0. Introduction

Bonjour à tous, si, comme moi, vous êtes assez mal organisé, cette petite BD de PHD Comics doit vous parler. Notre module s'intitule "La vitrine et l'envers du décor", mais nous aurions pu le baptiser "La théorie et la pratique".

L'apparence que nous donnons en tant qu'expert de notre domaine, surtout lorsque nous communiquons avec des collègues, c'est d'avoir une organisation impeccable, garantie de la qualité de nos travaux. Mais en pratique nous sommes souvent bien moins organisés que ce que nous aimerions être.

Si vous êtes comme moi, vous avez eu envie que ça change. Dans ce module, nous verrons ensemble ce qu'est un document computationnel et comment en réaliser un. C'est un terme barbare mais il nous a semblé décrire le mieux ce document permettant d'améliorer la traçabilité d'un calcul.

Pour vous convaincre de l'importance de cette traçabilité, je vais vous présenter quelques exemples récents de travaux dont les résultats ont été très controversés et discutés. Ils illustrent à quel point la capacité à pouvoir inspecter les méthodes utilisées pour produire tels résultats est essentielle. Comme nous le verrons, la plupart des problèmes rencontrés sont liés aux calculs : erreurs de programmation, transformations de données peu rigoureuses ou

procédures statistiques discutables. Une fois les origines des problèmes identifiées, je vous présenterai les bonnes pratiques afin de les éviter. Je vous expliquerai ce qu'est ce fameux document computationnel. Un document qui permet de présenter à d'autres ses travaux, ce que j'appelle "la vitrine", mais aussi d'accéder à l'ensemble des calculs sous-jacents, ce que j'appelle "l'envers du décor". Il existe plusieurs formats de tels documents.

Nous essaierons trois environnements permettant d'en produire facilement. Ils se distinguent par le niveau de technicité nécessaire à leur mise en œuvre. À vous de choisir ce qui vous convient, quitte à en essayer un autre plus tard.

Jupyter, le premier, a été intégré au MOOC et au GitLab et ne nécessite aucune installation de votre part. C'est le plus simple à mettre en œuvre. Les données et calculs seront hébergés sur nos serveurs, vous pourrez y accéder et les récupérer sur votre ordinateur à tout moment via le GitLab. Jupyter vous permettra de gérer des notebooks et d'utiliser le langage Python 3 ou bien le langage R.

RStudio, le second, est un environnement de développement spécifique au langage R. Il vous faudra l'installer et synchroniser vos productions avec le GitLab. C'est un logiciel très bien maintenu fonctionnant sous Lunix, Mac et Windows et son installation devrait être simple. Si vous êtes déjà familier avec le langage R, je vous le recommande. Ainsi, vous aurez, à la fin du MOOC, un environnement de travail prêt à l'emploi. Avec RStudio, il est également possible d'utiliser du code Python. Mais si vous voulez en faire principalement, il vaut mieux utiliser Jupyter.

Enfin, Org-mode. C'est le plus délicat à mettre en œuvre, mais c'est aussi celui que Konrad, Christophe et moi-même utilisons quotidiennement aussi bien pour gérer notre cahier de laboratoire que pour rédiger des articles ou des notes techniques. Nous nous sommes donc dit qu'il serait intéressant de l'installer et de vous le présenter. Il nécessite l'installation d'Emacs, un éditeur de texte qui révèle sa puissance dans un environnement Linux ou Mac. Il permet de combiner très efficacement différents langages et, par certains aspects, est moins limité que les précédents. Cette option est à réserver à un public averti, mais le jeu en vaut la chandelle.

Une fois familiarisé avec un de ces environnements, nous vous proposerons une séance pratique dont l'objectif sera de produire un petit document computationnel. Les dernières séquences peuvent être visionnées indépendamment et traitent de sujets un peu plus techniques, en particulier de comment utiliser de tels outils avec vos coauteurs, qui peuvent avoir leurs propres habitudes, leurs contraintes, et également, comment produire des

documents avec un style spécifique. Nous conclurons ce module avec une comparaison des différents outils et une présentation des bénéfices d'une utilisation quotidienne. À bientôt.

1. Exemples récents d'études assez discutées

Je vous avais promis quelques exemples récents de travaux dont les résultats ont été controversés et discutés. C'est ce que je vais vous présenter.

Commençons par des travaux en économie. En 2007, crise des subprimes aux U.S.A. De 2008 à 2009, deux économistes prestigieux, Carmen Reinhart et Kenneth Rogoff, présentent leurs travaux et annoncent que la crise financière est loin d'avoir produit toutes ses conséquences. En 2010, ils publient un article intitulé "Growth in a Time of Debt" qui étudie le lien entre dette et croissance. Leurs principales conclusions sont que lorsque la dette extérieure d'un pays dépasse 90 % du PIB, les conséquences sur la croissance sont dramatiques. De nombreux politiciens, journalistes et activistes s'appuient sur cet article pour soutenir la mise en place de politiques d'austérité budgétaire. Ces seuils de 90 %, ainsi que l'ampleur des conséquences sont très discutés. D'autant plus que certains chercheurs échouent à obtenir des résultats similaires en utilisant les données publiquement disponibles. Ils demandent donc à Reinhart et Rogoff l'ensemble des données et des feuilles de calcul utilisées dans l'étude. Ces derniers les leurs fournissent.

Des erreurs de calculs évidentes apparaissent rapidement ainsi que des traitements de données assez douteux. Exclusions de données, pondérations suspectes, etc. Reinhart et Rogoff répondent point par point en expliquant que ces quelques erreurs ne changent rien au résultat final, que leurs façons de calculer les statistiques sont standard. En fait, une fois les détails révélés, pour beaucoup de chercheurs, les valeurs utilisées sont discutables et les calculs ont très peu de sens. Il est malhonnête d'utiliser ces travaux pour justifier une politique d'austérité budgétaire. Mais le mal est fait. Pendant plus de 3 ans, l'austérité n'est pas présentée comme un choix mais comme une nécessité. Quand bien même l'article original est considéré comme non pertinent par les économistes, ses idées ont fait leur chemin et sont difficiles à détrôner. Au-delà du caractère idéologique de ce genre de travaux, une des raisons pour lesquelles ce débat a mis autant de temps à avoir lieu est liée à la non publication de l'ensemble des procédures de calcul et des données utilisées. Pratique assez courante en économie. Les auteurs ont été forcés de mettre à disposition ce qui sous-tendait leurs travaux. Mais sans pression médiatique particulière, en général rien ne se passe.

Continuons avec un autre exemple. L'imagerie cérébrale qui permet d'observer l'activité du cerveau lors d'une tâche cognitive. Ainsi, cela permet de mieux comprendre la structure et le fonctionnement du cerveau. L'IRM fonctionnelle est l'une de ces techniques. Elle mesure de très faibles variations locales du taux d'oxygénation du sang dans le cerveau. En 2010, Craig Bennett et ses encadrants de thèse ont une idée saugrenue. Ils placent un saumon mort dans un appareil d'IRM et lui présentent des images. Étonnamment, ils observent des signes d'activité cérébrale. Ce qui est surprenant car le saumon est bel et bien mort. Aussi drôle que cela puisse paraître, Bennet et ses encadrants savent ce qu'ils font. Les données brutes obtenues lors d'une IRM sont extrêmement bruitées. Toute une série de calculs et de tests statistiques sont appliqués pour transformer les données en images. Mais il arrive que le bruit soit trop important, que la machine soit mal calibrée, que la procédure de calcul soit inadaptée et que des signaux apparaissent fortuitement. Leur article est rédigé avec un ton très humoristique et il fait relativement sensation car il met le doigt sur des faiblesses méthodologiques.

En 2016, des collègues me sachant intéressé par ces problèmes de réplication me font suivre un article récent assez alarmant. Il présente un problème dans les procédures statistiques utilisées dans les logiciels les plus couramment utilisés. Ce qui remet potentiellement en cause les résultats obtenus depuis 15 ans. Étant donné l'ampleur de l'erreur, les auteurs concluent que 40 000 articles, au bas mot, pourraient être concernés. De plus, les données étant très volumineuses dans ce domaine, elles ne sont généralement pas archivées et il ne sera donc pas possible de les réanalyser. L'ensemble des expériences serait à refaire. En fait, suite aux retours qui leur sont faits, les auteurs revoient rapidement à la baisse leurs estimations alarmistes. Au final, le problème est méthodologique et lié à la capacité à vérifier les études suite à des erreurs de calcul. Ce problème reste entier, même s'il ne remet pas pour autant en cause l'ensemble des résultats obtenus ces dernières années.

