément indiquée mais qu'il faut par exemple une version supérieure de `python3-pyparsing` qui soit au moins 1.5.6. Mais si des *bugs* peuvent être introduits, comment être sûr que votre code fonctionnera de la même façon avec les versions `1.5.6`, `1.5.7`, ..., sachant que nous en sommes maintenant au moins à la version `2.2.0`.

En résumé, tout code, aussi petit soit-il, possède tout un arbre de dépendances qui sont le plus souvent cachées. Ce code s'exécute donc dans un environnement donné et une différence, même insignifiante, de cet environnement peut conduire à des résultats différents, c'est-à-dire à des problèmes de non reproductibilité.

Si vous pensez que ce problème n'est que théorique, nous vous invitons

à lire Gronenschild et ses co-auteurs (**?**) qui étudient l'influence de la version de MacOSX et de FreeSurfer, un logiciel permettant de mesurer l'épaisseur corticale et le volume de structures neuroanatomiques.

6.6 Le chaos numérique

Les nombres manipulés par ordinateur ne sont pas des nombres réels, avec une précision infinie, mais des nombres dits "à virgule flottante" qui n'obéissent pas exactement aux mêmes règles que celles que l'on nous enseigne à l'école. Par exemple, si vous demandez à, à peu près n'importe quel ordinateur si `0.1*3==0.3` ou si `3-2.9==0.1` il vous répondra très certainement `FALSE` (Faux) dans les deux cas. Cela est dû au fait que la représentation au format binaire de ces nombres, en apparence simple, n'est pas exacte. Beaucoup de machines à calculer ont une représentation interne en base 10 un peu différente, or, nous n'avons pas été habitués de manière précoce à intégrer ce genre de problème, sauf peut-être pour des nombres du genre $1/3 \approx 0,3333333$. Lorsque l'on programme, il faut donc faire très attention à cette subtilité qui joue des tours dès que l'on veut comparer deux nombres.

Un autre problème au prime abord surprenant, mais probablement plus simple à comprendre, est la non associativité des opérations. Si avec les nombres réels, il va de soi que $(a + b) + c = a + (b + c)$, ce n'est pas le cas avec les nombres en virgules flottantes. Par exemple, `(1e-10+1e10)-1e10` vaut 0 alors que `1e-10+(1e10-1e10)` vaut `1e-10`. Même une simple moyenne peut donc devenir problématique et, non, il ne suffit pas de trier les nombres avant de les additionner pour résoudre le problème.

Comme vous utilisez vraisemblablement un ordinateur parallèle (même votre téléphone a maintenant plusieurs cœurs de calcul), il est possible que la somme $a_1 + \cdots + a_n$ ne soit pas calculée comme vous l'imaginez (i.e., $(((((((a_1 + a_2) + a_3) + \cdots + a_n))$, mais en plusieurs parties (i.e., $((((a_1 + a_2) + \cdots + a_{n/2}) + ((((a_{n+1} + a_{n+2}) + \cdots + a_n))$, chaque cœur de votre processeur réalisant une des sommes partielles,

la somme finale étant faite à la fin. Les résultats peuvent changer par le simple fait de passer d'un ordinateur à un autre alors que les machines semblent avoir le même environnement. Les ordinateurs peuvent ne pas avoir exactement le même nombre de cœurs. Les cœurs d'un ordinateur n'allant pas toujours exactement à la même vitesse, un code un peu optimisé ajustera la taille des sommes partielles pour terminer le calcul le plus rapidement possible et le résultat du calcul variera donc d'une exécution sur l'autre alors que rien n'a changé ! Mais alors, comment décider lequel de ces différents résultats de calculs est le "bon" ?

Toutes ces petites imprécisions de calcul peuvent hélas rapidement devenir très problématiques lorsque le système sous-jacent correspond par exemple à la discrétisation d'une équation différentielle. Le calcul est alors très sensible aux conditions initiales et l'accumulation des imprécisions peut amener à une catastrophe (voir notamment *The Patriot Missile Failure* (**?**)).

Il y a de nombreux articles décrivant ce genre de cauchemars : (**?**) (**?**)

Vous pouvez vouloir lire le classique *What Every Computer Scientist Should Know About Floating-point Arithmetic* [2] (**?**) ou encore les travaux de Stodden et ses collègues (**?**). Pour une présentation de ces problématiques et de quelques solutions, vous pouvez aussi vouloir regarder ce séminaire sur la reproductibilité numérique (**?**).

6.7 Que faire ?

Pour résoudre ces problèmes, des solutions sont abordées aux chapitres 12, 14, 15 et 16.

2. https://www.itu.dk/~sestoft/bachelor/IEEE754_article.pdf

7

Communication des résultats

Nous allons voir dans ce chapitre qu'une recherche peut devenir non reproductible s'il existe une mauvaise utilisation des résultats de l'étude au moment de la rédaction d'un article : il peut s'agir d'une mauvaise sélection de votre part des résultats, d'un choix inapproprié du format de présentation ou d'une transformation de ces données.

7.1 Une mauvaise sélection des résultats

Les résultats fournis par un logiciel peuvent contenir de si nombreuses informations qu'il faille opérer une sélection parmi elles. En d'autres termes, vous pouvez être amené à devoir identifier les informations pertinentes pour la question de recherche faisant l'objet de votre article.

Confronté à cet amoncèlement, il peut vous arriver de mal sélectionner l'information pertinente : votre sélection à la souris a "oublié" quelques caractères en début ou en fin de séquence à sélectionner, par exemple. En outre, si cette information est complexe et difficilement compréhensible par vos collaborateurs parce que vous êtes seul spécialiste du domaine, alors cette erreur de sélection sera répercutée dans l'article et persistera après le processus de revue si les relecteurs ne la repèrent pas.

7.2 Transformation des résultats

Une autre erreur pouvant survenir à partir des résultats fournis par un logiciel est celle d'une "transformation" (bien entendu involontaire) de ces données. Cette modification délétère peut très facilement se produire si vous devez saisir de nouveau dans votre article les sorties résultats du logiciel. Une erreur de frappe est si facilement arrivée !

Un deuxième exemple de transformation des résultats est celui de l'amélioration d'une figure. Si vous trouvez que la figure que vous obtenez à partir d'un logiciel n'est pas satisfaisante, vous pouvez décider de la retravailler manuellement, par exemple en ajoutant une courbe interpolant des points, alors que les valeurs intermédiaires n'ont pas de sens. Cette manipulation est dangereuse, car le traitement de l'information n'est plus produit par une chaîne d'instructions validée et automatisée, mais par vous-même – et vous n'êtes pas infaillible.

Une transformation involontaire des résultats peut aussi se produire si vous collaborez sur une étude et si vous devez intégrer les résultats d'analyses réalisées par votre collaborateur, sans en comprendre toutes les subtilités. Un exemple que l'on rencontre malheureusement fréquemment dans la littérature concerne les résultats d'analyses statistiques. Vous avez demandé à votre collègue, spécialiste des statistiques, de traiter certaines de vos données, et il vous envoie ses résultats que vous ne comprenez pas dans le détail. Il peut alors facilement arriver que, par défaut de compréhension, vous retranscriviez mal ou partiellement les résultats dans l'article. De telles erreurs de saisie peuvent passer totalement inaperçues si les relecteurs n'ont pas non plus les compétences statistiques requises pour interpréter ces résultats.

7.3 Présentation des résultats

Une forme très courante de perte d'information est liée à l'absence ou à la forme inappropriée des incertitudes associées aux résultats d'une mesure physique ou virtuelle, ou d'une étude statistique. Très souvent, l'absence d'incertitude (par exemple, l'absence de valeur d'écart-type) empêche une comparaison de résultats, ou bien l'absence de matrice de corrélation entre les paramètres incertains d'une étude empêche leur réutilisation. Consulter : "How Measurements of Rate Coefficients at Low Temperature Increase the Predictivity of Photochemical Models of Titan's Atmosphere" (**?**).

Même lorsque des efforts ont été faits pour publier les informations adéquates, des problèmes d'arrondi peuvent anéantir la réutilisabilité des données. Une mauvaise sélection du nombre de chiffres significatifs à reporter dans un résultat numérique peut tout à fait se produire. Pour une question de "présentation" (taille d'une table de résultats), vous pourriez juger qu'un seul chiffre significatif peut suffire. Mais si vos résultats sont nécessaires à la réalisation d'autres recherches, pour permettre par exemple des simulations, les erreurs engendrées dans ces autres travaux pourront être amplifiées : une petite erreur initiale peut conduire à une erreur très importante en bout de course – cf. « Chaos numérique » dans le chapitre 6.

A titre d'illustration, la matrice de variance-covariance publiée par le CODATA en 2002 pour l'ajustement des constantes fondamentales, arrondie pour être présentable dans les annexes de l'article, s'est avérée inutilisable pour des travaux ultérieurs (**?**).

7.4 Que faire ?

La réutilisabilité des résultats d'une étude doit être une priorité. Pour cela, les données doivent être mises à disposition de futurs utilisa-

teurs (Chapitre 11), dans un format lisible par la machine (Chapitre 10), en utilisant un processus automatisé limitant les interventions manuelles (Chapitre 13).

Troisième partie

Solutions de la recherche reproductible

8

Le temps des changements ?

Nous espérons que les anecdotes du chapitre 2 vous ont semblé vraisemblables, mais vous souhaitons de ne pas trop vous reconnaître dans ces situations de crise ! Nous essayons de vous proposer d'intervenir de manière curative, mais aussi préventive.

8.1 Quand mettre en œuvre les bonnes pratiques ?

Il existe des solutions applicables à court comme à plus long terme. Si le déploiement de certaines solutions nécessite des compétences techniques avancées, la reproductibilité de la recherche demeure peut-être avant tout une question culturelle, dans la mesure où il s'agit de faire évoluer des pratiques parfois solidement ancrées. La crainte de perdre du temps en modifiant ses repères est une problématique majeure, d'où l'approche pragmatique du présent ouvrage. En outre, comment accompagner l'adoption de nouvelles pratiques à l'échelle collective, qu'il s'agisse d'un laboratoire ou d'une communauté disciplinaire ? Par exemple, pour un directeur d'unité ou d'équipe, il pourrait être tentant d'imposer tout ou partie des méthodes décrites en vue d'une plus grande efficience, mais faire évoluer les pratiques quotidiennes de ses collègues constitue un exercice délicat (**?**) (**?**). De fait, une approche très progressive qui valorise l'implication directe des individus a davantage de chances d'aboutir (ou moins de risques d'être rejetée, selon que vous voyez le verre à moitié plein ou à moitié vide).

Au fil de l'ouvrage, nous décrivons des situations de crise, moins par goût pour les histoires d'horreur que par conviction que l'iden-

tification des problèmes couramment rencontrés constitue l'une des meilleures options pour aborder la question de la reproductibilité.

