Méthodologie statistique

M 2014/01

La microsimulation dynamique : principes généraux et exemples en langage R

Didier Blanchet

Document de travail

Institut National de la Statistique et des Études Économiques

INSTITUT NATIONAL DE LA STATISTIQUE ET DES ÉTUDES ÉCONOMIQUES

Série des documents de travail « Méthodologie Statistique »

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

M 2014/01

La microsimulation dynamique : principes généraux et exemples en langage R

Didier Blanchet*

La rédaction de ce document a débuté dans le cadre d'une mission de coopération auprès du Haut Commissariat au Plan marocain. Elle s'est poursuivie en lien avec la construction du modèle PENSIPP de l'Institut des Politiques Publiques (IPP), dérivé du modèle Destinie de l'Insee. Je remercie les participants à ces différents projets, et tout notamment Antoine Bozio, Alexis Eidelman et Simon Rabaté. Ce texte a également beaucoup bénéficié des relectures successives de Pauline Givord, ainsi que de conseils de Magali Beffy, Carole Bonnet, Pierre Courtioux, Malik Koubi, Sophie Pennec et Anne-Sophie Robillard. L'auteur reste néanmoins seul responsable des erreurs ou omissions.

---

*Insee et Crest/IPP

Direction de la méthodologie et de la coordination statistique et internationale -Département des Méthodes Statistiques -
Timbre L101 - 18, bd Adolphe Pinard - 75675 PARIS CEDEX - France -
Tél. : 33 (1) 41 17 66 01 - Fax : 33 (1) 41 17 61 97 - CEDEX - E-mail : L001-dg@insee.fr - Site Web Insee : http://www.insee.fr

La microsimulation dynamique :
Principes généraux et exemples en langage R

Résumé

La microsimulation dynamique permet la modélisation de phénomènes complexes dans lesquels la variabilité des caractéristiques et des comportements individuels joue un rôle important. Son principe est très simple mais sa mise en œuvre peut soulever quelques problèmes méthodologiques qui ne sont pas toujours détaillés dans les descriptions des modèles existants. Par ailleurs, une question centrale pour la construction d'un modèle est le choix d'un langage de programmation. Quelques plateformes dédiées sont désormais disponibles. La simplicité de la méthode permet aussi des développements ex nihilo dans des langages de programmation généralistes, comme cela avait été fait pour les deux versions successives du modèle Destinie construites à l'Insee depuis le milieu des années 1990. Une troisième possibilité est de s'appuyer sur un logiciel statistique. Ce document propose une introduction générale à cette technique de modélisation et à ses quelques difficultés méthodologiques, et divers exemples de ce troisième style de programmation, rédigés en langage R, incluant un début de transcription, en R, du cœur du module démographique de Destinie.

Mots clés : Microsimulation dynamique, langage R, modèle Destinie

---

Dynamic microsimulation: general principles and examples in R

Abstract

Dynamic microsimulation allows modelling complex systems where the variability of individual characteristics and behaviours plays a dominant role. Its basic principle is extremely simple but its implementation raises some methodological issues that are not always documented in descriptions of existing models. In addition, a major question for model builders is the choice of a programming language. Some dedicated platforms now exist. The simplicity of the method also allows starting from scratch with generalist languages and this is the choice that had been retained for the two successive versions of the Destinie model developed at Insee since the mid 1990s. A third option is to rely on a statistical package. This document provides a general introduction to this modelling technique and to its few methodological difficulties, and some examples of this third programming style, written in R, including a partial adaptation, in R, of the core of Destinie's demographic module.

Keywords : Dynamic microsimulation, R language, Destinie model

Codes JEL :C63, C88, J11

Table des matières

Introduction

Ce document a deux objectifs : proposer une introduction à la microsimulation dynamique et aux quelques questions méthodologiques qu'elle soulève et en présenter des exemples de mise en œuvre en langage R. Ce langage n'est qu'une possibilité parmi beaucoup d'autres pour le développement de modèles de microsimulation, il existe même des langages qui ont été spécialement développés pour cette classe de modèles, mais le recours à R peut présenter certains avantages, notamment lorsqu'on utilise déjà ou envisage d'utiliser ce langage pour ses autres travaux statistiques.