Un dernier exemple, cette fois-ci en cristallographie. Geoffrey Chang est un chercheur à la trajectoire fulgurante récompensé par de nombreux prix. Son équipe, basée au Scripps Institute à l'université de Californie San Diego, a publié des articles dans des revues prestigieuses détaillant la structure de certaines protéines présentes dans les membranes de cellules. Ces protéines jouent un rôle essentiel dans la résistance de ces bactéries à certains médicaments. Connaître leur structure est une étape importante dans la compréhension de leur fonctionnement. Hélas, peu de temps après, d'autres équipes de chercheurs étudiant des protéines très similaires rapportent des structures anormalement différentes de celles publiées par Cheng et son équipe. En lisant ces travaux, Chang, horrifié, remonte vite à la source du

problème. Un des codes d'analyse aurait inversé deux colonnes de données et ainsi inversé la répartition de la densité d'électrons à partir de laquelle la structure finale de la protéine est calculée. D'après Chang, ce code aurait été hérité d'un autre laboratoire et s'était répandu depuis dans d'autres équipes. Même si toute l'acquisition des données avait été faite soigneusement, ce n'était pas le cas de l'analyse. C'est ce petit grain de sable qui a conduit à la rétractation immédiate de cinq articles par Chang et son équipe. Ces publications ont eu un impact énorme sur la communauté, à tel point que plusieurs années après la rétractation, les résultats contradictoires avec ceux de Chang paraissaient suspects et avaient du mal à être publiés.

Il n'y a à ce jour pas un domaine des sciences qui ne soit épargné par ces difficultés à reproduire les travaux publiés. En oncologie, un article récemment publié rapporte que plus de la moitié des études publiées ne peuvent être reproduites en laboratoire industriel et ce, même si elles sont publiées dans des journaux prestigieux. En psychologie, la capacité à reproduire les résultats publiés est également très très basse.

Le problème est méthodologique, mais également certainement sociologique. Il y a une pression productiviste trop importante. Mais attention à ne pas donner trop d'importance aux signaux d'alerte que nous venons de voir. Le problème est compliqué mais il faut garder à l'esprit que la remise en cause fait partie du processus scientifique. Il n'est pas surprenant que de telles difficultés de reproduction de travaux soient présentes. Cependant, 2 autres caractéristiques essentielles du processus scientifique sont la rigueur et la transparence. Et il est clair que dans l'ensemble des cas que nous venons de voir, il manquait souvent l'un et l'autre.

2. Pourquoi est-ce difficile ?

Comme nous l'avons vu, reproduire des travaux de recherche, même publiés dans des revues prestigieuses, est assez difficile. Dans cette séquence, nous identifierons les difficultés les plus communes.

La 1re source de difficultés rencontrée lors de tentatives de reproduction est simplement le manque d'informations. Si on ne prend pas soin d'expliciter comment on a obtenu nos données ou si on les garde secrètes, il sera difficile pour une tierce personne de vérifier si elle arrive aux mêmes conclusions, ou pas. De même, expliciter les choix effectués à chaque étape de l'étude s'avère essentiel. Quel protocole expérimental a été utilisé et pourquoi ? Quelles

données ont été conservées ? Quelles données ont été soigneusement écartées et pourquoi ? Quelle procédure statistique a été utilisée et les hypothèses sous-jacentes étaient-elles raisonnables, etc. ? Après tout, on fait toujours des choix, à un moment donné. Mais ne pas expliciter ses choix, ne pas les expliquer, même en étant rigoureux et de bonne foi, c'est prendre le risque que les lecteurs deviennent suspicieux. D'autre part, être totalement transparent sur ses choix, c'est permettre à d'autres ou à vous-même de décider s'il est nécessaire de revenir dessus ou pas. Pour garder trace de tous ces choix et des informations sur ces données, le cahier de laboratoire est un outil essentiel, encore faut-il ensuite mettre à disposition ces informations. Nous reviendrons sur ce point par la suite.

La 2e source de difficultés rencontrée lors de tentatives de réplication est les erreurs induites par l'utilisation effrénée des ordinateurs. Comme toute innovation technologique, les ordinateurs nous permettent d'aller plus vite, plus loin, mais aussi de faire des erreurs plus facilement et plus rapidement. Au premier plan des sources d'erreurs, ces logiciels intuitifs où l'interaction se fait à la souris et qui ne permettent pas d'expliciter ce qui est calculé, ni à partir de quoi. Leur facilité d'utilisation incite à les utiliser à peu près pour tout, même ce pour quoi ça n'était pas prévu au départ. Il est, par exemple, très difficile de comprendre la logique du calcul d'une feuille Excel, pleine de macros. Bien sûr, il est possible de gérer des stocks et la comptabilité d'une entreprise avec un tableur. Mais ceux qui ont déjà eu affaire à ce genre d'abomination savent qu'un ERP permet d'éviter bien des soucis.

De la même façon, un tableur n'est pas le bon outil pour les statistiques ou le traitement de données. Tenez, quelques exemples d'anecdotes effroyables liées à l'utilisation d'un tableur. En génomique, il est bien pratique de donner des petits noms aux gènes, tel que celui que vous voyez à l'écran, Membrane-Associated Ring Finger, bla-bla-bla. En général, ce gène-là, on l'appelle MARCH1. Seulement, pas mal de tableurs croient bien faire en interprétant ceci comme une date et, par effet de bord, comme le nombre de secondes écoulées depuis le 1er janvier 1970. Ou alors l'identifiant de ce gène, 2310009E13, se retrouve également interprété comme un chiffre. Bien sûr, il n'y a peut-être qu'une vingtaine de gènes, en tout et pour tout, concernés par ce genre de problème, sur les 30 000 que comporte le génome humain. Mais c'est quand même assez désagréable.

De manière générale, il est courant de se reposer sur une pile logicielle complexe. Complexe et souvent mal maîtrisée. Combien d'études reposent ainsi sur des logiciels propriétaires, sortes de boîtes noires dont on ne maîtrise pas le contenu, et qui appliquent aveuglément des procédures de calculs et des transformations de données ?

Je ne suis pas en train de dire qu'il faut tout reprogrammer soi-même. Ce n'est, bien sûr, pas la meilleure façon d'échapper aux bugs. Dans les cas de lecture de Reinhart et Rogoff ou des fausses structures de protéines de Cheng, les erreurs venaient de programmes maison. Mais il me semble essentiel d'être en mesure de déterminer, dans son analyse, si chaque brique est digne de confiance ou pas. Si l'un des composants, par sa nature, l'interdit, il faut sérieusement songer à le remplacer.

Enfin, la dernière cause méthodologique de non-reproductibilité et source d'erreurs est certainement le manque de rigueur et d'organisation. Même si le stockage ne coûte plus rien de nos jours, la sauvegarde des données est souvent mal assurée et il est courant d'en perdre à la suite d'une mauvaise manipulation ou bien de la suppression de son compte informatique, lors du passage d'une institution à une autre. En l'absence de mécanisme de gestion de versions, il est aussi courant de remplacer par inadvertance d'anciennes données par de nouvelles et de ne plus arriver à accéder aux anciennes observations. Dans un certain nombre de domaines, l'utilisation de plans d'expériences randomisés ou de pré-études n'est absolument pas systématique. De même, les bonnes pratiques de développement logiciel, comme la revue de code ou l'intégration continue, sont rarement appliquées aux logiciels utilisés dans la recherche.

Enfin, il serait naïf de ne pas évoquer la dimension culturelle et sociale du problème. Un article est une version simplifiée et intelligible des résultats. Certains diraient même de la publicité. Une description de haut niveau est essentielle, car elle permet de prendre du recul, mais elle est devenue la norme, alors que la technicité de la recherche actuelle est telle qu'il est clairement impossible de donner, dans un document de huit à dix pages, toutes les informations permettant de refaire les expériences et les analyses. Je parle de la description du protocole expérimental, qui est souvent assez succincte. Les données sont généralement bien trop nombreuses pour être données in extenso. Elles sont résumées à travers quelques courbes. Les traitements statistiques appliqués pour parvenir à ces courbes ne sont décrits que rapidement. Bien sûr, il est possible de tracer toutes ces informations et de les mettre à disposition. À ceci près que cela demande du temps, voire beaucoup de temps si on ne dispose pas des bons outils. Mais si personne n'exige ces informations, à quoi bon s'embêter ?

Et donner accès, mettre à disposition, cela veut dire tout rendre public ? N'est-ce pas un petit peu radical ou risqué ? Démontons ensemble quelques-unes des idées reçues à ce sujet. "Si je donne accès à tout ce que j'ai fait, il deviendra alors évident que ce que j'ai fait n'est pas aussi parfait que ce que je prétends, que c'est un petit peu sale, pas toujours très rigoureux,

que j'ai sélectionné les résultats que j'ai présentés." C'est sûr. En même temps, c'est la réalité. Vous auriez tort de croire que vous êtes le seul dans cette situation. Si ça se trouve, vous travaillez mieux que les autres. Dans un domaine où la réputation est essentielle, vous avez probablement plutôt intérêt à le montrer. Pire, si vous le cachez, ça finira par paraître suspicieux.

Vous me direz aussi : "En révélant tout, je prends le risque que quelqu'un trouve une erreur. Et puis, j'ai pas envie de passer pour un imbécile." C'est certain, mais tout le monde fait des erreurs, même les plus grands, même les plus célèbres. En ce qui me concerne, je préfère que quelqu'un trouve une erreur dans mes travaux, et m'aide à la corriger afin que ça redevienne correct.