8.2 Concrètement, que changer et comment s'y prendre ?

Bonne nouvelle : il est très peu probable que votre recherche soit à 100% non reproductible. La palette des solutions à votre disposition est variée ; en fonction de vos compétences et de vos urgences, vous pouvez choisir d'agir sur :

— la collecte et la gestion des données,
— le code et sa robustesse,
— les environnements logiciels,
— les sauvegardes,
— les versions et les archives,
— les licences.

Bonne nouvelle bis : le présent ouvrage a été conçu pour être hautement "*cherry-picking-proof*". Selon l'état de vos pratiques actuelles, les chapitres peuvent donc être lus dans l'ordre qui vous conviendra le mieux, même si nous conseillons bien sûr de tous les lire, voire de diffuser cet ouvrage auprès de vos collègues et étudiants (fin de la minute d'autopromotion).

9

Documenter ses pratiques

Cette partie expose une sélection de solutions aux problèmes soulevés dans le chapitre 4.

9.1 Rédiger un protocole de collecte des informations

Comme nous l'avons vu plus haut (chapitres 4 et 5), l'information doit être collectée de façon "standardisée". Cela implique de rédiger un protocole décrivant la façon dont l'information est collectée, et ce, donnée par donnée. Par exemple : chez une personne atteinte d'un diabète, vous recueillez la concentration du glucose dans le sang. Vous devez dès lors rédiger le protocole permettant d'obtenir la valeur de la concentration en glucose, en précisant notamment dans le protocole le volume de sang prélevé, à quel moment de la journée est-il prélevé, *etc*. Ce protocole doit non seulement être rédigé mais il doit par ailleurs être approuvé par votre équipe de recherche, ou *a minima*, par les personnes de votre équipe disposant des compétences pour juger de la qualité de ce protocole.

9.2 Partager le protocole de collecte des données

Idéalement, le protocole de collecte des informations devrait faire partie de l'article publié qui relate les résultats de votre recherche. Dans un monde idéal encore, les revues devraient systématiquement de-

mander aux auteurs de rédiger une telle partie dans l'article afin que d'autres chercheurs puissent éventuellement confirmer les résultats de votre étude. A noter : il ne s'agit pas forcément de rédiger *in extenso* le protocole de collecte des informations dans l'article. Il s'agit plutôt de se demander si, sur la base de votre descriptif, un tiers pourrait appliquer votre protocole de collecte de données et obtenir *in fine* des résultats similaires aux vôtres.

9.3 Tenir un cahier de laboratoire

Modifier des informations ne constitue pas en soi un problème. Les difficultés surviennent lorsque ces modifications n'ont pas été documentées, tracées. Une solution pour pallier ce problème consiste à rédiger un cahier de laboratoire. A l'origine, un cahier de laboratoire est un document physique dans lequel on consigne toute modification d'information au stylo (indélébile) : on indique la raison de la modification de la valeur d'une information, la date de la modification, et l'auteur de la modification. Même si le cahier physique a tendance à disparaître des laboratoires au profit de carnets de laboratoire numériques, la logique reste la même : toute modification des informations initiales doit être renseignée, en indiquant la raison de la modification, la date, et l'auteur de la modification.
Le tableur est-il mon ami pour effectuer ce type de tâche ? Pas vraiment. Plutôt pas du tout, dans la mesure où la modification d'une valeur dans une cellule peut être réalisée sans laisser de trace.

9.4 Collecter les données de façon répétable ET reproductible

Le protocole de collecte d'une information doit être défini de telle sorte qu'elle soit "répétable" et "reproductible" au sens du *VIM* (**?**) (cf. chapitre 4). Tout d'abord, si vous collectez une information deux fois sur

un individu dans un intervalle de temps restreint, les deux valeurs de l'information doivent être les mêmes, ou du moins être suffisamment voisines pour être "considérées" comme identiques (condition de répétabilité). La collecte des informations doit être reproductible entre opérateurs (on parle de "reproductibilité inter-opérateurs"). Pour garantir une bonne répétabilité et reproductibilité inter-opérateurs, il faut donc avoir standardisé au maximum le protocole de collecte des données, en s'appuyant sur un instrument de mesure capable de recueillir les informations avec le minimum d'erreurs possibles. Des indicateurs statistiques, tels que les coefficients de concordance, permettent de quantifier la "répétabilité" ou la "reproductibilité inter-opérateurs" d'une méthode de mesure.

9.5 Pour en savoir plus

Pour en savoir plus sur les cahiers de laboratoire, nous vous invitons à consulter le site du CNRS (**?**). Pour en savoir plus sur la quantification de la répétabilité et de la reproductibilité de la façon dont on collecte les données, vous pouvez consulter le document suivant : "Guide pratique de validation statistique de méthodes de mesure : répétabilité, reproductibilité, et concordance" (**?**).

10

Formater et structurer l'information

Dans ce chapitre, nous allons traiter des formats et structures de fichiers numériques, bien que ces concepts aient aussi leur importance en dehors de l'outil numérique.

10.1 Comment structurer mes informations ?

Les chercheurs peuvent être amenés à travailler sur des données de nature très variée. Si on peut spontanément penser à des nombres, les chercheurs peuvent aussi travailler sur des images ou du texte. Le plus souvent, les chercheurs travaillent en fait sur une "collection" de données liées les unes aux autres. La relation entre ces données est essentielle.

Par exemple, il peut être commode de rassembler des informations sur des patients sous forme d'une "table" comportant une ligne pour chaque individu et une colonne pour chaque type d'information (nom, âge, sexe, nature du médicament administré, dose, taux de glycémie après 1 heure, taux de glycémie après 5 heures). Dans ce cas, le nom et le type de chaque colonne sont souvent considérés comme des "métadonnées". Ce n'est pas la seule façon de représenter ce type d'information. On pourrait préférer avoir une entrée (une ligne) pour chaque mesure (et non pour chaque patient) en ajoutant une colonne indiquant quand la mesure a été réalisée, et une autre indiquant par qui elle a été réalisée.

Un autre exemple de structure est celui d'un arbre généalogique. Au prime abord, il semblerait naturel d'utiliser une "hiérarchie", mais la

vie étant faite de surprises (décès, divorces, re-mariages, voire mariages entre cousins, "Luke, je suis ton père", *etc*.), cette représentation va rapidement s'avérer inadaptée pour concevoir un arbre généalogique clair.

Un dernier exemple d'information dont la conservation est primordiale : il s'agit du protocole expérimental généralement consigné dans un cahier de laboratoire. On structure souvent cette information de façon chronologique avec des annotations sémantiques : qui ? quand ? où ? pourquoi ? dans quel contexte ? *etc*. Dans tous les cas, la question du lien entre ce cahier et les données archivées doit être posée et résolue.

Bref, même si une table, une hiérarchie ou un texte libre peut sembler la solution la plus naturelle, il est vraiment important de bien réfléchir à la façon la plus adaptée de structurer vos informations en fonction du traitement que vous allez vouloir réaliser car cela risque de considérablement affecter ce que vous allez pouvoir faire de vos données.

10.2 Quel format choisir pour enregistrer et stocker des informations ?

L'enjeu autour du format pour la recherche reproductible est double :

- assurer l'interopérabilité,
- minimiser les risques d'erreur de manipulation.

Le chercheur doit donc avoir en tête les bonnes pratiques établies dans sa communauté et s'assurer que son choix de représentation permette, voire facilite, la réutilisation de ses données et de ses résultats. Le nom des fichiers est également un point important du formatage et sera abordé dans le chapitre 12.

La recherche reproductible vise à limiter drastiquement les interventions manuelles dans le flux de production des résultats. Dans le choix d'un format d'enregistrement et de stockage des informations, l'objectif est de garantir la "lisibilité par la machine". On devrait donc dire

"lisible par toutes les machines" avec en tête, les spécificités des différents systèmes d'exploitation qui peuvent devenir problématiques pour certains formats.

On distingue 3 grands types de formats :

— les formats fermés/propriétaires pour lesquels le risque de perte de lisibilité n'est pas maîtrisé par l'utilisateur, et qui nécessitent que d'autres disposent également du logiciel nécessaire (parfois coûteux) pour pouvoir réutiliser les données.
— les formats codés, illisibles par l'humain, tels que les formats binaires ou de description de page tel que le PDF, qui nécessitent une étape de décodage, et qui peuvent parfois mal supporter la transition entre les systèmes d'exploitation et les architectures matérielles.
— les formats texte (tels que .csv pour les tableaux) qui sont lisibles par les humains comme par les machines, très interopérables, et dont les modifications peuvent être enregistrées par les outils de suivi de version (voir le chapitre 12).

Par exemple, pour des tables de données simples, mieux vaut privilégier les formats .csv ou .tsv plutôt que les versions plus ou moins propriétaires ou spécifiques à un tableur (.dot, .xls, .xslx, ...) qui peuvent parfois contenir des informations très difficiles à lire pour la machine (cellules colorées, cellules fusionnées, *etc*.).

Dans la mesure du possible, nous préconisons de privilégier les formats texte lisibles à la fois par l'humain et par la machine et d'éviter les formats propriétaires et codés. Toutefois, ce n'est pas toujours possible. Dans ce cas, il est préférable d'utiliser les standards de sa communauté plutôt que des formats exotiques.

Pour des données plus complexes, hétérogènes, de type hiérarchique, des formats adaptés, ouverts, et interopérables existent comme par exemple .yml, .json, ou .xml pour des formats textes et .hdf5 ou .fits ou pour des formats binaires.

10.3 La présentation des résultats numériques

Lors du stockage de données numériques, il est primordial d'éviter la perte ou l'érosion de l'information. Ceci implique, outre une documentation exhaustive précisant les unités et la provenance des résultats, de gérer correctement leur représentation numérique (**?**).

10.3.1 Nombre de chiffres significatifs

Les calculs numériques sont effectués avec une précision finie, et il faut donc choisir le nombre de chiffres significatifs à reporter dans une table de données ou de résultats. Dans un fichier de résultats, il est tentant d'inclure tous les chiffres significatifs dont on dispose. Mais cela n'est pas necessairement souhaitable (cela peut conduire à une inflation inutile des tailles de fichiers), d'autant plus que ce n'est pas nécessairement la précision dont on dispose réellement : par exemple, R n'affiche pas tous ses chiffres significatifs avec sa commande `print()`.

10.3.2 Incertitude

Les informations devraient idéalement toujours être accompagnées d'une incertitude. Cela s'applique à la fois aux mesures (qu'elles soient physiques ou virtuelles), ainsi qu'aux résultats d'analyse (par exemple des estimations) (**?**).

L'incertitude peut servir de guide pour choisir le nombre de chiffres significatifs. Par exemple la recommandation en métrologie (**?**) est d'arrondir (par excès) l'incertitude à deux chiffres significatifs, et de reporter le résultat au même niveau décimal. Par exemple, si le résultat de mesure vaut 1.23456789 et l'incertitude vaut 0.00456, on reportera 1.2346 avec une incertitude de 0.0046. En outre, on évitera dans un tableau les notations du type 1.2346(46) ou 1.2346 ± 0.0046, qui peuvent fragiliser la lecture automatique par une machine.