Le texte est organisé de la façon suivante. Une première partie revient rapidement sur le principe général de la méthode. Elle en rappelle les origines en la situant par rapport aux autres formes de modélisation auxquelles elle peut se substituer ou qu'elle vise à compléter. Elle en présente les différentes variantes et leurs applications. On donne en même temps un premier aperçu des problèmes techniques qu'elle peut poser. Son intérêt est la simplicité conceptuelle et la capacité à rendre compte de phénomènes complexes difficiles à capter par des approches à individu représentatif et *a fortiori* difficiles à modéliser analytiquement. Ses limites sont le caractère stochastique de son résultat et la difficulté à calibrer les modèles dès lors qu'ils sont de grande taille : on verra quelles solutions partielles peuvent être apportées à ces deux difficultés.

La deuxième partie passe ensuite aux premiers exemples de mise en œuvre en langage R avec plusieurs versions successives d'un modèle élémentaire se limitant à simuler les entrées/sorties du chômage au sein d'une population constante. L'intérêt de cette partie ne réside pas dans le modèle qui y sera présenté : il sert juste de support à la présentation des principaux éléments du langage R et d'une première bibliothèque de fonctions R mises au point pour ce document et visant à faciliter les opérations de microsimulation les plus courantes.

La dernière partie propose une application plus opérationnelle, à savoir un essai de transcription en R d'une partie du bloc démographique du modèle Destinie 2 de l'Insee. L'appui sur R a permis de donner à ce module une forme très compacte, ce qui en permettra une présentation exhaustive. On explicitera et on justifiera notamment les choix retenus en matière d'organisation des données et la façon dont on a cherché à caler le modèle sur la dernière version des projections démographiques publiées par l'Insee (Chardon et Blanpain, 2010).

Ce document peut ainsi se lire de deux manières. La première partie vise des lecteurs souhaitant découvrir ou compléter leur connaissance générale de la méthode. Les deux autres ne concernent que des lecteurs souhaitant la mettre en oeuvre, en langage R ou dans des langages permettant le même style de programmation. Dans ces deux dernières parties, le choix a été de présenter la quasi-totalité des codes sources des programmes, en faisant l'hypothèse que la mise en œuvre de la méthode se comprend mieux à partir d'exemples simples mais complets. La lecture de ces codes sources sera d'autant plus facile que le lecteur connait déjà le langage R, mais ils sont rédigés d'une façon qui doit permettre de les suivre sans connaissance préalable de ce langage. Pour les lecteurs moins familiers de R, les annexes apportent des précisions complémentaires sur quatre de ses aspects : (a) la manipulation des masques ou filtres logiques permettant de sélectionner des éléments d'une variable ou d'un tableau, (b) la gestion des variables globales et locales,

(c) les fonctions graphiques et enfin (d) l'accès général à R et à l'environnement de développement associé RStudio. Le lecteur ayant besoin de davantage d'informations sur le langage est renvoyé aux ouvrages spécialisés, notamment Lafaye de Micheaux *et al.* (2011) ou la documentation en ligne disponible sur le site du *Comprehensive R Archive Network* (http://cran.r-project.org/) sur lequel se fait également le téléchargement du langage.

On propose par ailleurs une bibliographie générale assez fournie mais qui ne peut prétendre à l'exhaustivité car la littérature consacrée à la microsimulation se présente sous une forme assez éparse : ce n'est que depuis une dizaine d'années que le domaine s'est structuré autour de sa propre revue, l'*International Journal of Microsimulation* (www.microsimulation.org/IJM/). La liste de références qui est donnée en fin de document présente donc probablement de nombreuses omissions. Mais elle donnera au moins une bonne idée de l'étendue des domaines couverts par la méthode.

1 La microsimulation dynamique : principes et problèmes généraux

1.1 Historique et typologie des modèles

Rendre compte de la dynamique de systèmes complexes par la simulation informatique de leurs composants élémentaires est une idée assez naturelle qu'on rencontre dans des domaines extrêmement variés. Dans le domaine économique, sa paternité est en général attribuée à Orcutt (1957). Orcutt oppose sa démarche à celle de l'individu représentatif adoptée par la modélisation macroéconométrique. L'idée était de rendre compte des interactions entre les comportements d'agents individuels ou collectifs correspondant aux principaux types d'acteurs économiques : ménages, entreprises, acteurs publics. L'approche devait être dynamique et récursive : à chaque instant, un certain nombre d'agents adoptent des comportements ou prennent des décisions sur la base de l'état du monde qu'ils constatent, et il en résulte un nouvel état du monde pour la période suivante. Ceci supposait à la fois une modélisation du comportement de ces agents mais aussi une simulation de leur apparition et de leur disparition éventuelle, donc une simulation de la démographie et du regroupement des individus en ménages, et une simulation de la dynamique des firmes.