Autre argument souvent évoqué : "Quelqu'un pourrait tirer parti de mes travaux, réutiliser ces données que j'ai eu tellement de mal à obtenir, ou utiliser ce code que j'ai eu tant de mal à écrire, et faire trois ou quatre papiers là où je n'ai eu le temps d'en faire qu'un." OK. Alors, d'abord, si quelqu'un réutilise votre travail, il se doit de le citer correctement et de vous rendre le crédit qui vous est dû. Ensuite, les articles les plus cités, les plus cités de tous les temps, sont ceux qui présentent des contributions méthodologiques ou alors du logiciel qui est devenu essentiel dans un domaine. Ne négligez donc pas le fait de donner accès à ce que vous avez fait. Une petite parenthèse. Ceux d'entre vous qui connaissent GitHub savent que cette plateforme est à mi-chemin entre la plateforme de développement logiciel et le réseau social. Les contributions d'un développeur à différents projets sont visibles, ainsi que la fréquence de ses contributions. Ce qui permet à un jeune développeur de se faire une carte de visite infalsifiable et ultravisible. Montrer ce que l'on fait, c'est une façon efficace d'atteindre une certaine forme de reconnaissance par la communauté. Mettre ses travaux à disposition de tous, c'est sans doute la meilleure façon de démontrer la propriété intellectuelle.

Enfin, il est possible que les données soient sensibles et ne puissent être divulguées sans prendre le risque de causer du tort à des gens. Par exemple, s'il s'agit de données médicales, de données judiciaires, d'informations sur des enfants ou sur des intentions de vote. Dans ce type de situation, où les travaux ont une dimension éthique, il convient de définir quelles sont les personnes qui auront accès aux informations pour les vérifier ou les réutiliser. Ensuite, il existe des techniques cryptographiques faciles d'accès, qui permettent de s'assurer que seules les personnes habilitées peuvent accéder aux données.

Dans tous les cas, même si, au final, ces informations sont semi-publiques, il est essentiel de se donner les moyens de permettre à autrui d'inspecter ce que nous avons fait. Pour

permettre l'inspection, il faut utiliser les bons outils. Cela commence par bannir, autant que possible, les logiciels et les formats propriétaires. Bien sûr, ces outils sont bien faits. Vous y êtes habitués. Mais vous en êtes également captifs, car les entreprises qui les développent n'offrent aucune garantie à long terme. Je suis sûr que ces mises à jour automatiques vous agacent, vous aussi. L'expérience montre qu'il est très courant de perdre des données et des informations lors de ces mises à jour. Soyons honnêtes, l'utilisation de logiciels libres ne vous protège pas du tout contre ce genre de mauvaises surprises. Mais comme vous avez accès librement à l'ensemble des versions précédentes, les chances d'arriver à récupérer vos données sont bien plus élevées. De même, l'adoption de formats texte simples et ouverts augmente vos chances d'arriver à les relire avec d'autres logiciels. En ce qui me concerne, à chaque fois que j'ai utilisé un format binaire ou pas ouvert, j'ai fini par perdre des données. Ça fait donc dix ans que je n'utilise plus que des formats purement texte ou alors des standards binaires, mais ouverts uniquement. Le problème ne s'est plus jamais posé, et cela malgré les mises à jour extrêmement régulières de ma machine.

Donc, règle numéro 1 : utiliser autant que possible du format texte. Markdown, Org-mode pour vos notes, csv pour les données.

Règle numéro 2 : utiliser autant que possible les logiciels et les langages de programmation libres, comme R ou Python que nous utiliserons dans ce MOOC.

Et règle numéro 3 : éviter de stocker vos données chez un hébergeur dont vous pourriez être captif. Répliquez-les à plusieurs endroits. En effet, les services gratuits, ou pas, et très intégrés sont tentants, mais correspondent à un business plan très particulier, qui peut se révéler incompatible avec vos besoins futurs et ne pas être pérenne ni ne vous donne les garanties de confidentialité que vous souhaiteriez.

Enfin, attention aux tableurs et aux outils graphiques qui peuvent vous donner, au premier abord, une impression de meilleure efficacité ou de productivité, mais qui, sur le long terme, ne vous permettent pas d'atteindre la traçabilité et l'inspectabilité que vous voudriez. C'est plus difficile, en particulier au début. Mais en utilisant des langages comme R ou Python, passée la première marche, on gagne vite en puissance et en efficacité.

Nous avons vu que la première cause d'échec de reproduction de résultats scientifiques est le manque d'informations, la non-disponibilité des données ou des procédures appliquées. Cela ne signifie pas que le résultat soit faux, mais cela empêche de le vérifier et de s'appuyer dessus. La seconde cause d'échec est simplement l'erreur de calcul qui peut se glisser n'importe où. Et de façon générale, le manque de rigueur scientifique et technique est souvent

à l'origine des deux problèmes précédents. Il me semble que la meilleure attitude à avoir est celle de la transparence. Expliciter augmente les chances de trouver les erreurs et de les éliminer. Ainsi, nous assistons, ces dernières années, à un changement de paradigme de recherche et on exige un accès à l'ensemble des données et des procédures de calcul. C'est d'ailleurs une demande de plus en plus pressante de la société civile, en particulier du Conseil européen, afin d'éviter les dérives, mais surtout d'améliorer l'efficacité de nos travaux.

3. Le document computationnel : principe

Dans cette séquence, je vais vous expliquer enfin ce qu'est un document computationnel. Je vais vous présenter les principes sous-jacents que nous retrouverons dans chacun des trois environnements. L'objectif essentiel de ce type de document est de permettre la transparence la plus complète possible.

Si une partie de l'activité scientifique moderne nécessite des mesures sur le terrain ou derrière une paillasse, la majeure partie se passe derrière un ordinateur, les données provenant de nombreuses machines spécifiques, comme un accélérateur de particules, des séquenceurs ADN, un télescope, un réseau de capteurs ou des simulations numériques qui s'exécutent sur des supercalculateurs. Une fois ces données obtenues, elles sont transformées pour être analysées et visualisées afin de mieux en comprendre la structure. On partage souvent ces visualisations avec des collègues, on envoie des mails, on a un certain nombre d'itérations, on change de vue jusqu'à trouver celle qui explique le mieux les choses. On utilise souvent une multitude de logiciels spécifiques pour obtenir ces visualisations. On est souvent déçu par les résultats, on n'y comprend rien, ou on se rend compte que les données ne sont pas intéressantes et qu'il faut se poser la question sous un angle différent. Dans ce cas, retour à la case départ, acquisition de données, visualisation statistique, échanges avec les collègues, etc. Et puis, ça y est, on trouve quelque chose d'intéressant. On prend nos plus belles figures, on y ajoute les explications qui sont les bonnes, on tente de rendre tout cela intéressant et on l'envoie à une conférence ou un journal. Hélas, dans un PDF de 8 pages, le lecteur trouvera une jolie histoire, des figures convaincantes, mais n'aura aucune idée du travail effectué. L'article est la partie immergée de l'iceberg. On ne peut alors plus revenir sur les nombreux tentatives et échecs, ni sur les calculs et les données derrière toutes ces figures. La recherche reproductible permet de pouvoir naviguer dans les deux sens et de combler le fossé qui sépare l'auteur du lecteur.

Notre objectif est donc d'avoir un outil nous permettant d'inspecter toute cette démarche, afin que l'auteur puisse justifier pourquoi tel ou tel code est utilisé, et que le lecteur puisse comprendre ce qui a été fait, et d'autre part, de permettre de refaire le calcul et l'analyse le plus simplement possible. C'est essentiel pour plusieurs raisons : pour permettre au lecteur de vérifier l'exactitude des calculs, pour permettre éventuellement de corriger ces erreurs s'il y en a, et enfin, pour permettre à d'autres de réutiliser ces travaux dans un contexte différent, ou de réutiliser la procédure d'analyse dans un autre contexte.