Une attention particulière doit être portée à certains objets afin de respecter leurs propriétés intrinsèques. Par exemple, les éléments d'une matrice de variance-covariance doivent être arrondis de manière à s'assurer que celle-ci reste définie-positive (en exigeant par exemple que la plus petite valeur propre de la matrice garde deux chiffres significatifs). Voir "Definition" (3.21)(**?**).

Les valeurs et les incertitudes devraient être présentées dans des colonnes séparées.

10.4 Pour en savoir plus

En ce qui concerne la structuration des données, une approche assez populaire consiste à utiliser le plus possible une structure de tableau. Nous vous recommandons de lire ce document sur le sujet : *Tidy data* (**?**).

11

Partager ses données, codes et résultats

La mise à disposition des données de la recherche est un point clé pour leur réutilisabilité et pour faciliter la reproductibilité de la recherche par vos pairs. Il existe plusieurs façons de les mettre à disposition. Pour un partage efficace, il est important de choisir :

— une licence adaptée (chapitre 17),
— un format *ad hoc* (chapitre 10).

Il convient aussi d'élaborer une documentation extensive levant les ambiguïtés d'interprétation (chapitre 14), et enfin de choisir un mode de partage le plus ouvert et pérenne possible.

11.1 *Compendium* de recherche

Avant de présenter différentes modalités de partage, intéressons-nous à la notion de *compendium* de recherche. Il s'agit d'un outil qui facilite la recherche reproductible en rassemblant dans un même "lieu" virtuel les données, les codes, les protocoles et la documentation liés à un projet de recherche.

La façon la plus simple d'élaborer un *compendium* de recherche est de créer un répertoire associé au projet, avec des sous-répertoires dans lesquels sont répartis les objets. Une convention de nommage explicite des objets et répertoires peut grandement faciliter la réutilisabilité. Par exemple, pour une simple analyse statistique d'un jeu de données, Marwick et ses co-auteurs proposent la structure suivante (**?**) :

Image CC-BY par Ben Marwick, Carl Boettiger & Lincoln Mullen.

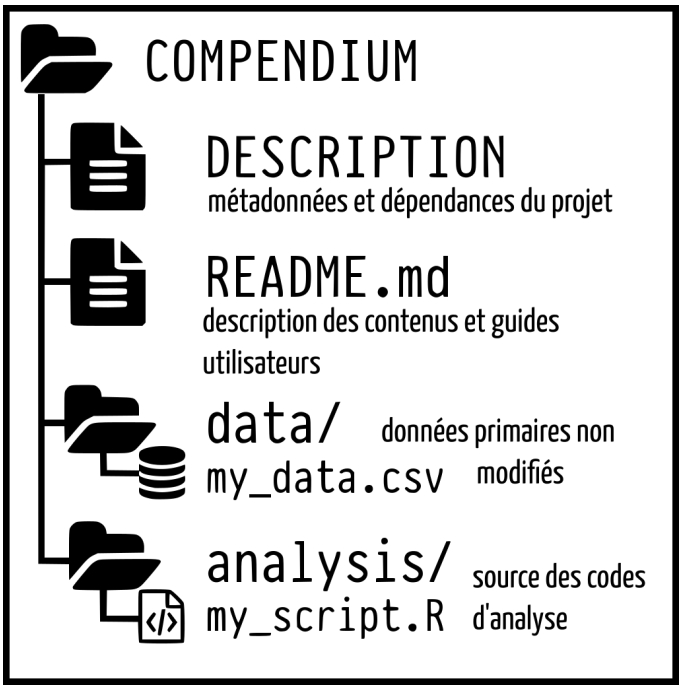

FIGURE 11.1

Ces *compendia* peuvent être alors archivés et référencés comme tout autre objet numérique sous forme de dossiers compressés ou sur des plates-formes spécialisées comme par exemple l'*Open Science Framework (OSF)* [1] (**?**).

11.2 Modalités de partage

Nous considérons dans cette partie que vos données peuvent être légalement ouvertes au public.

1. https://osf.io/

11.2.1 Sur demande

Le degré zéro (voire -1) de la mise à disposition des données pour la communauté, est de mentionner dans l'article qu'elles sont disponibles "Sur demande à l'auteur".

Cette approche permet à l'auteur de garder une forme de contrôle, mais elle s'avère peu adaptée à une gestion sur le long terme (ex. : départ à la retraite de l'auteur, changement d'adresse de courriel, *etc*.) et tend à disparaître. En effet, de nombreuses plates-formes éditoriales adoptent progressivement une politique d'ouverture des données et demandent aux auteurs de les mettre à disposition de leurs pairs. L'efficacité des dispositifs éditoriaux déployés est variable, comme le soulignent Stodden et ses co-auteurs (**?**), mais on note une évolution progressive des pratiques.

11.2.2 Page web personnelle

Créer une page web personnelle pour faciliter l'accès aux données est une solution simple et un premier pas dans la bonne direction, mais ici encore, de nombreux aléas risquent de compromettre la continuité de service. Si cette solution est envisagée, et nous ne vous la conseillons pas, l'utilisation d'une page web institutionnelle, intégrée au site d'un laboratoire offrira une visibilité accrue et une meilleure pérennité.

11.2.3 Hébergement par un tiers

Il existe des solutions permettant de partager un répertoire entier, la plus populaire étant Dropbox [2] ou son équivalent libre Framadrive [3]. Le niveau le plus élevé de granularité de ces outils est le fichier. C'est la raison pour laquelle, il existe des solutions plus spécifiques selon les formats de fichiers utilisés. Par exemple lorsque l'on travaille sur des données de type tableur, une solution populaire est Google Spread-

2. https://www.dropbox.com/
3. https://framadrive.org/login

sheet [4] (resp. Google Docs [5]) ou (mieux) son équivalent libre Framacalc [6] (resp. Framapad [7]).

Si ces services sont très pratiques et très faciles à utiliser, ils posent un certain nombre de problèmes dans un contexte recherche :

— Certaines informations sont confidentielles (données de santé, collaborations avec des partenaires industriels, idées ayant vocation à être brevetées, *etc*.). Déléguer la responsabilité du contrôle d'accès à un hébergeur extérieur (qu'il soit privé ou associatif) peut poser de sérieux problèmes. Il convient donc d'utiliser les équivalents institutionnels et à demander leur déploiement s'ils n'existent pas.
— La pérennité sur le long terme des outils comme des services n'est absolument pas garantie. On peut citer les cas de Gitorious et de Google Code, deux services d'hébergement de code dont la popularité n'a pas empêché la fermeture. En déléguant la responsabilité du stockage à un hébergeur extérieur, vous prenez un risque conséquent. Vous pouvez utiliser ce type de service mais il faut absolument vous assurer que vous disposez d'une solution d'archivage en complément.
— La gestion de versions est automatique mais très peu contrôlée. Dropbox par exemple ne conserve que quelques versions intermédiaires et ce, uniquement pendant une durée assez limitée dans le temps (quelques mois). Et comme la synchronisation avec votre propre ordinateur est automatique, il n'est pas rare de perdre des données au bout de quelques temps avant d'avoir eu le temps de réaliser quoi que ce soit. L'exemple le plus classique dans un contexte non professionnel est celui du partage d'un dossier : votre beau-père efface un jour les photos dudit dossier afin de faire de la place sur son disque dur personnel. Par ricochet et sans avertissement, les données sont ainsi effacées de votre propre ordinateur.

4. https://spreadsheets.google.com/
5. https://docs.google.com/
6. https://framacalc.org/
7. https://framapad.org/

11.2.4 Archives pérennes

Il existe actuellement de nombreuses plates-formes permettant d'héberger "gratuitement" les données de la recherche. Nous avons évoqué dans le chapitre 12 des solutions pour le suivi de version (GitLab, GitHub, ...) qui permettent une forme d'archivage. La pérennité de cet archivage est meilleure qualité que celle des solutions précédentes mais n'est pas parfaite non plus. D'autre part ces systèmes, initialement prévus pour héberger du code, gèrent assez mal les gros volumes de données et les données de type binaire (images bitmap, vidéogrammes, codes compilés...).

Des solutions privées de type Figshare [8], ou institutionnelles comme Zenodo [9], offrent un archivage sur le long terme pour tous types de données et fournissent un DOI permettant leur référencement stable dans la littérature. Pour les logiciels, une archive existe : Software Heritage [10]. En tandem avec un système de gestion de versions, ces archives permettent de conserver de façon pérenne et d'indexer les versions successives les artefacts de la recherche.

11.3 Pour en savoir plus

Vous pouvez consulter les recommandations du CNRS [11] (**?**) concernant la traçabilité des activités de recherche, ainsi que les principes FAIR (Findable, Accessible, Interoperable, Reusable) [12] (**?**).

Pour une exposition très claire des problèmes de pérennité, vous pouvez regarder le séminaire de Roberto Di Cosmo [13] sur ce sujet *in* "*Series*

8. https://figshare.com
9. https://zenodo.org
10. https://www.softwareheritage.org
11. http://qualite-en-recherche.cnrs.fr/spip.php?article315
12. https://www.go-fair.org/fair-principles/
13. https://github.com/alegrand/RR_webinars/blob/master/5_archiving_software_and_data/index.org

of webinars and documents on Reproducible Research - Preserving software : ensuring availability and traceability" (**?**).

12

Versionner, versionner, versionner

La sauvegarde des étapes successives du développement de documents (textes, bases de données, codes, *etc*.) est une des briques fondamentales de la recherche reproductible. Elle permet en effet de retrouver ou de reconstituer facilement des versions spécifiques et datées des documents. La mise en œuvre va de la simple sauvegarde de versions des fichiers, à des sauvegardes différentielles n'enregistrant que les modifications.

La mise en œuvre d'une stratégie de gestion de versions doit permettre d'éviter les cauchemars du type "Je ne retrouve pas mon code / mes données" et permet de répondre aux questions : quand ? qui ? pourquoi ?

Nous présentons deux solutions dans ce chapitre :

— la première s'appuyant sur la mise en place de convention de nommage des dossiers et fichiers,
— la seconde, plus technique présentant les outils de versionnage différentiel.

12.1 Versionnage par nommage de fichiers

Au cours des modifications d'un document, les étapes importantes devraient être sauvées explicitement. Cette approche nécessite des règles de nommage de fichiers évitant les ambiguïtés et facilitant l'exploration des versions.

Dans ce contexte, il est souvent recommandé de créer un nom de fichier constitué de plusieurs éléments (**??**) :

— la date de sauvegarde, sous format AAAAMMJJ ou AAAA_MM_JJ (permettant un tri des dates par ordre alphabétique)
— le titre du fichier, évitant les mots "vides" (article défini, indéfini, *etc*.)
— un numéro de version du document qui sera incrémenté aux étapes remarquables de l'évolution du document (v01, v02…)

Le titre ne devrait pas être trop long : sous certains système d'exploitation, il ne doit pas dépasser 31 caractères, extension comprise. Egalement pour des raisons de compatibilité, il est recommandé d'éviter l'utilisation de :

— signes diacritiques ; ex. : accents, trémas, cédilles,

— caractères spéciaux,
— espaces,
— *etc*.