Le projet était très ambitieux compte tenu des moyens informatiques de l'époque. Le recours à la microsimulation dynamique s'est assez précocement développé dans le domaine démographique, notamment pour la simulation du processus de formation des familles. Un modèle pionnier et qui fonctionne encore a ainsi été le modèle SOCSIM dédié à la simulation des effets de différents régimes démographiques sur les structures familiales, développé dans les années 1970 (Mason, 2011). Pour des vues d'ensemble sur ces applications démographiques, le lecteur pourra par exemple se référer à Van Imhoff et Post (1997) ou Morand *et al.* (2010).

Dans le domaine économique, en revanche, beaucoup d'applications se sont longtemps cantonnées à l'approche statique et comptable, avec des modèles consacrés à l'analyse de la fiscalité et des transferts sociaux. Pour la France, le premier en date a été le modèle Sysiff (Bourguignon *et al.*, 1988). Plusieurs autres modèles de ce type sont présentés dans le numéro spécial consacré à ce thème par la revue *Économie et Prévision* au début des années 2000 (Legendre *et al.*, 2003) dont le modèle Ines, développé et utilisé par l'Insee et la Drees (Albouy *et al.*, 2003), et le modèle Myriade de la CNAF (Legendre *et al.*, 2001). Le modèle TAXIPP développé plus récemment à l'IPP appartient également à cette famille (Landais *et al.*, 2011, Bozio *et al.* 2012). La modélisation statique est aussi utilisée pour l'évaluation des politiques de transferts dans le domaine de la santé (Breuil-Genier, 1998; Lachaud *et al.* 1998; Debrand et Sorasith, 2010; Geoffard et de Lagasnerie, 2012). Le fait que la microsimulation se soit d'abord imposée dans ces domaines est facile à comprendre. Les barèmes fiscaux et sociaux sont des barèmes complexes et les effets qu'on veut en mesurer sont des effets au niveau des individus ou des ménages. La démarche la plus naturelle pour traiter ces problèmes est donc bien de partir de fichiers de ménages, avec les diverses caractéristiques en termes de revenu et d'emploi de leurs membres puis à simuler, sur ces ménages, l'impact de règles variables de prélèvements et de transferts. On en dérive les gains ou pertes individuels à la modification de ces transferts, ainsi que leur bilan total pour les finances publiques. Cela a un sens de le faire dans un cadre statique donnant une photographie de ces transferts à une date donnée et permettant sa comparaison avec les photographies qui auraient résulté de la mise en œuvre d'autres barèmes sociaux et fiscaux.

Le passage à l'approche dynamique consiste à ajouter à ces modèles une dimension temporelle. Ceci peut se faire dans un simple but d'actualisation des résultats du modèle, lorsqu'il s'appuie sur une base de

données ancienne, ou pour des besoins de projection à court-terme. On peut se contenter pour cela de simples repondérations des observations de la base initiale (voir par exemple Cai *et al.*, 2006). La modélisation devient véritablement dynamique lorsqu'on fait évoluer à moyen et long terme les caractéristiques des individus qu'on simule et tel doit être le cas lorsqu'on s'intéresse à des transferts dont l'incidence ne peut s'apprécier que dans la durée, ce qui sera typiquement le cas des retraites. Une complémentarité s'est ainsi développée entre usages démographiques et économiques de la microsimulation (Dupont *et al.*, 2004). Il existe de très nombreux modèles de ce type à l'étranger, les tableaux 1 et 2 donnant une liste limitée aux seuls pays de l'OCDE, hors France, empruntée à la revue récente de Li et O'Donoghue (2012). Dans le cas de la France, les principaux modèles sont le modèle Destinie de l'Insee (voir notamment Bonnet et Mahieu, 2000; Bardaji *et al.* 2003; Blanchet *et al.* 2011; Bachelet *et al.*, 2014) et le modèle Prisme portant sur la population des affiliés au régime général (Poubelle *et al.*, 2006). D'autres modèles appliqués aux retraites sont désormais en cours de développement, le modèle PENSIPP de l'IPP qui dérive du modèle Destinie (Blanchet *et al.*, à paraître), le modèle Trajectoire de la Drees (Duc *et al.*, 2013), ou le modèle Pablo du Service des Retraites de l'État. Même si la question des retraites est prédominante parmi les usages des modèles dynamiques, ils sont par ailleurs mobilisés sur l'ensemble des questions liées au vieillissement démographique telles que la prospective des dépenses de santé (Dormont *et al.*, 2006) ou de dépendance (Duée et Rebillard, 2004; Marbot et Roy, 2012).