Voici la partie émergée du document computationnel. Au premier abord, c'est un document banal, avec un titre, du texte, éventuellement un morceau de code illustrant une procédure de calcul, des résultats numériques, des formules de maths, des figures, etc., bref un article classique au format PDF ou HTML. Voici ce qui se cache derrière ce document. Ici, un notebook Jupyter, tel que nous allons le voir pendant le cours. Voici ce que vous verriez avec un navigateur Web, un document dynamique, constitué de différentes parties, sur lesquelles on peut interagir. On y trouve d'abord du texte au format Markdown, je l'ai surligné en orange. Le formatage est assez simple, on peut y inclure des liens hypertexte ou des formules mathématiques, mais vous le savez déjà. Entre ces zones de texte, on trouve des zones de code, en bleu. Il s'agit de fragments de code Python relativement simples. Dans cet environnement, on peut éditer directement ces fragments de code et les exécuter. En fait, un notebook a en interne une console Python ouverte. Pour exécuter un fragment de code, celui-ci est envoyé dans la console et le résultat automatiquement récupéré. Le résultat de chacun de ces différents codes est stocké juste en dessous, dans les zones que j'ai surlignées en jaune. Puisque cette console reste vivante tout au long du document, les résultats calculés dans un bloc sont visibles dans le bloc suivant, ce qui encourage à ne pas écrire de longs blocs de code, mais à écrire de petits fragments, à calculer les choses petit à petit, tout en expliquant dans les zones de texte, en orange, comment les différents fragments de code en bleu s'enchaînent et éventuellement pourquoi, en fonction de ce que l'on a calculé en jaune, et pourquoi on décide d'appliquer le nouveau fragment. Une fois satisfait des calculs, on aime en général créer un document classique en exportant vers du PDF ou du HTML. Chaque zone, texte, code, résultat, est convertie et assemblée en un document Markdown qui est transformé vers le format désiré avec un outil compound doc. Bien sûr, dans le document final, on peut ne pas souhaiter rendre tous les fragments de code et résultats de calcul visibles. C'est pourquoi, pour chaque zone, on peut spécifier si on souhaite la cacher ou pas lors de l'export. Si les résultats intermédiaires sont stockés dans le notebook, l'environnement permet de

réexécuter simplement l'ensemble du notebook et de mettre à jour les résultats. C'est à vous de décider, selon le contexte, si vous souhaitez partager le document final, souvent plus compact, ou le document computationnel, qui va permettre une inspection complète et une réutilisation.

Les 3 environnements les plus matures pour créer de tels documents sont Jupyter, que vous venez de voir, RStudio/knitr et Org-mode. Ils se distinguent par le niveau de technicité nécessaire à leur mise en œuvre. Je ferai une démo de chacun dans la séquence suivante. À son issue, vous pourrez choisir lequel vous convient le mieux.

Vous verrez que le principe reste le même dans les 3. Tout d'abord, on a un seul document comprenant un entrelacs d'explications format Markdown, de code exécutable et les résultats de ces exécutions. Cette organisation permet l'inspection et la réexécution qui sont nos objectifs. En interne, une console Python ou R est active et s'assure que les variables sont persistantes d'un fragment de code à l'autre. C'est la notion de session sur laquelle nous reviendrons. Enfin, il est possible d'exporter le document computationnel vers un format traditionnel, en masquant parfois certaines parties.

Les différences entre ces 3 environnements sont mineures : quelques différences de syntaxe, Jupyter et RStudio utilisent du Markdown, et Org-mode du format org, légèrement différent mais tout aussi lisible. La façon d'indiquer les zones de code et les résultats est un peu différente, mais c'est sans importance une fois l'environnement bien installé. Plus important, l'interopérabilité entre différents langages. Jupyter permet de faire du code Python, du R, du Ruby ou du Julia, mais pas vraiment d'utiliser plusieurs langages dans un même notebook. C'est possible, mais ça demande du travail. RStudio est conçu autour du langage R. Il est possible d'utiliser du Python, mais ce n'est pas aussi convivial. Enfin, Org-mode permet une interaction assez naturelle entre les langages, mais la courbe d'apprentissage est un peu plus raide. Enfin, ces outils diffèrent par le contrôle offert en termes de mise en page lors de l'export. Jupyter et RStudio se reposent sur Markdown et donc, sur Pandoc. Le style par défaut est très bien, surtout pour produire juste du HTML, mais pour produire un document PDF respectant un style particulier, ça peut être compliqué de demander à Pandoc d'appliquer ce style. C'est faisable, ceci dit. Org-mode ne fait à peu près rien pour vous, il vous faudra de toute façon configurer le style de votre export. Ceci dit, j'apprécie, lorsque je prépare un article en LaTeX avec Org-mode, de pouvoir écrire directement du LaTeX qui sera passé tel quel à l'export, ce qui me permet de faire ce que je souhaite. Vous savez tout, alors maintenant : démo.

4. Prise en main de l'outil

4A. Prise en main de l'outil (Jupyter)

Dans cette séquence, je vais vous présenter l'environnement Jupyter, qui permet de créer des documents computationnels avec le langage Python. Sur la gauche, j'ai ouvert un document computationnel avec Jupyter, tel que vous pouvez y accéder dans le cadre de Fun.

Je vais commencer par créer un nouveau document. Ici, j'ai accès au contenu de mon répertoire. Je vais pouvoir créer un nouveau notebook. Je vais commencer par lui donner un nom. Par exemple, "tutorial". Ce que je vous ai présenté dans les séquences précédentes, l'environnement Jupyter est composé de plusieurs cellules. Par défaut, mes cellules sont du code, mais je peux changer en Markdown et ainsi donner un titre à mon document. "Titre du document".

Je peux ensuite insérer une nouvelle cellule en dessous dans laquelle je peux écrire du code, du code extrêmement compliqué comme celui-ci. Je peux ensuite l'exécuter et avoir le résultat qui s'affiche automatiquement. Il faut prendre soin de sauvegarder le document, ici. Jupyter m'indique qu'un checkpoint a été effectué. Je peux même, entre deux sauvegardes, revenir à la dernière sauvegarde du document. Vous avez ici accès à l'aide. Je vous laisse regarder par vous-même.

Comme je vous l'expliquais, le principe est d'avoir des blocs de code qu'on peut exécuter, et que le résultat soit inséré automatiquement juste en dessous. Il faut faire relativement attention avec ce genre de choses parce que si je définis une nouvelle variable ici et que j'affiche son contenu, comme je pouvais m'y attendre, je vais avoir la valeur 10 qui va s'afficher. Si je modifie le contenu de ma variable, de cette façon-là, je récupère quelque chose qui a l'air normal, à savoir 20. Si je réexécute cette cellule, je récupère un autre résultat, parce qu'à chaque fois, mon environnement Python n'est pas réexécuté et l'état des variables est conservé. Je peux rapidement me retrouver avec un notebook dans un état incohérent. Pour cela, pas de panique, il suffit de réexécuter le kernel depuis le début pour récupérer un notebook dans un état cohérent. On réexécute l'ensemble des calculs.

Maintenant, au bout d'un certain temps, vous en aurez peut-être assez de faire des allers-retours avec les menus. Si vous accédez ici à l'aide, vous avez l'ensemble des raccourcis claviers. Vous verrez en particulier que shift enter et ctrl enter vous permettent de réexécuter les cellules et d'insérer une cellule directement en dessous, ce qui est pratique lorsqu'on exécute un ensemble de commandes rapidement. Je vais procéder de cette façon à partir de

maintenant. Petit exemple de complétion, maintenant. En Markdown. Mettons que je souhaite utiliser NumPy pour générer un ensemble de nombres aléatoires. Je vais donc charger la bibliothèque NumPy. À présent, Jupyter va me permettre de faire de la complétion automatique sur l'ensemble des objets et des fonctions. Si je souhaite générer des nombres aléatoires, avec tab, j'ai la complétion sur l'ensemble des fonctions qui commencent par ce que j'ai commencé à écrire. Je voulais générer des nombres suivant une loi normale.

Voilà. Pareil pour les paramètres de la fonction, je voudrais des choses centrées en Mu, d'écart type sigma. Je vais en générer, mettons, 10 000. Voilà. Je vais sauvegarder ces résultats dans une variable de façon à pouvoir vérifier, au moins visuellement, si mon hypothèse de normalité est raisonnable. Pour cela, je vais utiliser matplotlib. Comme pour NumPy, je vais le charger afin de bénéficier de l'autocomplétion. Je peux faire "plot", "histogram de x", regarder le graphe correspondant. Petit détail technique, précédemment, les objets que je récupérais étaient en général du texte. Ici, il va falloir indiquer à Jupyter que le résultat est un graphique matplotlib et qu'on souhaite qu'il soit intégré directement dans mon document.

Et voilà. Il existe tout un tas d'autres invocations magiques comme celle-là. Vous pouvez obtenir leur liste avec la commande lsmagic. Je vous laisserai découvrir tout ça. Ces commandes magiques vont être pratiques, en particulier pour interagir avec d'autres langages, par exemple avec le langage R. "Utilisation d'autres langages". Encore une fois, on a du code Markdown. Afin d'indiquer à Jupyter qu'on souhaite pouvoir exécuter des fragments de code R, voici la commande. On va charger un module qui va permettre une interaction entre Python et le langage R. Cela va me permettre d'écrire directement du code R ici, ce que j'indique avec ce "%%R". Si j'essaie d'afficher des données présentes par défaut dans le langage R, tout se passe très bien. Ça me permet d'avoir une interaction potentielle entre Python et R. On peut utiliser d'autres langages, du Shell, du Perl, toujours avec le même type de préfixes magiques, des "%%sh" ou "%%perl".

Enfin, un point important sur ce qui est du partage et de la production de documents. Ici, tous les résultats que j'ai obtenus sont stockés dans mon document. Chaque résultat apparaît ici. La façon la plus simple de partager ce document avec d'autres consiste à le "commiter" dans GitHub ou GitLab. Nous avons rajouté un bouton à cet effet, ici, qui va vous permettre de "commiter" votre document. "Un premier essai... avec Jupyter". Les données sont alors "commitées" et je peux les regarder sur GitHub ou GitLab. Mon document apparaît alors sous GitHub formaté de façon tout à fait agréable. Tous les résultats sont stockés dans le notebook.