Exemple

La chronologie du nommage de fichiers successifs de scripts R pourrait être :

— 20190417_analyse_donnees_v00.R
— 20190417_analyse_donnees_v01.R
— 20190509_analyse_donnees_v02.R

Ces informations sont une forme appauvrie de métadonnées : lorsque le format de données ou le système de fichiers utilisé ne permet pas de les conserver de façon pérenne, on indique directement les métadonnées dans le nom du fichier. Cette méthode fournit des informations rudimentaires : dans le cas où plusieurs personnes ont travaillé sur un même fichier, on a en effet du mal à savoir qui a fait quoi et pourquoi. Conserver les métadonnées de base *via* cette méthode est une bonne habitude à prendre, mais il s'agit d'une approche qui a ses limites.

La section suivante présente une approche plus aboutie de la gestion de versions, mais plus technique à mettre en œuvre.

12.2 Outils de versionnage différentiel

Le stockage des versions successives de fichiers entraîne un gaspillage d'espace de stockage et peut devenir très incommode pour la gestion de projets comportant de nombreux fichiers évoluant à des vitesses différentes. Il est en général plus efficace de passer par un logiciel de gestion de versions qui gère automatiquement le stockage de versions différentielles des fichiers, les métadonnées de type "qui, quoi, quand, pourquoi", et permet de figer des versions du projet à des étapes cruciales du développement.

Ces outils peuvent être intégrés à des logiciels (par exemple des éditeurs de texte tels que Word ou LibreOffice) ou se présenter comme des solutions autonomes (comme par exemple le logiciel Git) avec des interfaces en ligne de commande ou des interfaces graphiques (parfois intégrés directement dans les éditeurs de code tels que RStudio ou Emacs).

Si un logiciel de gestion de versions est installé sur l'ordinateur de l'utilisateur, il est prudent, voire impératif, de mettre en œuvre une synchronisation vers un dépôt distant, ce qui présente plusieurs avantages à la fois : sauvegarder les travaux, les partager, et collaborer. Ce dépot distant peut être sur une plate-forme institutionnelle (par exemple le GitLab de l'Inria [1] ou de l'IN2P3 [2] ou encore un plate-forme privée comme GitHub [3]). En outre, ces plates-formes permettent le travail collaboratif sur un projet ou de mettre le projet à disposition du public dans des versions bien définies des documents. Pour que la recherche soit reproductible, le lecteur devrait systématiquement disposer du lien vers la version des données ou des codes informatiques

1. https://gitlab.inria.fr/learninglab/mooc-rr/mooc-rr-ressources/gitlab
2. https://gitlab.in2p3.fr/CTA-LAPP/HiPeCTA
3. https://github.com/

utilisés dans un article. Pour cela, il est conseillé d'activer en plus un archivage pérenne des versions importantes du projet sur un dépôt fournissant un identifiant permanent (ex. DOI). Ce sujet sera abordé dans le chapitre 11. Dans tous les cas, avant de choisir une solution, il est essentiel de se renseigner sur les pratiques en cours dans votre communauté de recherche.

Exemple

Voici un témoignage d'un chercheur dont la pratique quotidienne permet d'obtenir aisément une recherche reproductible :

> "Lors de l'élaboration de mes analyses statistiques, je développe des codes en R pour traiter les données, générer des tableaux de résultats en LaTeX et des figures en png. En parallèle, je rédige l'article en LaTeX.
>
> Tous les codes et l'article sont suivis dans un gestionnaire de versions local (j'utilise Git). Les tableaux et les figures sont des résultats secondaires que je peux regénérer facilement. À chaque étape de modification d'un document, j'effectue un *commit* qui met à jour la base de version locale. Après un certain nombre de *commits*, je me synchronise avec le dépôt distant en y propageant mes modifications locales, m'assurant ainsi d'une sauvegarde régulière. Réaliser régulièrement ces opérations est l'occasion pour moi de bien réfléchir à ce que je fais, de vérifier que j'ai bien sauvegardé tout ce qu'il fallait, et d'en profiter pour expliquer dans les messages de *commits* ce que je fais.
>
> Lorsque je soumets l'article, je fige une version du projet, et je l'archive sur un dépôt pérenne pour obtenir un identifiant (DOI) que je référence dans l'article. Ainsi, les relecteurs peuvent s'assurer de la reproductibilité de mon étude et j'ai la garantie de disposer de la bonne version des fichiers s'ils me demandent des modifications, même si j'ai continué à développer mes codes durant ce temps."

12.3 Pour en savoir plus

Pour en savoir davantage sur la gestion de versions, vous pouvez vous reporter au module 1 du MOOC sur la recherche reproductible [4] (**?**) qui vous permettra de vous familiariser avec GitLab *via* une mise en condition réelle.

GitLab et GitHub sont des sur-couches Web du logiciel Git ; elles en permettent une utilisation simplifiée mais ne donnent pas accès à l'ensemble de ses fonctionnalités. Les éditeurs comme RStudio proposent une très bonne intégration *via* une interface graphique des interactions avec Git. Il est également possible d'utiliser la ligne de commandes pour bénéficier de toutes les fonctionnalités de Git. Voici quelques moyens d'apprendre à utiliser Git et d'aller un peu plus loin :

— Happy Git With R [5] (**?**)
— Le Software Carpentry Git tutorial [6] (**?**)
— Le livre Pro Git (gratuit) en anglais [7] (**?**) ou en français [8]. Les deux premiers chapitres suffisent pour bien commencer (**?**).
— Le site Apprenez Git Branching [9] (**?**) permet d'apprendre Git interactivement et de comprendre les branches.

4. https://learninglab.inria.fr/mooc-recherche-reproductible-principes-methodologiques-pour-une-science-trar
5. https://happygitwithr.com/
6. http://swcarpentry.github.io/git-novice/
7. https://git-scm.com/book/en/v2
8. https://git-scm.com/book/fr/v2
9. https://learngitbranching.js.org/

13

Apprendre à programmer

Dans ce chapitre, nous mettons en évidence les limitations des interfaces graphiques en termes d'efficacité et de reproductibilité ; nous expliquons qu'il peut être nécessaire d'apprendre à programmer en conseillant deux langages devenus incontournables en traitement et analyse des données : Python et R.

13.1 Limiter le recours aux interfaces graphiques

Une interface graphique est un moyen simple et visuel d'indiquer à un logiciel ce que l'on souhaite faire. Par exemple, dans un logiciel de traitement de texte de type WYSIWIG (*What You See Is What You Get*), à l'aide de la souris, on peut surligner un passage afin de le mettre en italique ou en gras en cliquant sur le bouton correspondant. Ce type d'interface intuitive est omniprésent dans les logiciels et simplifie la vie de nombreux utilisateurs.

Dans de nombreux logiciels scientifiques, ce type d'interface est également disponible, par exemple pour traiter les données ou concevoir des figures. L'exemple le plus célèbre est très certainement le tableur Excel, très utilisé y compris pour faire des traitements statistiques relativement compliqués. Dans un premier temps, cet outil peut simplifier les traitements mais dès que vous souhaiterez effectuer des tâches plus subtiles ou des tâches répétitives, il va rapidement devenir un obstacle. Si vous effectuez des calculs exotiques, vous serez alors contraint d'entrer des formules de calcul complexes dans les cellules. La bonne nouvelle est que, en faisant cela, vous avez commencé à programmer. La mauvaise est que votre calcul n'est pas reproductible.

"Mais j'ai sauvé ma feuille Excel pourtant!". Certes, mais entre le moment où vous avez chargé vos données initiales et le moment ou vous avez sauvé vos résultats, un certain nombre de manipulations que vous avez faites n'a pas été enregistré.

Par exemple, si vous devez trier vos données selon une certaine colonne, les données dans la feuille sauvegardée seront triées mais vous aurez perdu la trace de cette opération de tri et il vous sera facile d'oublier de la répéter si vous devez recommencer le traitement avec un autre jeu de données (sauf si vous êtes précautionneux et tenez scrupuleusement à jour votre cahier de laboratoire).

13.2 De l'intérêt de la programmation

Pour surmonter les difficultés liées aux interfaces graphiques, il est nécessaire de pouvoir exprimer d'une façon claire, documentée et non ambigüe ce que l'on souhaite faire faire au logiciel. C'est ici que la programmation entre en jeu. Plutôt que de cliquer à droite ou à gauche, on va expliquer textuellement, à l'aide d'un jeu d'instructions relativement restreint, ce que l'on souhaite faire. Cela veut dire, entre autres choses, qu'un programme peut être simplement exprimé en quelques lignes seulement. Plus le langage utilisé sera de haut niveau et moins il y aura à écrire.

Reprenons notre exemple où je dois charger des données, les trier selon la deuxième colonne et faire la moyenne sur la première moitié des données de la première colonne. Un façon de l'écrire est :

```
table <- read.csv(file = 'data.csv') # Charger les données en mémoire
table_tri <- table[order(table$age), ] # Réordonne toutes les lignes du tabl
# Sélectionne la première moitié des lignes de la première colonne
# pour en faire la moyenne
mean(table_tri$IMC[1:(nrow(table_tri)/2)])
```

En faisant abstraction de la syntaxe particulière (il s'agit ici du lan-

gage R) de ce programme simple, on comprend néanmoins à la lecture ce qu'il fait : (1) charger les données, (2) les trier, puis (3) calculer la moyenne. On a de fait simplement traduit le traitement que l'on souhaitait réaliser en jeu d'instructions. Par rapport à Excel, le programme offre deux avantages majeurs :

— d'une part, il effectue les traitements de manière reproductible,
— d'autre part, il documente cette chaîne de traitements, si bien que quiconque est maintenant en mesure de comprendre et de reproduire ces traitements, éventuellement à l'aide d'un autre de langage de programmation.

13.3 Le choix des armes

Apprendre à programmer n'est pas tout à fait équivalent à apprendre un langage de programmation. Il est nécessaire de comprendre au préalable ce que l'on peut attendre d'un programme et comment exprimer ce que l'on souhaite faire de façon non ambigüe car l'ordinateur est extrêmement bête (si, si) et ne fera que ce que vous lui demandez de faire, ni plus, ni moins. L'apprentissage de la programmation passe donc par la découverte d'un nombre limité de concepts que l'on va retrouver dans la plupart des langages de programmation. Notez que nous parlons de "langages" au pluriel pour refléter leur diversité. Les raisons de l'existence d'une telle variété de langages sont multiples. Heureusement pour nous, au cours de ces dernières années, deux langages se sont quasiment imposés dans le monde de la recherche : Python et R. Entendons-nous bien : nous ne disons pas que ce sont les seuls langages utilisables dans ce contexte (loin de là) mais néanmoins, ils ont su traverser les frontières de tous les champs disciplinaires. Il y a aujourd'hui des conférences scientifiques qui leur sont entièrement dédiées et où l'on peut voir un physicien théorique discuter avec un sociologue ou bien un doctorant enseigner à un professeur.