Ces modèles dynamiques peuvent par ailleurs s'envisager sous deux formes, soit des modèles « de période » projetant la population et ses droits à différents horizons, soit des modèles de cohorte faisant vieillir parallèlement les individus d'un même groupe d'années de naissance, par exemple pour analyser la dynamique des inégalités ou de la redistribution au niveau intragénérationnel. Des exemples français de modèles de cohorte sont le modèle Gameo utilisé pour la simulation des rendements de l'éducation sur cycle de vie (Courtioux *et al.*, 2011; Courtioux, 2012), ou le modèle développé dans Allègre *et al.* (2012) ainsi que le modèle TAXIPP-LIFE en cours de développement à l'IPP qui est une extension sur cycle de vie du modèle TAXIPP.

À elle seule, la dynamisation des modèles ne répond cependant qu'à une partie de l'objectif initial d'Orcutt, à deux égards. Idéalement, il faut d'abord y rajouter une endogénéisation du comportement des agents face aux variations des politiques économiques ou sociales que simule le modèle, et aussi modéliser les effets de bouclage, c'est-à-dire les interactions entre les comportements de ces agents. Une forme courante d'endogénéisation des comportements consiste à faire varier l'offre de travail en fonction du profil des prélèvements et des transferts ou, dans le cas de la retraite, à modéliser la façon dont la modification des barèmes peut modifier les comportements de liquidation (dans le cas du modèle Destinie, voir Bachelet *et al.*, 2011). On peut se contenter de cette étape si on considère que la résultante macroéconomique de ces réactions individuelles équivaut à leur somme. Par exemple, à long terme, on considère en général qu'un décalage donné des âges de sortie du marché du travail n'a pas d'effet sur l'emploi des tranches d'âge plus jeunes. Mais il est de nombreux cas où on voudrait que la microsimulation prenne en compte des effets de bouclage. Il existe alors deux possibilités. L'une est d'analyser ces effets à un niveau plus macro, et de n'utiliser le modèle de microsimulation que pour les reventiler au niveau individuel. Par exemple, un modèle de microsimulation peut être utilisé en complément d'un modèle d'équilibre général calculable pour analyser les effets individuels de politiques d'ajustement structurel. On parle d'approche *top-down* ou *séquentielle* (voir par exemple Bourguignon *et al.*, 2008). Mais, inversement, on peut vouloir pousser jusqu'au bout le principe de l'approche

Modèle	Pays	Utilisation
ANAC	Italy	Examines the effect of demographic changes on the Italian saving rate and the reform of the pension system
APPSIM	Australia	Designed to provide answers regarding the future distributional impact of policy change and other issues associated with policy responses to population ageing
CAPP_DYN	Italy	Analyses the long term redistributive effects of social policies
CBOLT	USA	Analyses potential reforms to federal entitlement programmes and quantifies the US nation,Äôs long-term fiscal challenges
CORSIM	USA	Models changes occurring within kinship networks, wealth accumulation, patterns of intergenerational mobility, the progressivity and the life course of the current social security system, as well as potential reforms, household wealth accumulation, health status, interstate migration, time and income allocation, and international collaborations
DEMOGEN	Canada	Models distributional and financial impact of proposals to include homemakers in the Canadian pension plan
DYNACAN	Canada	Models the Canada Pension Plan and its impact on the Canadian population Dynamic Model Ireland Models inter-temporal issues relating to the degree of redistribution within the tax-benefit system
DYNAMITE	Italy	Models microeconomic issues and the impact of macroeconomic/institutional changes on the distribution of income
DYNAMOD I & II	Australia	Models life course policies such as superannuation, age, pensions and education, long-term issues within the labour market, health, aged care and housing policy, future characteristics of the population and the projected impact of policy changes
DYNASIM I & II	USA	Forecasts the population up to 2030 by employing different assumptions regarding demographic and economic scenarios, and analyses the cost of teenage childbearing to the public sector under alternative policy scenarios, also includes a link to a macro model
DYNASIM III	USA	Designed to analyse the long-term distributional consequences of retirement and ageing issues
FAMSIM	Austria	Models the demographic behaviour of young women
FEM	USA	A demographic and economic simulation model designed to predict the future costs and health status of the elderly and to explore what current trends or future shifts might imply for policy, developed by RAND
HARDING	Australia	Analysis of lifetime tax-transfer analysis, for analysis of policy concerning the Higher Education Contribution Scheme and redistributive impact of government health outlays over the lifetime of an individual HouseMod Australia Simulates the impacts of different policy options at the small area level in Australia
IFSIM	Sweden	Studies intergenerational transfers and the interdependence between demography and the economy
IFS Model	UK	Studies pensioner poverty under a variety of alternative tax and benefit policies
INAHSIM	Japan	Simulates demographic and social evolution, able to simulate kinship relationships in detail
INFORM	UK	Developed for forecasting of benefit caseloads and combinations of receipt, designed to incorporate significant benefit reforms planned over the coming years, based entirely on administrative data
Italian Cohort	Italy	Analyses lifetime income distribution issues
Japanese Cohort	Japan	Looks at the impact on household savings of changes in demographic structure
LABORsim	Italy	Simulates the evolution of the labour force over future decades in Italy
LIAM	Ireland	Evaluates potential reforms to the Irish pensions system in terms of changes to life-cycle incomes
LIFEMOD	UK	Models the lifetime impact of a welfare state