Si je vais voir le contenu du fichier, vous verrez que c'est en réalité un document JSON. Vous retrouvez l'ensemble des images, des commandes qui ont été tapées, ce qui permet d'avoir une trace parfaite de ce qui a été exécuté. Si vous ne souhaitez pas partager vos fichiers avec GitHub ou GitLab, il est possible de passer par un export HTML. Ici, je vais télécharger mon document comme un document HTML et retrouver un document HTML complètement autocompris, autodécrit, avec les images qui sont à l'intérieur. Je peux l'envoyer par mail facilement et ce sera lisible par d'autres sans difficulté.

Je peux faire des exports PDF en passant par LaTeX. Tout ceci fonctionne bien. Si je souhaite préparer un document pour un besoin particulier, comme un article, je vais être embêté par le fait que tous ces morceaux de code apparaissent et que tous les résultats soient visibles. Dans ce cas, il est possible de contrôler la visibilité des cellules, ici, avec "Hide code". Je peux ainsi cacher soit les fragments de code, soit les sorties. Je peux créer un document dans lequel les calculs sont présents, mais pas visibles lors de l'export. Pour avoir un contrôle fin lors d'un export pour préparer un document pour un journal, on peut insérer directement des commandes LaTeX ou HTML, voire personnaliser les feuilles de style lors de l'export. Au lieu de passer par les menus pour exporter, en interne, on exécute ce genre de commande qu'on peut personnaliser.

Ça fait beaucoup d'informations en peu de temps. Je vous invite à vous saisir de cet environnement, à l'utiliser, à le découvrir, et à essayer de reproduire des documents similaires. Nous avons déposé dans votre espace personnel d'autres documents dont vous pouvez vous inspirer. À très bientôt.

4B. Prise en main de l'outil (R studio)

Dans cette séquence, je vous présente l'environnement RStudio, qui permet de créer des documents computationnels en utilisant principalement le langage R, ainsi que knitr, qui permet de transformer un document computationnel en un document classique.

Je viens donc de lancer l'environnement RStudio, ici, à gauche. La première chose que je vais faire, c'est ouvrir un document qui mélange du R et du Markdown. Pour cela, je vais dans "Fichier", "Nouveau fichier", "R Markdown". Il faut que je donne un titre à mon document. "Mon premier document". À ce moment-là, RStudio me fait un stub, avec un en-tête, des bouts de codes qui apparaissent ici en gris, ensuite, du Markdown, et si je descends un peu plus bas, encore des morceaux de codes en gris, du Markdown, des morceaux de codes en gris, du Markdown. C'est ce que je vous expliquais dans la séquence précédente.

Je vais sauvegarder mon document, lui donner un nom sans grande importance pour l'instant. L'extension pour les documents R Markdown est ".rmd". Je vais vous montrer comment travailler avec RStudio, mais si vous avez besoin de plus d'informations, dans la section "Aide", vous trouverez tout ce dont vous avez besoin, en particulier, un certain nombre de pense-bêtes faits par l'équipe de développement de RStudio, extrêmement bien faits. Je vous encourage à les regarder. Moi-même, quand je vais les consulter, j'apprends plein de choses.

Donc, dans vos documents, on trouve un en-tête, du code Markdown, du code R, un peu après, du code Markdown, du code R, ici à chaque fois en gris, du Markdown, du R et ainsi de suite. Exécutons donc chacun des morceaux, les uns après les autres. Ici, avec le petit triangle vert, je peux exécuter le chunk courant. Le résultat apparaît juste en dessous. Que s'est-il passé ? J'ai ici une console. Lorsque j'exécute ceci, le bout de code qui est ici est copié-collé dans la console et exécuté. Le résultat est récupéré et collé juste en dessous. Cela fonctionne aussi bien pour des sorties en mode texte, que pour des sorties en mode graphique, ce qui est assez confortable.

Il se peut, selon votre configuration de RStudio, votre version, que les sorties n'apparaissent pas directement ici, mais un peu sur le côté. Tout ça n'a pas grande importance. Ça se règle dans les paramètres de configuration. Je peux donc aller à la fin du document et commencer mon propre document. Pour cela, je vais insérer un petit morceau de code. Je peux alors exécuter des choses tout à fait triviales, comme cette opération, et définir des variables et observer leur contenu. Je peux insérer un nouveau bloc, travailler dessus, travailler sur mes variables.

Quelque chose auquel il faut faire attention, ceci dit. Je peux ré-exécuter ce bloc à peu près autant de fois que je veux. Vous voyez d'ailleurs les choses qui apparaissent dans la console. Je me retrouve avec quelque chose d'incohérent en première apparence, pour quelqu'un qui lirait ce document. Car j'ai une 1re déclaration de variable, des valeurs, puis, des résultats qui ont l'air incohérents. Le fait d'avoir une console et des données persistantes d'un bloc à l'autre vous permettra d'exécuter chacun des blocs dans l'ordre que vous voulez, et potentiellement d'arriver à des situations incohérentes. Pas de panique. Pour cela, il vous suffit d'aller ici et de ré-exécuter tous les chunks qui sont au-dessus de là où vous êtes. Vous vous retrouvez dans un état cohérent, qui correspond à ce qui est écrit dans votre notebook.

Au bout d'un moment, vous en aurez assez de cliquer à droite, à gauche, en haut, en bas. En fait, tous les raccourcis sont ici. Pour exécuter le chunk courant, "Ctrl+Shift+Enter", sur

ma machine. Sur la vôtre, ça peut être différent. Pour insérer un bloc, pareil, c'est ici. On arrive, du coup, à faire des choses assez rapidement, sans avoir besoin de se déplacer dans tous les sens. De même, vous avez ici un véritable environnement de développement fait pour R. Si j'utilise la fonction "rnorm", que j'évoquais tout à l'heure, j'ai la complétion, ainsi que l'aide, qui apparaît. Assez rapidement, vous allez vous retrouver avec du code, des données, du code, des données, un document assez long et difficile à exploiter.

Quelque chose qui est relativement pratique aussi, c'est ce que vous avez ici, qui permet de replier des zones assez facilement, voire de replier tout une section. Ce qui vous permet d'avoir un aperçu de l'ensemble du document assez simplement. De la même façon, tout ceci se fait avec des raccourcis clavier, comme ceci. Tout est dans les menus. Maintenant que j'ai créé mon document, que j'ai exécuté chacun des blocs, je ne pourrai pas partager ce document facilement avec d'autres personnes. Ils ne verront pas les mêmes choses. En particulier, là, j'ai généré des nombres aléatoires, sans faire attention à contrôler la graine de mon générateur. Mettons que je souhaite maintenant préparer un document que je puisse envoyer à un de mes collègues. Ici, vous avez ce petit logo, "Knit", qui transformera votre document en document HTML. Vous voyez, il est tout joli, avec toutes les commandes que nous avons exécutées, vos commentaires sur les analyses, etc.

Si vous appuyez sur "Publish", ici, l'ensemble de ce document sera envoyé sur les serveurs de RStudio, sur Amazon. Il sera alors simple de le partager avec des collègues en leur envoyant simplement l'URL. C'est comme ça que je travaille avec mes étudiants. Tout ce qui est rendu de devoirs ou de TP est fait via un document comme ça. Ils publient leur adresse et m'envoient le résultat final avec l'ensemble des codes qui ont été exécutés. Dans mon document, ici, vous voyez, j'ai mon graphique qui commence et le code n'apparaît pas. Il est possible, lorsqu'on prépare un document de spécifier si on souhaite que le code apparaisse, que le résultat apparaisse ou si on souhaite, au contraire, les cacher. Ce qui, lorsqu'on rédige un article, ou quelque chose de plus haut niveau, peut être utile. Si je vais dans la section qui a produit ce graphique, vous voyez qu'ici, le début de mon bloc R a un nom, ce qui permet de le nommer et d'y faire référence par la suite, et également une option, ici "echo=FALSE", qui permettra d'indiquer que le bloc ne doit pas être apparent lors de l'export.

Il peut être compliqué de connaître toutes les options, mais, encore une fois, RStudio vous aide et vous donne toutes les options possibles permettant de contrôler la largeur, la hauteur de votre graphique, la résolution, une caption. Tout est ici. Enfin, ici, je vous ai montré comment produire un document HTML. Mais il se peut que je veuille préparer plutôt un

article pour une revue, qui, dans ce cas-là, préfère avoir un document PDF. Aucun problème. Il suffit de changer le type de sortie. L'ensemble de mon document a été converti, cette fois-ci, en Markdown, en récupérant chacun des morceaux de code et des graphiques, et en les intégrant. Et au lieu d'exporter mon Markdown en HTML, le Markdown a été exporté en LaTex, lui-même compilé via pdfLaTex, et voici mon document.