Pour commencer à programmer, nous ne saurions trop vous conseiller

de commencer par apprendre un de ces deux langages. Le choix entre les deux vous appartient, sachant que vous avez tout intérêt à observer les pratiques dans votre domaine ou bien encore à regarder les programmes des conférences telles que Scipy [1] (**?**), EuroScipy [2] (**?**), useR ! [3], Les Rencontres R [4] (**?**), *etc*.

13.4 Pour en savoir plus

Il existe une multitude de ressources pour apprendre à programmer en R ou en Python, notamment les formations continues proposées par les institutions (services de formation des personnels, écoles doctorales, mais aussi formations internes à votre unité de recherche). En raison de cette multitude de ressources, il est difficile de choisir lesquelles sont les plus pertinentes. C'est la raison pour laquelle nous donnons ci-dessous une liste très restreinte de ressources accessibles en ligne que nous jugeons de qualité.

13.4.1 MOOCs

— "Introduction à la statistique avec R" [5] (**?**)
— "Python : des fondamentaux à l'utilisation du langage" [6] (**?**)

13.4.2 Tutoriels

— *Begin'R* [7] (**?**)
— Le tutoriel officiel de Python [8] (**?**)

1. http://conference.scipy.org/
2. https://www.euroscipy.org
3. http://www.user2019.fr
4. https://r2018-rennes.sciencesconf.org
5. https://www.fun-mooc.fr/courses/UPSUD/42001S06/session06/about
6. https://www.fun-mooc.fr/courses/inria/41001S03/session03/about
7. http://beginr.u-bordeaux.fr
8. https://docs.python.org/fr/3/tutorial

13.4.3 Livres

— *R for Data Science* [9] (**?**)
— *Dive into Python* [10] (**?**)

13.4.4 Ateliers en ligne "Software Carpentry"

— "Programming with R" [11] (**?**)
— "R for Reproducible Scientific Analysis" [12] (**?**)
— "Programming with Python" [13] (**?**)
— "Plotting and Programming in Python" [14] (**?**)

9. https://r4ds.had.co.nz
10. https://www.diveinto.org/python3
11. http://swcarpentry.github.io/r-novice-inflammation
12. http://swcarpentry.github.io/r-novice-gapminder
13. http://swcarpentry.github.io/python-novice-inflammation
14. http://swcarpentry.github.io/python-novice-gapminder

14

Rendre son code compréhensible

> "*Programs must be written for people to read, and only incidentally for machines to execute*."
> – Harold Abelson

Les chapitres précédents (et en particulier le chapitre 6 "Programmation et calcul" (chapitre 6) ont souligné l'importance des questions de diffusion et de partage des codes informatiques. Une étape préalable à cette démarche est de rendre son code facilement compréhensible par un lecteur externe. Ce premier lecteur peut être un collègue, comme vous-même : ce qui était clair au moment de la rédaction l'est nettement moins après quelques semaines.

Il existe là encore différents degrés de complexité permettant de faciliter la compréhension de son code :

1. nommer ses variables et ses fonctions de manière informative
2. être explicite
3. commenter son code
4. documenter son code
5. utiliser des documents computationnels
6. restructurer et automatiser avec un *workflow*

14.1 Nommer ses variables et ses fonctions de manière informative

> "*There are only two hard things in Computer Science : cache invalidation and naming things*."
> – Phil Karlton

Une manière d'obtenir un code plus facile à comprendre par un observateur extérieur est d'utiliser des noms informatifs qui explicitent directement ce que représente une variable ou une fonction. C'est une tâche qui peut se révéler étonnamment ardue ! Lisez le code suivant et essayez de comprendre ce à quoi il peut bien servir par exemple :

```
ninja = 100
XX = 2.0000
a = 0.5
turtle = 3.2
bluE_Pizza = a * ninja * turtle ** XX
print(bluE_Pizza)
```

Vous avez trouvé ? Pas encore ? Cet exemple est créé de toutes pièces mais nous avons tous déjà rencontré des codes bien pires que ça. Ces quelques lignes sont pourtant rigoureusement équivalentes au code suivant :

```
mass = 100
speed = 3.2
energy = 1/2 * mass * speed^2
print(energy)
```

Dans le code qui précède, le nom des variables est parfaitement clair et le code se passe alors de commentaire.

Une autre manière de rendre son code plus lisible est de le modulariser en créant des fonctions aux noms explicites qui permettent :

— à la fois de le rendre robuste, en évitant les répétitions d'instructions et minimisant ainsi le nombre d'erreurs,
— mais aussi de le condenser et donc de le rendre plus lisible

Dans le cadre de cette modularisation, il vous faudra également bien réfléchir à la portée de vos variables (variable locale ou globale) et aux paramètres de vos fonctions afin d'éviter autant que possible les variables globales, les effets de bord étant une source de confusion et d'erreur inépuisable.

Pour limiter au maximum les erreurs dans un code, il faut donc essayer de l'écrire de façon à la fois claire, non ambigüe, commentée et concise. Lorsqu'on débute en programmation, c'est étonnamment difficile à faire car il faut s'adapter au style du langage utilisé. Il faut en effet intégrer les concepts essentiels du langage de programmation pour exprimer ce que l'on souhaite faire de la manière la plus élégante possible.

Nous parlerons dans le chapitre suivant de techniques et conseils pour rendre son code plus robuste. Mais avant même cette étape, nous pouvons déjà agir sur le processus d'écriture.

14.2 Etre explicite

Tim Peters a écrit *The Zen of Python* (que vous pouvez lire en saisissant `import this` dans une session Python) qui donne un ensemble de règles qu'il faut garder en tête lorsqu'on écrit un programme. Les six premières lignes sont :

Beautiful is better than ugly
Explicit is better than implicit
Simple is better than complex
Complex is better than complicated
Flat is better than nested
Sparse is better than dense

Ces six conseils peuvent paraître plus ou moins évidents à mettre en œuvre pour un débutant, mais focalisons-nous sur le deuxième. Il peut être mis en œuvre très simplement en refusant par exemple l'usage des "arguments par défaut". "Arguments par défaut" signifie que lorsque vous appelez une fonction qui nécessite normalement n paramètres, vous appelez celle-ci avec un nombre réduit $k < n$ paramètres, et les valeurs des n-k paramètres manquants sont complétées automatiquement avec celles définies par défaut dans la fonction. C'est bien pratique : vous économisez de l'écriture de code, mais en

vous reposant sur cette fonctionnalité, vous laissez implicitement à la bibliothèque utilisée le soin de définir à votre place la valeur des paramètres. La plupart du temps, déléguer votre responsabilité à un tiers ne pose pas de problème majeur… jusqu'au jour où, après une mise à jour, la valeur d'un de ces *n-k* paramètres par défaut est changée ! Vos résultats changent, alors que votre programme n'a pas changé d'un bit. Le problème aurait pu être évité : vous auriez pu expliciter l'ensemble de vos *n* paramètres, y compris les éléments non obligatoires car disposant d'une valeur par défaut.

14.3 Commenter son code

Commenter son code de manière pertinente est une tâche moins évidente qu'il n'y paraît. Il ne s'agit pas de décrire dans une langue comprise par les humains ce que le programme effectue, comme nous allons le voir. Considérons le programme suivant :

```
ninja = 100    # This is the mass. It is expressed in kilograms.
XX = 2.0000    # This is the exponent
a = 0.5        # This is a magical constant
turtle = 3.2   # This corresponds to the speed at which the ninja is moving
# Now I will use the famous kinetic energy formula, which is of course only
bluE_Pizza = a * ninja * turtle ** XX
print(bluE_Pizza) # Now, let's print it on the screen
```

Les commentaires ci-dessus n'aident pas vraiment à comprendre de quoi il retourne ; la version du code avec des noms de variables explicites est bien plus simple à comprendre. On pourrait toutefois objecter cet argument : un code bien écrit se passe de commentaires. Il s'avère que les commentaires de l'auteur du code s'adressent à son lecteur, censé connaître à la fois le langage utilisé et le contexte du logiciel. Les commentaires servent par exemple à signaler :

- les unités ou le domaine de définition d'une variable toujours positive,
- une ruse de calcul,
- un point un peu délicat qui devrait être amélioré,
- les conditions particulières dans lesquelles une fonction doit ou ne doit pas être utilisée,
- la raison de l'assignation d'une nouvelle valeur à une variable dans les données, *etc*.

Les commentaires attirent l'attention du lecteur et répondent de manière anticipée à une partie de ses questions.

Enfin, il est possible d'atténuer l'austérité de cette tâche en commentant son code au fur et à mesure de sa rédaction. Si commenter du code peut sembler à peu près aussi attractif que de se faire un tatouage sur le visage quand on ne souhaite pas embrasser une carrière dans le *hip hop*, on se rend pourtant compte de l'utilité de cette tâche une fois face à des lignes écrites plusieurs mois, voire plusieurs semaines auparavant.

14.4 Documenter son code

Le commentaire s'adresse au programmeur qui va chercher à comprendre ou à faire évoluer le code (*cf*. ci-dessus). La documentation s'adresse quant à elle aux utilisateurs du logiciel. Cela commence en général par un fichier `README` expliquant succinctement l'objectif du logiciel, comment l'installer, comment l'exécuter. Au fur et à mesure que le code évolue, sa documentation peut devenir plus conséquente et, dans ces cas, cette documentation s'intègrera souvent directement dans le code source grâce à des outils comme Sphynx (pour Python) ou Roxygen (pour R). Il est donc important de bien faire la distinction entre ces deux types d'annotations (commentaire et documentation) qui peuvent se retrouver simultanément à l'intérieur du code.

14.5 Utiliser des document computationnels (*notebooks*)

La programmation lettrée a été conceptualisée en 1984 par Donald Knuth (**?**) et propose des principes pour produire un code compréhensible. Il s'agit principalement de considérer l'écriture d'un code comme un moyen d'expliquer à d'autres personnes les tâches demandées à l'ordinateur. À l'époque, l'objectif était d'autoriser les développeurs à s'affranchir de l'ordonnancement imposé par l'ordinateur et de se concentrer sur leur pensée.

Dans le cas d'un chercheur, l'enjeu se concentre sur l'exécution du code : les données d'entrées et les résultats obtenus permettent d'enrichir la compréhension du phénomène étudié.

Les documents computationnels ou *notebooks* s'inspirent d'une certaine façon de la programmation lettrée et proposent une manière de travailler devenue très populaire parmi les chercheurs.