Table 1: Principaux modèles nationaux utilisés dans les pays de l'OCDE hors France (extrait de Li et O'Donoghue, 2012)

Modèle	Pays	Utilisation
LifePaths	Canada	Models health care treatments, student loans, time-use, public pensions and generational accounts
Long Term Care Model	UK	Models long term care reform options
MICROHUS	Sweden	Models dynamic effects of changes to the tax-benefit system on the income distribution and economic-demographic effects of immigration
MICSIM	Germany	Analyses German pension and tax reform
MiMESIS	Sweden	Evaluates Swedish Pension Reform
MIDAS	Multi	Analyses pension system and social security adequacy
MIDAS	New Zealand	Models wealth accumulation and distribution
MIND	Italy	Simulates the economic impact resulting from alternative values of the income growth rate and real interest rate
MINT	USA	Forecasts the distribution of income for the 1931-1960 birth cohorts in retirement, MINT5 extends to the 1926-2018 birth cohorts
MOSART 1/2/3	Norway	Models the future cost of pensions, undertakes micro level projections of population, education, labour supply and public pensions, incorporates overlapping-generations, models within a dynamic microsimulation framework
NEDYMAS	Netherlands	Models intergenerational equity and pension reform, the redistributive impact of social security schemes in a lifetime framework
PENMOD	Japan	Public pension system analysis
PENSIM	UK	Models the treatment of pensioners by the social security system across the income distribution
PENSIM2	UK	Estimates the future distribution of pensioner incomes to analyse the distributional effects of proposed changes to pension policy
PENSIM	USA	Analyses lifetime coverage and adequacy issues related to employer-sponsored pension plans in the USA.
Pensions Model	Belgium	Analyses and forecasts the medium term impact of a change to pension regulations
POHEM	Canada	A longitudinal microsimulation model of health and disease, it is used to compare competing health intervention alternatives within a framework that captures the effects of disease interactions
POLISIM	USA	Demographic-economic and social security projection for US social security administration
PRISM	USA	Evaluates public and private pensions
PSG	USA	Analyses the lifetime implications of social security policies for a large sample of people born in the same year
SADNAP	Netherlands	Evaluates the financial and economic implications of the problem of ageing
SAGE	UK	Dynamic demographic/tax model for the UK
SESIM	Sweden	Models budget and distributional impact of inter-temporal policy issues such as student grants, labour supply, savings decisions and pensions
Sfb3	Germany	Analyses pension reforms, the effect of shortening worker hours, distributional effects of education transfers
SVERIGE	Sweden	Models human eco-dynamics (the impact of human cultural and economic systems on the environment)
Swedish Cohort	Sweden	Models the replacement of social insurance by personal savings accounts and the distribution of lifetime marginal effective tax rates
Tdymm	Italy	Analyses the Italian labour market and pension system, with a focus on pension adequacy and related distributional effects

Table 2: Principaux modèles nationaux utilisés dans les pays de l'OCDE hors France (suite)

microfondée, c'est-à-dire rendre compte du bouclage global sur la base des interactions se nouant au niveau individuel. On parle alors d'approche *bottom-up* ou intégrée (voir par exemple Cogneau et Robilliard, 2007).

On tend alors à rejoindre un autre courant de modélisation, celui des modèles dits d'agents ou ACE pour *agent-based computational economics* (Colander *et al.* 2008; LeBaron et Tesfatsion, 2008). Ce courant a d'autres origines que l'article d'Orcutt : l'idée de base est de proposer des analyses des phénomènes d'interaction économiques et sociaux par la simulation de petites sociétés artificielles (Gilbert, 2008). Elle a été développée assez tôt dans des champs plus sociologiques, avec souvent une composante d'analyse spatiale, l'exemple le plus connu étant la simulation de l'émergence spontanée de phénomènes de ségrégation spatiale dans des populations hétérogènes (Schelling, 1971). Dans le domaine économique, des applications de ces modèles sont la dynamique de l'emploi et de la formation des entreprises (Ballot, 2002; Fagiolo *et al.*, 2004; Barlet *et al.*, 2009; Ballot *et al.*, 2013) ou le domaine financier, avec la possibilité pour ces modèles de rendre compte des effets déstabilisateurs de comportements moutonniers ou à rationalité imparfaite (Winker et Gilli, 2001; Franke, 2009). On trouvera un aperçu de ce courant dans l'ensemble d'études rassemblées récemment par Gaffard et Napoletano (2012). Dans le domaine démographique, on pourra aussi se référer à Billari et Fürnkrantz-Prskwetz (2003).