De même, si je souhaite obtenir du Word, cela ne pose normalement pas de difficulté. Le document a été créé et apparaît ici, et a, à peu près, la même tête que tout à l'heure. Lorsqu'on passe par Markdown de cette façon-là, contrôler finement la façon dont la transformation d'un document final est effectuée n'est pas toujours simple. Si on a besoin de faire des mises en page bien spécifiques, d'utiliser un style bien particulier fourni par l'éditeur, cela peut être compliqué. Ceci dit, il est possible d'écrire, dans ce cas-là, si on souhaite faire du LaTex, directement du TeX et du R en même temps, ça s'appelle du R Sweave. L'extension de votre fichier, c'est du ".rnw". Ça se fait exactement de la même façon qu'avant. Ici, vous retrouverez du R Sweave, une façon de faire du LaTex, ou bien du Rhtml, si vous voulez écrire un document bien spécifique avec des balises bien spécifiques HTML.

Enfin, comme je vous l'avais expliqué dans la session précédente, RStudio est particulièrement adapté lorsqu'on fait du R, mais ce n'est pas le seul langage qui est supporté. "Avec d'autres langages". Ici, je peux insérer d'autres types de codes. Vous voyez ici du Bash, du Python, du C++. Cela fonctionne sur le même principe. Je devrais donc obtenir, si j'exécute ceci, une valeur, à peu près, un peu plus grande que zéro. Tout va bien. Je peux donc exécuter du code Python, vous voyez, de façon tout à fait normale. Néanmoins, si j'essaye de faire la même chose ici, ma valeur "x" étant définie, je risque d'avoir une mauvaise surprise. En effet, il n'y a pas de persistance des variables, de tout ce que vous avez pu charger, d'un morceau de code Python à un autre. En fait, à chaque fois que vous exécutez ici, une nouvelle console Python est créée. Le code est exécuté, le résultat est réinséré. Mais ça s'arrête là. La conséquence, c'est que si on veut faire des interactions entre plusieurs blocs de code Python ou entre du code Python et du code R, il est nécessaire de passer par des fichiers. Cela aura aussi pour conséquence que les morceaux de codes Python que vous écrirez auront tendance à être longs, si ce que vous faites est compliqué. C'est pour cela que c'est possible, mais ce n'est pas l'environnement le plus adapté pour faire du Python.

Ça fait beaucoup d'informations en assez peu de temps. Je vous ai montré l'essentiel de cet environnement. La meilleure façon d'apprendre, c'est de l'utiliser. À vous de jouer.

4C. Prise en main de l'outil (Org-mode)

Dans cette séquence, je vais vous présenter l'environnement Emacs/Org-mode, qui permet de combiner assez facilement différents langages. Sur la gauche, vous voyez ici un document Org-mode tel que vous le trouverez dans votre espace FUN et tel que vous allez l'ouvrir avec Emacs. Org-mode est un langage de balisage léger comme ceux que nous avons vus précédemment avec Markdown. La syntaxe est légèrement différente.

Lorsque j'ouvre mon document, il est généralement présenté complètement replié : j'ai plusieurs sections, sous-sections, sous-sous-sections, et chacune de ces sections est repliée. En appuyant sur Tab, je vais pouvoir déplier chacune de mes sous-sections pour observer le contenu qui m'intéresse. Ça me permet d'avoir un premier aperçu et de regarder une partie spécifique.

Comme c'est un langage de balisage léger, je vais pouvoir facilement mettre du gras ou de l'italique exactement de la même façon. Emacs est un éditeur de texte puissant, doté de nombreux raccourcis clavier. Je ne ferai pas une démonstration de tous ces raccourcis. Néanmoins, ils sont disponibles dans FUN, la majorité, et lorsque j'utilise des choses spécifiques à Org-mode et qui permettent d'accélérer les choses, vous les verrez en surimpression.

Passons au cœur du document computationnel et laissez-moi vous montrer comment exécuter des calculs et récupérer les résultats. Il y a un certain nombre de raccourcis clavier par défaut. En particulier, pour faire du R, il y a un template prédéfini. Si je commence à taper ceci et que j'appuie sur "tab", ça va être complété automatiquement en ce bloc de code, dans lequel je vais pouvoir écrire du code R tout à fait normalement. Si je cherche à regarder un premier jeu de données, j'écris mon code ici et je l'exécute avec le raccourci "Ctrl+c", "Ctrl+c". Le résultat apparaît alors juste en dessous. Ça ne se limite pas à des sorties textuelles. Un raccourci du même genre, avec un grand R, va me permettre d'avoir une complétion et un bloc préparé pour une sortie graphique. Ainsi, de la même façon, j'aurai mes calculs et mes résultats apparaissant les uns après les autres.

Comment tout ce code est exécuté ? En interne, Emacs a ouvert une session R, accessible ici, en inspectant les buffers. Vous y retrouvez les commandes que j'ai exécutées, ainsi que les résultats. Et donc, ce qui se passe lorsque je viens ici et que j'exécute des commandes, quand je tape "Ctrl+c", "Ctrl+c" : ce code est copié-collé dans la session, le résultat est récupéré et collé en dessous. C'est relativement efficace et pratique, mais ça peut comporter quelques risques. Si je définis une variable ici et que j'inspecte son contenu, le résultat est 10, comme je

pourrais m'y attendre. Par contre, si maintenant je modifie le contenu de ma variable et que je regarde son résultat, j'obtiens 20, ce qui paraît normal, à ceci près que je peux réexécuter ce bloc-là et obtenir quelque chose relativement incohérent avec ce qui a été écrit au-dessus. C'est dû au fait que ma session a un état et que je suis en train de modifier cet état petit à petit. Pas de panique, il suffit de réexécuter l'ensemble des codes depuis le début, ce que je vais faire de cette façon-là. "org-babel-execute-buffer". Et voilà, l'ensemble de mes commandes vient d'être réexécuté.

Je vous expliquais qu'il était possible d'avoir plusieurs langages. Laissez-moi vous montrer ceci. De la même façon que j'ai des raccourcis comme celui-ci pour avoir du R, je peux avoir des raccourcis comme celui-ci pour avoir du code Python et je vais pouvoir les exécuter de la même façon, très simplement, avec les mêmes raccourcis clavier que précédemment. Je vais pouvoir également exécuter des commandes shell. Par exemple, regarder ce qu'il y a dans un répertoire. Et faire des choses même plus compliquées, le gros avantage étant que, potentiellement, j'ai une session, et donc un historique très riche sur les commandes que j'ai tapées et des sorties qu'il y a eu.

Si, par exemple, ici, je demande sur quelle machine je suis, il s'agit de ma machine, elle s'appelle Icarus. Mais je peux parfaitement décider d'aller sur une autre machine. Donc là, je me connecte via SSH sur une machine distante. Vous voyez, j'y suis arrivé. Et à partir de maintenant, les commandes que je vais exécuter vont être exécutées à distance. Ici, vous voyez que je suis sur une autre machine, nipmuk, que je peux voir ce qu'il y a dans son tmp et que ce n'est pas ce que j'avais avant. Il est donc très facile en Org-mode d'avoir plusieurs langages, voire plusieurs sessions : je peux très bien avoir une session R, une session shell, voire une autre session shell, l'ensemble va être nommé. Donc, c'est extrêmement puissant.

Par contre, c'est un outil à maîtriser et à utiliser avec précaution. Maintenant, l'ensemble de mes résultats, ici, apparaît. Néanmoins, pour ce qui est de l'image qui est ici, il s'agit en fait d'un lien vers un fichier. Donc, lorsque je vais souhaiter partager un document comme celui-ci, je vais pouvoir le commiter, faire un commit, un push, l'inspecter. Je vais avoir un affichage assez joli dans GitLab, vous pouvez regarder vous-même. Néanmoins, il faudra prendre soin de ne pas stocker que ce document-là, mais aussi les fichiers que vous avez produits.

Maintenant, si vous souhaitez partager votre document, peut-être pas permettre à d'autres d'exécuter les choses, mais simplement de voir le contenu, une solution également assez simple consiste à l'exporter, par exemple en HTML ou en PDF. Pour ça, il existe un certain

nombre de raccourcis clavier, comme celui-ci. Ici, je souhaite exporter en HTML et ouvrir le résultat. Vous voyez qu'ici, je retrouve l'ensemble de mon document et des choses que j'ai exécutées. Je peux faire la même chose en PDF, ça marche tout aussi bien.