Les documents computationnels permettent d'intégrer dans un même document : du texte rédactionnel, du code informatique et les résultats de ce code. La partie narrative, rédigée dans un langage de balisage très léger (tels que Markdown [1] (**?**)), est régulièrement agrémentée de fragments de codes exécutables (par exemple en R ou en Python) dont les résultats textuels ou graphiques sont automatiquement accolés. Cette structure correspond assez bien à la démarche suivie quotidiennement par les chercheurs :

1. "Je propose une hypothèse que je décris" = partie narrative
2. "Je réalise une expérience/analyse" = j'exécute mon programme
3. "J'inspecte le résultat de mon expérience/analyse" = le résultat du programme
4. "J'interprète les résultats et je décris mon interprétation" = partie narrative avant de proposer une nouvelle hypothèse

1. https://daringfireball.net/projects/markdown

Pour compléter, voir diapo n° 37 *in* "La vitrine et l'envers du décor : Le document computationnel", par A. Legrand (**?**).

Ce procédé permet ainsi de documenter chaque étape de la recherche : chaque partie du code est gérée de manière indépendante et de fait, liée à sa finalité directe. Différents outils permettent d'écrire de tels documents, les plus matures étant :

— Jupyter [2]
— Rmarkdown [3] ; voir aussi (**?**)
— Org-mode [4]

De tels documents computationnels offrent de nombreux avantages :

— d'une part ils permettent une meilleure transparence du code effectivement exécuté,
— d'autre part, ils facilitent sa compréhension car le code devient une partie intégrante d'une trame narrative ; certains *notebooks* peuvent ainsi quasiment s'apparenter à des articles,
— enfin, ces documents peuvent répondre à des besoins de traçabilité en raison de leur caractère dynamique : on obtient en effet des cellules de codes prototypes qu'il est possible de réécrire et de réexécuter avec des paramètres différents.

Pour en savoir davantage sur les documents computationels, vous pouvez vous rapporter aux module 2 et 3 du MOOC sur la recherche reproductible [5] (**?**).

14.6 Restructurer et automatiser l'exécution du code avec un *workflow*

Nous avons déjà évoqué l'importance de modulariser son code pour le rendre plus compréhensible et plus facile à faire évoluer. Il existe de

2. https://jupyter.org
3. https://rmarkdown.rstudio.com
4. https://www.orgmode.org/fr
5. https://learninglab.inria.fr/mooc-recherche-reproductible-principes-methodologiques-pour-une-science-tran

nombreux concepts pour vous permettre d'y arriver : la programmation fonctionnelle, la programmation orientée objet, les *design patterns*, *etc*.

Dès lors qu'il est question de transformer des quantités importantes de données par des calculs complexes, le concept de systèmes de *workflows* scientifiques [6] (**?**) trouve toute sa pertinence. Il existe de nombreux *workflows* répondant aux besoins de différentes communautés scientifiques : astrophysique, génétique, *etc*.

Lorsque votre *notebook* se stabilise mais devient trop long et trop complexe, il sera certainement temps de le restructurer. Les *workflows* peuvent vous y aider et vous permettre de passer par la même occasion à une étape supérieure d'automatisation de son exécution : c'est d'ailleurs souvent la motivation principale des utilisateurs de *workflows*.

Les *workflows* sont également des éléments à archiver pour rendre ses travaux reproductibles.

14.7 Pour en savoir plus

Le sujet des *workflows* est développé dans la conférence en ligne : *Reproducible Science in Bioinformatics : Current Status, Solutions and Research Opportunities* [7] (**?**).

6. https://en.wikipedia.org/wiki/Scientific_workflow_system
7. https://github.com/alegrand/RR_webinars/blob/master/6_reproducibility_bioinformatics/index.org

15

Exactitude des codes

Dans ce chapitre, nous présentons quelques techniques standards pour se prémunir autant que possible des problèmes liés au code. Ces solutions ne pourront certifier la justesse de votre code, mais pourront néanmoins garantir un minimum d'exactitude.

15.1 Programmation en binôme

Vous avez des amis adeptes de la ligne de commande, vous souhaitez animer une soirée ou un repas familial ? Une technique de programmation relativement facile à mettre en œuvre consiste à coder en binôme : on parle de *pair coding*. Le *pair coding* est issu des méthodes dites "agiles". Il s'agit de s'asseoir à deux en face d'une machine : l'une des personnes conduit le développement en ayant le contrôle du clavier ; l'autre observe et commente ce qu'écrit l'autre afin de corriger d'éventuelles erreurs ou simplement, questionner la pertinence de telle ou telle ligne de code. Les deux personnes doivent changer de rôle fréquemment. Cette pratique permet de réduire les erreurs : d'une part, vous êtes deux à relire le code et d'autre part, vous devez être en mesure d'expliquer à votre partenaire ce que vous êtes en train de faire.

Expliquer oralement ce que vous pensez avoir écrit est une bonne façon de détecter vos erreurs.

15.2 Tester votre code

> "*I don't need tests; I have users*"
> – Karl Broman

Une deuxième façon de rendre votre code plus robuste est de le tester sur des cas concrets dont on connaît par avance le résultat. Par exemple, si je souhaite écrire une fonction qui fait la somme de deux nombres, je peux vérifier que la somme de 1 et 2 me retourne bien 3, que la somme de `a` et `-a` est bien zéro, *etc*. Ces tests peuvent être plus ou moins formels et le cadre conceptuel des **tests unitaires** permet de mettre en place cette pratique de manière rigoureuse. Il s'agit alors d'envisager des cas d'usage caractéristiques de chaque partie (unité) et fonctionnalité du code, afin de tester que ce dernier adopte bien le bon comportement face à des situations variées : par exemple, les erreurs sont bien renvoyées lorsque les arguments n'ont pas de sens. La difficulté dans l'écriture de ces tests est de prévoir les cas singuliers auxquels on ne pense pas forcément. Par exemple, dans le cas de l'addition que nous venons d'expliquer, nous avons proposé de vérifier que la somme de `a` et `-a` faisait bien `0`, mais que se passe-t-il si `a = inf` ? (la réponse est que la somme est alors être non définie (`NaN`))

Une autre modalité pour tester un code est de le diffuser au sein de votre communauté. Vos collègues ne manqueront pas de vous faire des retours sur tel ou tel cas limite et il faudra dans un premier temps modifier votre code, puis ajouter ce test spéficique à votre collection de tests unitaires.

15.3 Intégration continue

Idéalement, aucune modification de votre code ne doit empêcher de réaliser les tests unitaires. Par exemple, le code passe les 100 tests unitaires. Problème : après une modification, il n'en passe plus que 80.

On parle alors de “régression” : votre nouveau code est moins bon que la version d’avant. Il existe aujourd’hui des outils qui permettent d’automatiser la vérification des tests unitaires, en les évaluant :

— à chaque nouvelle version du code, ou
— à intervalles de temps réguliers (chaque nuit par exemple).

Ces solutions peuvent être locales, telles que Jenkins, ou deportées, telles que Travis CI [1] (**?**) ou Appveyor [2].

Ces outils d’intégration continue peuvent en outre servir à vérifier que le code se comporte correctement sous différents environnements logiciels (voir le chapitre suivant)

15.4 Pour en savoir plus

— “Ten quick tips for teaching programming”, (**?**)
— “testthat : Get Started with Testing”, (**?**)
— “A Beginner’s Guide to Travis-CI for R”, (**?**)

1. https://travis-ci.org/
2. https://www.appveyor.com/

16

Environnement logiciel

Attention : ce chapitre traite de sujets appelant des compétences techniques avancées. Nous vous proposons d'aborder la (complexe) question de l'environnement (ça pique mais ça compte).

L'environnement logiciel désigne l'intégralité des logiciels qui assurent la gestion de l'ordinateur et qui sont utilisés pour réaliser la recherche. Les chercheurs peuvent avoir tendance à sous-estimer l'impact de l'environnement logiciel sur leurs résultats. Pour une recherche reproductible, il existe trois enjeux importants autour de l'environnement logiciel :

1. identifier et décrire son environnement
2. permettre à quelqu'un d'autre d'utiliser un environnement **exactement identique**
3. construire une variation de cet environnement

Il n'existe ni solution simple, ni solution unique pour répondre à ces enjeux. Ce chapitre a pour vocation de fournir une description critique de ces solutions en évoquant notamment leurs limites respectives.

16.1 Identifier les dépendances : dessine-moi une dépendance

La manière la ~~plus simple~~ moins complexe de décrire son environnement logiciel est d'en identifier les dépendances. Mais cette simplicité n'est qu'apparente car l'arbre cache la forêt : chaque dépendance a elle-même une dépendance, qui a elle-même une dépendance, et ainsi de

suite… Cette tâche peut donc rapidement devenir difficile. Deux méthodes permettent d'aboutir à une telle identification :

1. À partir d'un langage interprété (tel que R ou Python), on peut effectuer cette "intro-spection" et lister les différentes dépendances, la liste des bibliothèques chargées avec leur numéro de version. En R, on peut par exemple utiliser la commande `sessionInfo` (ou encore `devtools::session_info`). Mais cette méthode relativement simple est surtout limitée et assez peu précise car elle ne signale que quelques unes des dépendances système de plus haut niveau. Elle correspond à la zone verte dans le schéma ci-dessous. Cependant, il est très facile d'inclure ces informations descriptives dans vos documents computationnels (*cf.* chapitre précédent 15 "rendre son code compréhensible"), et nous vous recommandons cette bonne pratique.
2. En utilisant le gestionnaire de paquets du système d'exploitation, on obtient la liste exhaustive de l'ensemble des applications installées (listant donc des applications non pertinentes pour la recherche ainsi que les doublons de versions successives). Elle correspond à l'ensemble du schéma ci-dessous (donc à TOUT le système d'exploitation).

L'identification des dépendances est donc soit lacunaire et imprécise (méthode n°1 ci-dessus), soit au contraire, illisible car trop étendue (méthode n°2 ci-dessus).

Entre ces deux extrêmes, il existe cependant un moyen d'identifier ces dépendances, notamment *via* des gestionnaires de paquets de langages interprétés tels que Packrat pour R.

Ainsi, permettre à un collègue de travailler avec un environnement logiciel similaire au sien s'avère une tâche ardue. Fournir une description *ad hoc* ne peut être une opération manuelle tant cette tâche est fastidieuse. De plus, cette description ne résiterait pas à l'absence de standardisation entre systèmes d'exploitation. Il s'agit donc d'une solution informative sur son environnement, mais qui ne permet pas de le reproduire en pratique.