Il est assez normal d'assister à une certaine convergence entre ces modèles ACE et les modèles de microsimulation : les modèles de microsimulation traditionnels mettent davantage l'accent sur la représentativité et le réalisme des populations qu'ils simulent, en s'appuyant autant que possible sur des données réelles et une description détaillée des barèmes sociaux et fiscaux qui leurs sont appliqués, sans mettre en avant les questions de comportement ou de bouclage. A l'inverse, les modèles d'agents mettent l'accent sur ces deux derniers points, mais en les analysant en général sur des populations artificielles fictives, de petite taille, dans des environnements très stylisés. Il s'agit là de différences qui ont vocation à se réduire, avec davantage de prise en compte des comportements dans les modèles de microsimulation et un souci croissant de réalisme des populations simulées dans le cadre des modèles d'agents. Tout ceci doit conduire à une classe de modèles de plus en plus en ligne avec le projet initial d'Orcutt. On verra que le modèle présenté dans la section 3 reste plutôt représentatif des modèles de microsimulation dynamique traditionnels, et plus spécifiquement centré sur l'aspect démographique, mais les éléments de modélisation présentés dans la section 2, même s'ils sont très en deça de l'approche ACE, sont en partie empruntés au cadre ACE de Barlet *et al.* (2009).

Pour compléter cette vue d'ensemble sur le positionnement des techniques de microsimulation, on soulignera les interactions possibles avec un autre domaine, celui de l'évaluation microéconométrique des politiques publiques, qu'elle se fasse à partir d'expériences naturelles ou controlées (Givord, 2010; L'Horty et Petit, 2010). La plupart des modèles de microsimulation partagent ce même objectif d'évaluation des politiques publiques qu'il s'agisse de fiscalité, de retraites ou de politiques d'emploi.... C'est tout aussi vrai dans d'autres champs que l'économie. Un exemple est celui des applications épidémiologiques consistant à utiliser la microsimulation pour modéliser l'enchainement des différentes étapes de développement d'une pathologie donnée et à en déduire l'efficacité potentielle de politiques de dépistage et de traitement selon l'étape du processus où elles interviennent (Rutter *et al.*, 2011; Zuchelli *et al.*, 2012). La microsimulation projette les effets de ces politiques sur la base d'hypothèses ou de connaissances préalables relatives aux différents canaux par lesquels elles peuvent jouer. À l'inverse, l'évaluation par expériences naturelles ou contrôlées relève d'une approche en forme réduite ne détaillant pas les canaux par lesquels jouent les politiques. Leurs effets finaux sont inférés au seul vu de l'observation, par comparaison entre des groupes de bénéficaires et de

non bénéficiaires de ces politiques. Ces deux démarches sont complémentaires. Le point fort de l'évaluation par expériences naturelles est de s'appuyer sur la stricte observation des faits, mais on n'a jamais la garantie totale d'isoler l'effet pur de la politique en question, malgré la sophistication des moyens mis en œuvre. Les expériences contrôlées se rapprochent davantage de la mesure d'un effet pur, mais avec la difficulté à transposer au monde réel des résultats obtenus dans un contexte expérimental particulier. Les deux méthodes rencontrent également des limites en présence d'effets de bouclage, c'est-à-dire lorsque ce qui est fait au groupe test a aussi un impact sur le groupe témoin. Enfin, l'observation des faits ne permet pas d'anticiper les effets de politiques alternatives n'ayant jamais été mises en œuvre ou expérimentées, sauf à raisonner par extrapolations ou analogies. Face à ces limites, l'évaluation par microsimulation permet de décomposer les canaux par lesquels passe l'effet final des politiques, elle peut intégrer des effets de bouclage et elle permet la simulation *ex ante* des effets détaillés de politiques nouvelles. Mais la construction de ces modèles a besoin d'une base empirique et une démarche possible serait donc d'essayer de caler leur comportement sur un corpus d'évaluations *ex post* ou d'expérimentations existantes, si elles sont suffisamment convergentes.