Enfin, il va être parfois intéressant de contrôler la visibilité de certaines choses que l'on montre. Par exemple, ici, peut-être que je ne souhaite pas faire apparaître la commande exécutée. C'est donc à ce niveau-là, au lieu d'écrire "both" ici, qui exporte à la fois le code et le résultat, que je vais indiquer que je ne suis intéressé que par le résultat. Quand je vais exporter, cette fois-ci, vous voyez que la partie de code qui était visible avant a disparu. Donc, ça va me permettre de créer des documents avec des parties de code cachées, d'autres visibles, de façon à montrer uniquement les choses qui me semblent importantes. D'autre part, indiquer, pour chacun des blocs, s'ils doivent être visibles ou pas peut être un peu pénible. C'est pour ça qu'il est aussi possible de carrément enlever toute une section et d'indiquer qu'on ne souhaite pas qu'elle soit visible lors de l'export. C'est ce que je fais ici en indiquant ce tag "noexport". Et maintenant, si je réexporte mon document en HTML, vous voyez que je ne trouve plus que la partie sur les autres langages et que la partie sur R a disparu du document. Donc, c'est très flexible et puissant et ça me permet de rédiger aussi bien des articles, des cahiers de laboratoire ou des notes techniques. Comme je vous le disais, il est possible d'exporter en PDF, en LaTeX, en HTML, et ce qui va être important, c'est qu'on peut reprendre le contrôle et contrôler assez finement comment l'export se fait, insérer directement du code LaTeX ou des commandes HTML, contrôler les feuilles de style.

Ça fait beaucoup d'informations en peu de temps. Clairement, la difficulté principale d'Emacs/Org-mode, c'est de maîtriser l'ensemble des raccourcis clavier, qui sont très nombreux. Pour être honnête, je n'en maîtrise qu'une petite partie, et ça suffit très bien au jour le jour. Néanmoins, l'éditeur est puissant, rapide, et vous permet de combiner énormément de choses, tous les outils que vous connaissez déjà. C'est la raison pour laquelle aussi bien Konrad que Christophe et moi-même l'utilisons au quotidien et le préférons aux deux autres environnements que nous vous présentons. Il ne vous reste plus qu'à mettre tout ça en pratique.

5. Travailler avec les autres

Nous avons vu différents environnements de travail permettant de générer des documents computationnels, nous allons aborder la question du travail avec d'autres personnes. Mettons

que vous deviez préparer un document pour un journal, par exemple un PDF. Dans les 3 environnements que nous avons vus, c'était facile, il suffisait de cliquer au bon endroit, mais toute la complexité menant à cette transformation est cachée. En interne, l'outil utilise soit Pandoc, soit knitr, soit Emacs/Org-mode. En général, on passe par un document en LaTeX, donc il doit être correctement installé pour pouvoir produire le PDF. Si ces dépendances ne peuvent être installées sur votre machine, il est toujours possible d'exporter vers OpenOffice ou Word. Pour Jupyter, c'est possible aussi, mais ça demande du travail. Sinon, l'export HTML fonctionne très bien. Reste à voir si cela correspond à ce dont le journal a besoin.

Dans tous les cas, vous devrez décider quelles cellules seront apparentes et lesquelles cachées, quelles figures doivent être visibles et lesquelles doivent rester cachées pour votre analyse, et tout configurer pour utiliser le bon style.

Bref, soyons honnêtes, produire un document prêt à l'impression à partir d'un document computationnel demande un environnement parfaitement configuré, et c'est un peu de travail. Comment convaincre vos coauteurs de faire cet effort ? Il y a plusieurs réactions possibles.

La première, c'est l'enthousiasme : "Aucun problème, ça me plaît, j'adopte ça, je m'y mets." La deuxième, c'est la bienveillance, on vous dit : "C'est un bon outil, néanmoins, je n'ai pas le temps d'apprendre à l'utiliser. Je ne peux pas l'installer sur ma machine, c'est trop compliqué, mais je suis prêt à faire des efforts". Enfin, la troisième, qui est assez fréquente, c'est : "Un nouvel outil ? Ça change tout le temps, je n'apprendrai pas de nouvel outil. On travaillait bien avec les outils précédents, pourquoi changer ?"

Face à cela, il y a plusieurs organisations possibles. Regardons la première, l'enthousiasme, la plus appréciable a priori, à ceci près que vous apportez un nouvel outil, vous devrez assurer le service après-vente, donc vous assurer que votre document computationnel fonctionne dans les différents environnements, Mac, Windows, Linux, et que vous obtenez le même résultat quelle que soit la machine utilisée.

Ensuite, les outils que nous proposons, Jupyter, RStudio, Emacs ou Git, demandent un peu de prise en main, donc il faudra gérer cette complexité. Ceci dit, une fois ce cap dépassé, il faut être conscient qu'avec plusieurs coauteurs qui utilisent le même environnement, on a plusieurs yeux qui vérifient que tout fonctionne et s'assurent que tout est reproductible et inspectable, et pas juste sur votre propre machine.

Deuxième option, vos coauteurs sont bienveillants et prêts à faire des efforts, mais ne peuvent pas tout installer. Ils vous laisseront gérer le code, les résultats, les figures, mais sont prêts à éditer votre document. S'il s'agit d'un notebook Jupyter hébergé sur une plate-forme

cloud, tout le monde peut éditer en même temps. S'il s'agit de RStudio ou d'Emacs, ils doivent être installés sur leur propre machine. Si ce n'est pas le cas, ils peuvent éditer le texte de l'article puisque c'est du Markdown ou Org-mode, n'importe quel éditeur de texte suffit. Ils ne pourront pas refaire les calculs, faire l'export et voir le document final. J'ai vécu cette situation, en général, ça se passe bien. La seule contrainte est que je suis responsable à la fin de faire l'export et de soumettre l'article.

La dernière option, relativement courante, des coauteurs réfractaires, qui ne veulent ou ne peuvent pas, faute de temps, changer leurs habitudes. Je recommande l'approche suivante : avoir un document computationnel séparé produisant les résultats et les figures et un autre document classique incluant les figures générées par le document computationnel. Ainsi, tout est conservé, documenté, recalculable, dans le document computationnel. Et vos coauteurs n'ont aucun effort à faire. Lorsqu'un des revieweurs ou un coauteur demande à faire une modification sur une figure, vous prenez le document computationnel, vous le modifiez, vous réexportez, et les modifications apparaitront automatiquement dans le document classique.

Maintenant, une fois que vous avez trouvé votre mode opératoire, comment partager votre document avec des collègues ? Ceux qui ont suivi la séquence sur RStudio ont entendu parler de RPubs. C'est l'outil idéal pour partager de façon très efficace et rapide un document que vous venez de produire. Mais ce n'est pas une approche pérenne. Les données sont hébergées sur Amazon et l'url n'est pas forcément pérenne. Pour la pérennité, il faudra utiliser autre chose. On a déjà évoqué les possibilités d'utiliser Dropbox ou autres. Le problème de la pérennité se pose à nouveau. Le contrôle d'accès de ces documents par d'autres personnes va poser problème. Si vous utilisez des outils comme GitLab ou GitHub, ce que nous recommandons, une fois que vous avez fini de travailler sur votre dépôt privé, que vous souhaitez rendre l'article public, il est possible de rendre l'ensemble du dépôt public. Cela veut dire que vous rendrez également public votre historique, ce qui n'est pas souhaitable si vous ne l'aviez pas prévu, en particulier si vous avez fait des commentaires désobligeants sur les rapporteurs, si vous avez stocké dans votre dépôt des documents sous copyright.

Bref, si c'est l'approche que vous souhaitez adopter, il vaut mieux le prendre en compte dès le départ et construire votre dépôt en gardant à l'esprit qu'à terme, les choses seront publiques. Vous pouvez aussi faire le ménage dans votre dépôt, archiver l'état courant dans un site compagnon. Qu'est-ce qu'un site compagnon ? Certains éditeurs ou domaines proposent des services d'hébergement afin de mettre à disposition votre code et vos données. C'est le cas de MyCode ou de certains éditeurs.

Plus pérennes, des archives ouvertes, comme HAL, qui est soutenue par l'État français, ou figshare et Zenodo, Zenodo étant soutenue par le CERN. Ces outils d'archivage vous permettront de déposer votre document final, le PDF, le HTML, et de déposer des documents en annexe, comme des images ou du code. En conclusion, plusieurs modalités de fonctionnement sont possibles.

Choisissez celle qui vous convient en fonction de vos coauteurs, de vos contraintes techniques, de confidentialité et de copyright. Lorsque vous essaierez de mettre tout cela en œuvre, vous rencontrerez peut-être des difficultés. Il y a des solutions, n'abandonnez pas.

6. Analyse comparée des différents outils

Dans cette séquence, je vous présenterai une analyse comparée des trois différents environnements que je vous ai présentés jusqu'ici. Nous avons vu ensemble Jupyter, RStudio, Org-mode. Lequel utiliser ?

Tout dépend du type de documents que vous souhaitez réaliser.

Une première possibilité est celle d'un cours ou d'un tutoriel. Lorsque je donne des TD avec des étudiants, il est fréquent que nous déployions un notebook Jupyter qui va leur permettre très facilement d'avoir accès au même environnement et de pouvoir ensemble co-construire la solution d'une feuille de TD, ensemble, pendant le TD, et de leur mettre à disposition à la fin. C'est le cas ici, vous voyez, d'un TD de proba sur les générateurs. Ils ont accès à l'ensemble du code, aux résultats. Et la partie intéressante, c'est qu'ils vont pouvoir modifier certaines parties de ce document, les réexécuter, pour bien comprendre les concepts sous-jacents à telle ou telle fonction et réessayer de nouvelles choses. Le gros avantage ici est d'avoir un document facile à prendre en main et complètement dynamique.