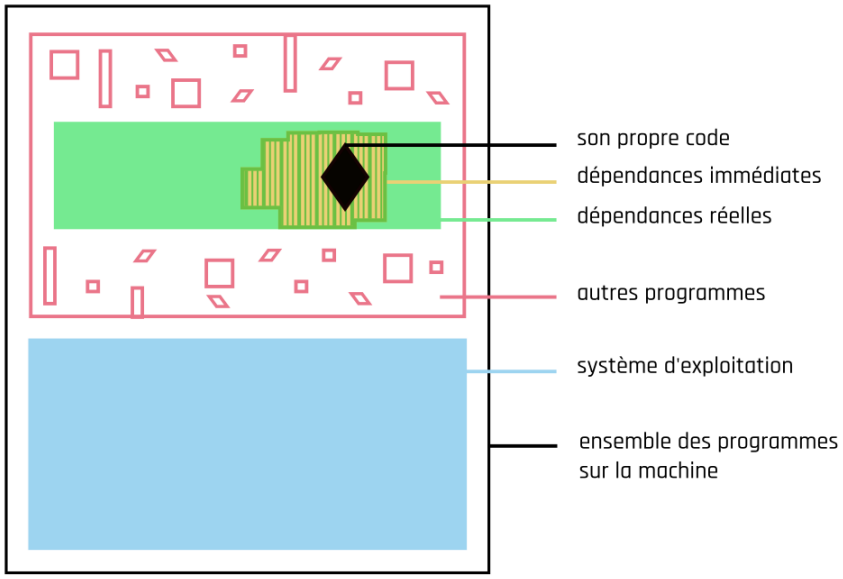

FIGURE 16.1 : Schéma des différents environnements logiciel

16.2 Préserver le désordre (*aka “Preserve the mess”*)

L’identification des dépendances permet de décrire son environnement, mais ne permet que très difficilement de le conserver ou le partager en pratique. Une façon de s’assurer de pouvoir conserver et partager un environnement logiciel à l’identique est de “figer” l’ensemble du système.

16.2.1 Isoler une machine

La façon la plus simple d’éviter les problèmes de versions des dépendances logicielles, consiste à installer sur une machine tous les programmes dont on a besoin. Ce *scenario* exclut toute mise à jour : l’environnement est installé une bonne fois pour toutes.

C’est le cas de figure décrit en section 2.5. Chacun des membres de

l'équipe peut alors se connecter sur cette machine pour y faire ses calculs, ce qui peut d'ailleurs parfois être source de tensions : or, personne ne souhaite découvrir à cette occasion les talents en *free fight* de ses collègues. Sur le long terme, isoler une machine s'avèrera problématique car si la machine continue (normalement) de calculer la même chose, elle conserve ses vieux *bugs* qui peuvent affecter vos résultats. D'autre part, elle finira un jour par mourir de sa belle mort et s'exposera entre-temps à des failles de sécurité. C'est donc certainement la solution la plus simple à mettre en œuvre (au début), mais c'est une solution à éviter autant que possible.

16.2.2 Isoler un environnement à l'aide d'un conteneur

Un conteneur est un outil permettant d'émuler un système d'exploitation particulier avec un certain nombre d'applications installées. Une solution très populaire pour utiliser les conteneurs est Docker [1]. Par rapport à une machine virtuelle, le conteneur a pour avantage de ne pas reproduire l'ensemble du système d'exploitation, en se passant du kernel. Le conteneur est donc plus léger, moins gros et plus rapide. L'inconvénient est que le conteneur ne fonctionne pas pour toutes les combinaisons de machine réelle - machine virtuelle.

Il existe d'ailleurs des outils permettant d'identifier et de capturer automatiquement un tel environnement logiciel minimum pour pouvoir le partager avec d'autres : CDE [2] ou ReproZip [3]. En savoir plus sur Reprozip : (**?**). Ces approches automatiques sont efficaces et très pratiques mais peuvent passer à côté de certaines dépendances et ne permettent pas de variation. En effet, on obtient un environnement "binaire" figé sans sa recette de construction et il est donc très difficile de de changer une bibliothèque particulière ou de faire évoluer cet environnement.

1. https://www.docker.com/
2. http://www.pgbovine.net/cde.html
3. https://vida-nyu.github.io/reprozip/

16.3 Les systèmes de gestion de paquets

L'approche la plus aboutie pour la reproduction de l'environnement logiciel s'appuie sur les systèmes de gestion de paquets fonctionnels tels que GUIX [4] (voir aussi (**?**) et (**?**)) et NIX [5]. Il s'agit d'outils permettant de décrire et d'isoler très précisément l'environnement strictement nécessaire à la recherche reproductible, de le partager avec d'autres pour qu'ils puissent le mettre en œuvre, et même d'y apporter des modifications très précisément contrôlées afin d'évaluer l'impact de tel ou tel changement.

16.4 Pour en savoir plus

Au sujet de l'identification des dépendances, vous pouvez vous référer au module 4 du MOOC sur la recherche reproductible [6] (**?**) et aux ressources correspondantes (slides [7] (**?**), ressources [8]) (**?**).

Vous pouvez également vouloir regarder ce séminaire en ligne présentant quelques solutions pour contrôler son environnement [9] (**?**).

4. https://www.gnu.org/software/guix/
5. https://nixos.org/
6. https://learninglab.inria.fr/mooc-recherche-reproductible-principes-methodologiques-pour-une-science-trar
7. https://gitlab.inria.fr/learninglab/mooc-rr/mooc-rr-ressources/blob/master/module4/slides/C028AL_slides_module4-fr-gz.pdf
8. https://gitlab.inria.fr/learninglab/mooc-rr/mooc-rr-ressources/blob/master/module4/ressources/resources_environment_fr.org
9. https://github.com/alegrand/RR_webinars/blob/master/2_controling_your_environment/index.org

17

Sortez couverts !

" Au fait, ai-je le droit ? "
– Anonyme

" *Oops, I dit it again* "
– Britney S.

Il est primordial de se poser la question du droit avant de partager des documents, *a fortiori* sur internet.

17.1 Licence

Qu'il s'agisse de données ou de code, l'utilisation d'une licence facilite la spécification des conditions dans lesquelles se fait le partage. On précise notamment comment le document peut être cité, utilisé, modifié tout en posant des conditions et des limites d'usage. Apposer une licence à un document que vous avez conçu épargne à un tiers de fastidieuses démarches de demandes d'autorisation.

Traiter du choix de ces licences dépasse très largement le périmètre de cet ouvrage et nous renvoyons le lecteur vers son employeur ou ses organismes de tutelle.

À noter cependant que les licences utilisées pour le partage des données sont le plus souvent différentes de celles couvrant les programmes et les logiciels : le site "Choose a license" [1] (**?**) et le site "Crea-

1. https://choosealicense.com/

tive commons" [2] (**?**) permettent de choisir une licence logicielle ou non logicielle (respectivement) en répondant à 2 ou 3 questions.

Sur les licences d'utilisation, vous pourrez lire aussi ce billet de blog (**?**).

17.2 Confidentialité et protection des données, respect de la vie privée

Dans le chapitre 11, nous avons vu que le partage des données était une des composantes de la recherche reproductible. Or il existe certains domaines (tels que la santé) où le partage des données n'est pas toujours possible pour :

- des raisons éthiques,
- de confidentialité,
- de secret industriel,
- de protection des personnes…

Il est donc indispensable de se poser la question du droit, notamment dans le cadre du Règlement Général sur la Protection des Données (RGPD). Un certain nombre de textes officiels et de documents sont disponibles sur le site de la Commission nationale de l'informatique et des libertés (CNIL) [3] (**?**).

Des solutions alternatives au partage totalement libre des données peuvent être mises en œuvre : on peut citer la mise à disposition sur demande *via* une charte de confidentialité.

2. https://creativecommons.org/choose/
3. https://www.cnil.fr/fr/textes-officiels-europeens-protection-donnees

17.3 Pour en savoir plus

Le droit évolue constamment et les ressources sur le sujet sont pléthoriques. De fait, nous vous indiquons quelques sites web qui opèrent une sélection de contenus :

— le site web du Comité pour la Science Ouverte (**?**)
— l'Inra a dédié un site aux questions de partage de données scientifiques (**?**)
— les fiches pratiques du Cirad vous permettent d'aborder facilement les questions juridiques (**?**)
— l'essentiel des ressources est en anglais, mais vous trouverez aussi quelques contenus en français sur le site de *Foster Open Science* (**?**)

Nous vous invitons aussi à consulter les vidéos de l'intervention de L. Maurel sur le RGPD (**?**).

Enfin, le site WillO, développé par l'Université de Lille, vous permet de mieux comprendre l'impact de la loi pour une République Numérique sur vos travaux de recherche (**?**).

Bibliographie

Le texte intégral de certains articles cités n'est pas accessible en accès ouvert.

Dans ce cas, nous vous invitons à vous rapprocher des bibliothécaires de votre structure (laboratoire, université) pour en savoir plus sur les abonnements en cours ou les services de prêt entre bibliothèque.

Nous vous signalons quelques outils pour rechercher de manière légale la version en libre accès d'un article :

— oaDOI [4] : https://oadoi.org/
— Unpaywall [5] : http://unpaywall.org/
— Scili [6] : https://www.scilit.net/

4. https://oadoi.org/
5. http://unpaywall.org/
6. https://www.scilit.net/

Quatrième partie

Annexes

18

Annexes

18.1 Modalités de conception de l'ouvrage

Ce document a été conçu dans le cadre d'un *book sprint* organisé par l'Unité régionale de formation à l'information scientifique et technique (Urfist) de Bordeaux du 15 au 19 avril 2019. *Book sprint* = rédaction collaborative inspirée des *sprints* informatiques.

18.2 Objectif

Réunir des profils variés pour mettre à disposition du plus grand nombre, sans barrière de coût, un ouvrage rédigé le temps du *sprint*.

Dans la forme traditionnelle du *book sprint* (**?**), les auteurs conçoivent la **table des matières à partir de rien** : il n'y a pas de séance de travail préparatoire. **C'est le travail collaboratif qui permet de faire émerger les contenus et leurs publics cibles**. Le *book sprint* n'est ni un séminaire, ni une retraite, mais en présente certaines caractéristiques car généralement, les auteurs sont logés dans un même endroit. Dans les années 2000, les *book sprints* se sont développés sur les modèles des sprints informatiques pour produire rapidement de la documentation sur les logiciels libres. Adam Hyde, fondateur de *FlossManuals* (**?**) a été pionnier dans le domaine. Si la méthode s'est popularisée dans les pays anglo-saxons, et particulièrement dans le secteur de l'entreprise et le monde associatif, cette technique de rédaction reste moins utilisée par les publics académiques, même si elle tend à être progressivement adoptée.

Elle appelle en effet des adaptations pour être applicable, mais offre des potentialités variées aux auteurs, une fois la méthode testée.

18.3 Que peut apporter un book sprint à des chercheurs.euses ?

La méthode du *book sprint* est actuellement encore peu diffusée en France, dans le milieu de la recherche mais :

- **les modalités de diffusion du livrable sont adaptées au milieu académique** : archive ouverte, plates-formes scientifiques
- **la méthode est transposable à d'autres activités de rédaction** : documentation technique, appels à projets, supports pédagogiques, *white papers*, *etc*.