1.2 Micro versus macrosimulation : avantages et inconvénients

Au vu de ce premier tour d'horizon, la microsimulation ou les modèles d'agents présentent de nombreux avantages par rapport aux approches de type plus agrégé. Ces approches agrégées ne se justifieraient que lorsqu'on ne s'intéresse qu'à des résultats globaux et lorsque les conditions sont remplies pour qu'on puisse raisonner en termes d'agent représentatif. Or il y a une demande croissante pour que les phénomènes socio-économiques soient abordés en termes de variance autant que de moyenne : cette demande vaut aussi bien pour les travaux de modélisation que pour les simples travaux d'observation. Et quand bien même on voudrait se limiter à la modélisation des moyennes, on sait que les conditions permettant d'en rendre compte par la fiction de l'agent représentatif ne sont jamais remplies dans la pratique (Kirman, 1992). C'est d'ailleurs de cette critique de l'hypothèse d'agent représentatif qu'était parti Orcutt.

Conclure au rejet des approches agrégées est toutefois trop rapide. Il y a beaucoup de cas où la microsimulation est un outil d'une lourdeur disproportionnée dont l'utilisation présenterait plus d'inconvénients que d'avantages. Il faut savoir apprécier le poids relatif des uns et des autres. L'ensemble de ce qui sera présenté dans ce document aidera à s'en faire progressivement une idée, mais on peut en donner dès à présent un aperçu, en prenant l'exemple des applications de type démographique. La question est de savoir dans quel contexte il est intéressant de basculer de l'approche traditionnelle à l'approche par microsimulation, ce que sont les inconvénients d'un tel basculement, et comment on peut y remédier. Ceci introduira notamment la thématique du calage et de l'alignement qui jouera un rôle central dans les applications des sections 2 et 3.

Dans le domaine démographique, les problèmes de modélisation sont en général des problèmes dynamiques de projection de la structure d'une population. Le cas élémentaire est la projection de l'effectif par âge d'une population sans flux migratoires. L'état de cette population est décrit à une date courante par un vecteur $P_t(a)$ donnant les effectifs des différentes tranches d'âge. Le passage de ce vecteur au vecteur décrivant la population à la date suivante se fait par des opérations du type

$$P_{t+1}(a+1) = [1 - q(a)].P_t(a)$$

où les $q(a)$ seront des probabilités de décès par âge et, en faisant abstraction de la mortalité infantile :

$$P_{t+1}(0) = \sum_a f(a).P_t(a)$$

où les $f(a)$ sont des taux de fécondité par âge des parents. Ces équations peuvent être écrites de manière plus compactes sous forme matricielle, en disant que le vecteur P_{t+1} est égal au produit AP_t où A est la matrice de passage dite « de Leslie » dont les éléments non nuls seront une sous-diagonale de termes $1 - q(a)$ et une première ligne contenant les taux de fécondité par âge $f(a)$. Cette méthode est qualifiée de méthode « des composants ».

Ce type d'approche se généralise au cas où on décrit la population par un plus grand nombre de caractères, à la condition que ces caractères ne prennent qu'un nombre limité de modalités. Par exemple, une projection selon l'âge et la région de résidence dans laquelle, outre la fécondité et les décès, il faudra écrire les probabilités de migrations entre chaque couple de régions considérées. Ou encore, en vue de la simulation des droits à retraite, une projection jointe de la population selon son âge et son nombre d'années travaillées, puisque celui-ci impacte les droits à la retraite. Ce cadre de modélisation peut être qualifié d'approche matricielle ou encore par cellules (*cell-based*) consistant à avoir décrit la population par sa répartition dans un nombre fini de cellules et à modéliser les déplacements de groupes d'individus entre ces cellules au cours du temps.

Cette méthode est simple à mettre en œuvre. On dispose par ailleurs de résultats analytiques qui nous indiquent vers quel genre de dynamique à long terme est supposé évoluer ce type de système. Qu'est-ce qui peut justifier d'y substituer la microsimulation et, ce faisant, quels sont les autres problèmes auxquels on se trouve confronté ?