Une autre possibilité, un autre cas d'étude, c'est le cas d'un journal. Qu'est-ce qu'un journal ? Je vais vous présenter mon journal en Org-mode. Donc, régulièrement, depuis 2011, je prends des notes sur les réunions auxquelles je participe, les articles que je peux lire, des morceaux de code que je peux écrire pour telle ou telle raison. On est dans le cas ici d'un document rédigé principalement par un seul auteur : moi-même. Donc, j'ai une organisation chronologique. Vous voyez qu'ici, pour chaque année, j'ai des entrées, et je vais pouvoir déplier certaines de ces années, aller à n'importe quel mois, et ainsi de suite, à une année. La présentation d'un étudiant, ici. Ici, un keynote donné avec un collègue, et ainsi de suite. Et vous voyez que pour chaque entrée, éventuellement, je vais avoir des étiquettes. Ici, vous

voyez, j'ai une étiquette sur des aspects autotuning. Donc, cette organisation chronologique me permet de savoir où je prends note de telle ou telle chose. Les choses se mettent à la fin, dans l'ordre.

Par contre, lorsque je recherche quelque chose, je vais utiliser le mécanisme d'étiquettes. Org-mode propose des mécanismes pour filtrer facilement les entrées qui correspondent à une étiquette. C'est ce que je vais faire. Je vais chercher, par exemple, tout ce qui concerne le langage R. Voilà, ici, j'ai une vue filtrée, avec tout un tas d'entrées, des articles sur la recherche reproductible, des travaux sur des Gantt charts. Et si je prends, par exemple, celui-ci... Ici, on a un collègue qui m'avait envoyé des données qu'il souhaitait analyser et qui ne savait pas forcément comment bien les représenter en R. Donc, j'ai copié-collé les données, je les ai reformatées, et puis j'ai écrit un bout de code R permettant de faire une visualisation qui correspondait à ce qu'il souhaitait. Le code n'est pas complètement trivial non plus, donc j'en ai gardé note, au cas où j'aurais à refaire ce genre de choses plus tard. Encore une fois, ici, je n'ai pas stocké le résultat, uniquement le code, cela suffit. Il suffit juste de réexécuter ça. Vous voyez, il récupère les données, il exécute le code R, génère l'image, et vous voyez le résultat qui apparaît juste ici.

Donc, dans mon journal, je vais trouver à la fois des notes sur des réunions, des liens hypertextes suite à des articles que j'ai lus, des références, des pages Web, des liens vers des documents PDF, et puis des morceaux de code. L'ensemble est relativement hétérogène mais je vais pouvoir retrouver tout cela assez facilement. Un troisième cas d'étude : celui d'un cahier de laboratoire, ici, encore une fois en Org-mode. Cette fois, on n'a plus une organisation chronologique, on aura une organisation sémantique dans laquelle on va indiquer les conventions d'organisation de ce document. Ici, j'ai une première section avec tout un tas de scripts, où je vais expliquer comment j'accède à des informations sur telle machine. Je vais trouver également des scripts de conversion d'un format à l'autre, des scripts de traitements en R. Tout est regroupé comme ça au début. Ce sont les fonctions les plus courantes. On écrit ces choses-là et puis on les affine, et quand elles sont relativement stables, on les déplace en haut du document pour qu'elles soient communes et que tout le monde y ait accès. D'autre part, on va trouver une section avec des expériences, une section avec des analyses. C'est un choix, on pourrait en imaginer d'autres. Dans ces sections, on va trouver une organisation plus chronologique. On a plusieurs auteurs, qui sont précisés ici : moi-même et deux autres collègues. Je vais donc pouvoir facilement trouver les entrées... que j'ai écrites moi-même. Ici, la dernière concerne une analyse qui fait elle-même référence à quatre expériences, la 26, la

25 et la 24. Donc, si je cherche, encore une fois, par exemple l'expérience 26, je vais trouver toutes les entrées qui correspondent à l'expérience 26, et ainsi de suite. Donc, j'ai des liens dans mes documents qui vont me permettre de m'y retrouver et des conventions qui doivent être explicitées au début du document.

Considérons un dernier cas d'étude : celui d'un article reproductible tel que je peux en écrire moi-même. Il s'agit ici d'un article en cours de préparation. Vous avez ici un aperçu de l'article. Vous retrouvez une partie introduction, une partie contexte, l'état de l'art, et ainsi de suite. Sur la fin, des sections qui apparaissent avec le tag "noexport", qui ne seront donc pas visibles lors de l'export. Et vous allez voir qu'il y en a tout au long du document, des sections comme ceci. J'ai d'autres sections dans mon document qui sont cachées et dans lesquelles je vais avoir des morceaux de code qui me permettent de générer telle ou telle figure. Donc, ce document, qui est complètement en Org-mode, je vais pouvoir l'exporter en LaTeX. Le voici qui s'ouvre. Et donc, il a une tête parfaitement normale d'article, avec le style de la conférence en question. Maintenant, imaginons que je souhaite modifier cette figure-là. Il s'agit de celle qui parle de... "Replacing the calls". Voilà. Ici, en cherchant dans mon document, je vais assez rapidement retrouver la partie LaTeX qui correspond. Ici, j'ai du code LaTeX directement pour avoir un double colonnage, les sections, etc., exactement comme je souhaite que les choses soient indiquées. Je vois qu'il s'agit ici d'une figure nommée "validation_kernel_modeling". Si je cherche avant dans mon document "validation kernel modeling", je vais trouver le bout de code qui le génère. Il y a en fait deux zones de code. Une première ici, que je vais exécuter. Voilà. Et une deuxième, ici, sur laquelle imaginons que je souhaite modifier l'apparence de cette figure. Pas sûr que ce soit beaucoup plus joli comme ça, mais allons-y. Je regénère cette figure-là. Je peux ouvrir le document en question. Vous allez voir que l'apparence a complètement changé. Si je réexporte mon document, les modifications vont apparaître directement. Ce type de possibilités est évidemment très pratique lorsque vous préparez un document, que ce soit pour une version finale ou lors d'un "review send", où un reviewer va vous demander de modifier telle ou telle chose. D'autre part, si je remonte un peu plus haut, vous voyez qu'ici, au-dessus des bouts de code expliquant comment générer la figure, je vais avoir un lien vers l'entrée du journal correspondant, vers GitHub, qui indique l'entrée d'où je suis reparti, expliquant comment les données ont été générées et comment on a transformé ces données pour arriver à telle ou telle figure. On a donc une approche qui nous permet de partir du document, et si je publie mon document Org-mode, un lecteur averti va être capable de revenir jusqu'aux données brutes qui sont derrière

chaque figure. En résumé, comment se comparent ces environnements ? Tous trois permettent de produire des documents computationnels, de mélanger différents types de langages, de faire des exports HTML ou PDF. Tous les trois sont relativement stables et matures, c'est pour ça que nous vous les présentons. Certains sont plus anciens, et cela a des conséquences sur les technologies utilisées. Jupyter est une application Web, ce qui le rend très facile d'accès. C'est principalement centré sur le langage Python, avec une communauté active. RStudio est un environnement de développement intégré développé en Java mais qui exploite à fond les fonctionnalités du langage R.

Org-mode, enfin, est un éditeur, pas spécialement spécifique à un langage en particulier, même s'il est développé en Emacs Lisp. Tout ceci a des conséquences sur la facilité d'utilisation de ces environnements. Jupyter et RStudio sont, à mon avis, les plus faciles d'accès. Org-mode reste un peu plus complexe, même s'il demeure assez puissant, en particulier concernant la navigation.

Comme je vous l'ai montré précédemment, la navigation et la recherche d'informations dans Org-mode est très puissante. Une différence assez fondamentale aussi tient au format utilisé. Jupyter utilise du format JSON, qui est un format texte, mais orienté machine : on va retrouver à l'intérieur l'ensemble des sorties texte, des images, qui vont être encodées en mode texte à l'intérieur. Cela rend la lecture par un humain un peu délicate si on cherche à lire directement le document, donc il faudra quelque chose pour le prévisualiser. En revanche, RStudio et Org-mode utilisent un format de texte enrichi, un peu comme Markdown ou Org-mode. Un dernier point, peut-être, tient au fait qu'il soit possible de générer un article qui puisse être imprimé par un éditeur en respectant une feuille de style spécifique. Même si c'est possible dans ces trois environnements, dans Jupyter, c'est très difficile, et je ne connais personne qui l'ait déjà fait. En revanche, j'ai rédigé des articles de bout en bout dans RStudio et dans Org-mode, et cela devient courant.

Au final, l'outil importe peu. Ce qui importe, c'est de collecter l'information, de l'organiser et de la rendre exploitable, c'est-à-dire qu'elle soit exploitable pour soi-même dans une semaine, dans un mois, dans un an, et exploitable pour d'autres personnes. Ça va complètement changer la façon dont on va rédiger le document. Enfin, il va falloir la rendre disponible, soit via des plateformes comme GitHub et GitLab, soit via des entrepôts de données.