Ce n'est pas un cadre pour faire surgir des idées totalement nouvelles : la très forte contrainte temporelle fait qu'il ne faut pas surestimer la capacité à faire surgir des idées complètement innovantes sur un sujet. En revanche, **le *book sprint* permet de confronter et de tester ses idées avec des auteurs issus d'autres disciplines, ayant d'autres expertises**, voire de modifier son point de vue sur certains sujet au fil des échanges.

Le *book sprint* peut aussi être l'occasion de **formaliser des idées dans un cadre plus souple que les formats de publication classiques**. Un **travail préparatoire** de la part des organisateurs permet aux auteurs de se projeter **sans pour autant rédiger à l'avance** leur partie : la table des matières est conçue grâce aux échanges.

La qualité du livrable ne peut pas être d'un niveau égal à celui d'un article ou d'un chapitre d'ouvrage ; il n'est pas réaliste d'attendre le même degré d'aboutissement qu'une production académique compte tenu du cadre imparti, mais :

- cette méthode permet de réaliser des travaux qui auraient peu de chances de voir le jour dans des conditions de travail plus traditionnelles. La contrainte temporelle agit comme un levier puissant pour se concentrer sur l'essentiel. Il s'agit de consi-

dérer le livrable comme le *draft* le plus abouti possible d'un contenu amené à évoluer.
— la plate-forme de diffusion permettra le versionnage.
— les attentes concernant un guide pratique ne sont pas les mêmes que celles qui portent sur un article.

Le *book sprint* n'est pas conçu pour améliorer les techniques rédactionnelles des auteurs. En revanche, la présence d'un éditeur permet d'homogénéiser les disparités rédactionnelles, de délester les auteurs des vérifications syntaxiques et orthographiques.

La notion de premier auteur n'existe pas, mais :

— le droit de paternité, au sens du code de la propriété intellectuelle, est respecté. Le livrable est déposé dans une plate-forme permettant de rendre le livrable citable la validation des contenus est collégiale car la philosophie du *book sprint* met l'accent sur la collaboration.
— le facilitateur conduit les échanges et permet aux auteurs de conserver une dynamique adaptée au rythme du *sprint*. Si la nature fortement collaborative du travail fait émerger une part d'imprévu, il s'agit toutefois d'un exercice cadré, fondé sur une méthodologie.

Il est difficile de travailler avec des co-auteurs qu'on ne connaît pas, *a fortiori* s'ils ne sont pas spécialistes du même domaine que soi ; pour le bon déroulement du *book sprint*, certains rôles sont définis à l'avance : le facilitateur aide les auteurs à coordonner leurs idées. La rencontre avec des co-auteurs inhabituels enrichit les échanges.

Portraits

Auteurs

Ce livre a été rédigé de manière collaborative par plusieurs auteurs. Qui sont-ils ?

Loïc Desquilbet

Enseignant-chercheur en biostatistique et en épidémiologie clinique à l'Ecole Nationale Vétérinaire d'Alfort [1], Loïc Desquilbet travaille dans le domaine de la recherche clinique vétérinaire, incluant la validation de méthodes de mesure (répétabilité, reproductibilité, et concordance de méthodes). Il travaille sur SAS et a conçu une macro SAS pour prendre en compte les variables quantitatives dans un modèle multivarié. Ses enseignements visent à former les étudiants à l'analyse critique d'articles.

— site web [2]

Boris Hejblum

Enseignant-chercheur en biostatistique à l'Institut de Santé Publique d'Épidémiologie et de Développement [3] (ISPED, Université de Bordeaux), Boris Hejblum est membre de l'équipe SISTM dans le centre

1. https://www.vet-alfort.fr/
2. https://loicdesquilbet.wixsite.com/biostat-epidemio
3. http://www.isped.u-bordeaux.fr/

Inserm U1219 Bordeaux Population Health et Inria Bordeaux Sud-Ouest [4].

Il travaille sur l'analyse de données biomédicales répétées de grande dimension, en particulier la statistique génomique dans le cadre de la recherche vaccinale ainsi que sur le traitement de dossiers médicaux informatisés par des approches de machine learning. Il développe de nouvelles méthodologies statistiques implémentées sous forme de paquets R open-source permettant leur réutilisation, et la reproductibilité des résultats de ses publications.

— packages [5]
— site web [6]
— suivre B. Hejblum sur Twitter [7]

Arnaud Legrand

Chercheur au Laboratoire d'informatique de Grenoble [8], Arnaud Legrand pilote l'équipe Polaris (Inria) [9]. Il s'est spécialisé dans l'évaluation (simulation, observation, statistiques, analyse de traces) et l'optimisation de performance de grandes infrastructures de calcul (supercalculateurs, grille, calcul bénévole, clouds, ...). L'axe optimisation de ses travaux porte sur la proposition ou l'évaluation de méthodes souvent sophistiquées et où il est important de conserver tous les détails pour comprendre leur performance et comment les améliorer. Arnaud Legrand co-pilote le Mooc "Recherche reproductible" [10].

— site web [11]
— suivre A. Legrand sur Twitter [12]

4. https://www.inria.fr/equipes/sistm
5. https://borishejblum.science/software/
6. https://borishejblum.science/
7. https://twitter.com/borishej
8. http://www.liglab.fr/
9. https://team.inria.fr/polaris/
10. https://www.fun-mooc.fr/courses/course-v1:inria+41016+session01bis/about
11. http://mescal.imag.fr/membres/arnaud.legrand/
12. http://twitter.com/arnaudlegrand17

Pascal Pernot

Physico-chimiste et directeur de recherches au CNRS, Pascal Pernot travaille au Laboratoire de Chimie Physique à l'Université Paris-Sud. Ses thématiques principales sont la gestion des incertitudes en modélisation physico-chimique, et notamment la quantification des incertitudes de prédiction des méthodes de chimie quantique et moléculaire. Il assure des formations en lien avec les méthodes standardisées de propagation des incertitudes. Il utilise l'environnement Rstudio/Github/Zenodo pour accompagner ses publications des données et codes nécessaires à la reproduction des résultats.

— page web [13]
— GitHub « Reproducible Research » [14]

Nicolas Rougier

Chercheur à l'Inria, Nicolas P. Rougier est membre de l'équipe Mnemosyne project [15] dont l'approche se situe à la frontière entre les neurosciences intégratives et computationnelles en association avec l'Institut des maladies neurodégénératives [16], le Laboratoire Bordelais de Recherche en Informatique [17], l'Université de Bordeaux et le CNRS. En 2015, il a co-fondé avec Konrad Hinsen [18], la revue *ReScience* [19] spécialisée dans la publication d'études de réplication computationnelles. Il anime régulièrement des formations pour des doctorants, des chercheurs, des ingénieurs.

— site web [20]
— suivre N. Rougier sur Twitter [21]

13. http://pagesperso.lcp.u-psud.fr/pernot/
14. https://github.com/ppernot/Reproducible-Research
15. https://mnemosyne-proj.org/
16. https://www.imn-bordeaux.org/
17. https://www.labri.fr/
18. https://github.com/khinsen
19. http://rescience.github.io/
20. http://www.labri.fr/perso/nrougier/
21. https://twitter.com/NPRougier?lang=fr

Facilitatrice

Elisa de Castro Guerra, ActivDesign

Formatrice à Activdesign [22] et présidente de Floss Manuals Francophone [23], Elisa de Castro Guerra est spécialisée en création numérique et web. Elle travaille sur le développement d'applications web depuis plusieurs années. E. de Castro Guerra milite pour l'utilisation des logiciels libres, en particulier par la facilitation d'écriture de documentations en langue française. E. de Castro Guerra anime ainsi régulièrement des book sprints pour des associations, des entreprises.

Coordinatrice du projet

Sabrina Granger

Docteur en langue et littérature françaises, Sabrina Granger est conservatrice des bibliothèques à l'Unité Régionale de Formation à l'Information Scientifique et Technique de Bordeaux. Ses différents postes l'ont amenée à travailler dans les domaines de l'informatique documentaire, la communication en BU et les services aux chercheurs. S. Granger est co-pilote du collège « Compétences et formations » du Comité pour la Science Ouverte [24].

22. https://activdesign.eu/
23. https://www.flossmanualsfr.net/
24. https://www.ouvrirlascience.fr/presentation-du-comite/

Contributeurs de la 1re édition

Ils ont participé à différentes étapes du *book sprint* :

— **Martine Courbin-Coulaud**, documentaliste, Information et Edition Scientifique, Centre de recherche Inria Bordeaux Sud-Ouest
— **Ludovic Duvaux**, bio-informaticien, Unité Mixte de Recherche "Biogeco", Inra
— **Pierre Gravier**, conservateur des bibliothèques, Direction de la documentation, Université de Bordeaux
— **Grégoire Le Campion**, ingénieur en traitement et analyse de données, Unité Mixte de Recherche "Passages", CNRS
— **Solenne Roux**, ingénieure en traitement et analyse de données, Equipe d'Accueil "Laboratoire de Psychologie", Université de Bordeaux
— **Frédéric Santos**, statisticien, Unité Mixte de Recherche "Pacea", CNRS

Contribuer

Vous aussi lecteurs, vous pouvez contribuer en proposant :

— une *pull-request* sur le répertoire contenant les sources de ce livre à l'adresse : https://github.com/alegrand/bookrr/pulls
— en envoyant un message à urfist@u-bordeaux.fr [25]

25. mailto:urfist@u-bordeaux.fr

Remerciements

Sincères remerciements à **Isabelle Scarpat-Bouvet** pour son implication dans l'organisation du projet.

Quatrième de couverture

Pour un chercheur, il n'y a rien de plus frustrant que l'impossibilité de reproduire des résultats majeurs obtenus quelques mois auparavant. Les causes de ce type de déconvenues sont multiples et parfois pernicieuses. Ce phénomène participe à ce que certains identifient comme une "crise de la reproductibilité de la recherche".

Cet ouvrage considère un ensemble de situations et de pratiques potentiellement dangereuses afin d'illustrer et de mettre en évidence les symptômes de la non-reproductibilité dans la recherche. À chaque fois, il propose un éventail de solutions allant de bonnes pratiques faciles et rapides à implémenter jusqu'à des outils plus techniques, tous gratuits et mis à l'épreuve par les auteurs eux-mêmes.

Avec ce livre, étudiants, ingénieurs et chercheurs devraient trouver des moyens efficaces et à leur portée pour améliorer leurs pratiques de la recherche reproductible.

---

*For a researcher, there is nothing more frustrating than the failure to reproduce major results obtained a few months back. The causes of such disappointments can be multiple and insidious. This phenomenon plays an important role in the so-called "research reproducibility crisis".

This book takes a current perspective onto a number of potentially dangerous situations and practices, to examplify and highlight the symptoms of non-reproducibility in research. Each time, it provides efficient solutions ranging from good-practices that are easily and immediately implementable to more technical tools, all of which are free and have been put to the test by the authors themselves.

With this book, students and engineers and researchers should find

efficient and accessible ways leading them to improve their reproducible research practices.*

Bibliographie