Les raisons poussant à préférer la microsimulation sont de deux ordres. D'une part, la méthode matricielle conduit rapidement à devoir gérer des tableaux de grande taille. En second lieu, les programmes écrits de cette façon sont difficiles à adapter pour l'inclusion de critères supplémentaires de classement de la population. Si on démarre par une ventilation simple par âge, on stocke en général l'information sous forme de vecteurs indicés par l'âge et la projection consistera à écrire une ou plusieurs boucles selon l'âge. Si on rajoute une dimension supplémentaire, on le fera en général en ajoutant un indice de plus en passant d'une description vectorielle à une description de la population qui sera elle-même matricielle. On change ainsi le système d'indiçage de données : il faut rajouter au moins une boucle au programme de simulation et modifier l'ensemble des instructions où apparaîssent des quantités indicées par les critères de classement. Tout ceci n'est pas dirimant tant qu'on en reste à une catégorisation de la population selon un nombre limité de critères. En revanche, ces limites peuvent devenir insurmontables lorsque la dimensionalité du problème s'accroît. Un exemple est celui de la simulation des structures familiales : les configurations familiales dans lesquels peut se trouver inséré un individu sont très variées. On pourrait certes se contenter d'un classement selon la taille du ménage et la position dans le ménage, mais la probabilité de passer d'une valeur à une autre de ces paramètres dépend en général des caractéristiques détaillées des autres membres du ménage : la probabilité de se retrouver veuf ou veuve dépend ainsi de l'âge du conjoint. Il faut donc gérer une double classification selon l'âge individuel et l'âge du conjoint, se rajoutant aux autres critères classants qu'on a envie de projeter. Un autre exemple, pour la projection des droits à retraite, est la prise en compte des caractéristiques détaillées de la carrière. Comme on l'a indiqué, un double classement des individus selon leur âge et la durée déjà cotisée à cet âge est gérable à faible coût. Le problème reste gérable lorsqu'on doit rajouter une différenciation selon le type

de régime dans lequel ont eu lieu ces différentes années de cotisation : c'est par exemple ce que fait le modèle de retraite Promess de la Drees (Aubert *et al.*, 2010). Mais la démarche devient inapplicable lorsqu'on veut faire intervenir d'autres critères de différentiation, par exemple la diversité des trajectoires salariales, or le mode de calcul des retraites dans le système français fait jouer un rôle significatif à ces profils de salaires.

Pour répondre à ces besoins, la microsimulation dynamique va consister à définir autant de variables individuelles qu'il y a de critères dont la simulation doit rendre compte, variables qui pourront être discrètes ou continues, puis à programmer l'évolution de ces variables, d'une année sur l'autre, individu par individu. Ceci pourra se faire de trois manières :

- Soit sur la base de règles déterministes simples : l'exemple de base est l'incrémentation annuelle de l'âge ou de l'ancienneté dans tel ou tel des états courants.

- Soit sur la base de tirages pseudo-aléatoires : si on connait la probabilité instantanée de passer d'un état à un autre pour une unité individuelle, on simule une variable logique valant vrai ou faux tirée selon cette probabilité et on opère le changement d'état à chaque fois qu'est tiré un résultat positif; dans le cas d'une variable continue, telle que le salaire, on peut tirer son évolution d'une période sur l'autre dans une loi combinant une composante déterministe, telle qu'un effet moyen de l'expérience, et une partie aléatoire représentant les chocs positifs ou négatifs qui affectent le salaire.

- Soit sur la base de modèles de comportement plus élaborés : par exemple, dans Destinie, on dispose d'un module de simulation du comportement de départ en retraite en fonction des caractéristiques de l'individu et des niveaux de pension auxquels il a droit aux différents âges possibles de liquidation.

On obtient ainsi un style de programmation assez intuitif et flexible. On simule les évènements tels qu'ils arrivent aux individus (*model what you see*). La taille des objets gérés par le programme ne croît que linéairement avec le nombre de critères pris en compte, au lieu de la croissance exponentielle à laquelle on avait affaire avec l'approche matricielle. Enfin, enrichir le modèle par la prise en compte d'un critère non prévu au départ peut, au moins sur le papier, se faire assez simplement. Il suffit de rajouter les lignes d'instructions gérant l'évolution de cette variable additionnelle, en conditionnant ces évolutions de la façon que l'on souhaite en fonction des autres variables déjà simulées. Une fois que c'est fait, on bénéficie, à n'importe quel horizon, de tous les types de croisements possibles entre cette nouvelle variable et les autres variables déjà gérées par le modèle. Par exemple, dans un modèle initialement appliqué aux seules retraites, il est possible de rajouter un module simulant l'entrée dans la dépendance, conditionnée par des variables telles que le sexe, l'âge et la catégorie sociale, puis de bénéficier du croisement entre cette nouvelle variable et le calcul des droits à retraite pour simuler l'impact de prestations dépendance conditionnées par le niveau de revenu. Dans la pratique, ce type d'opération sera évidemment un peu plus lourd que cela, car il faudra prendre en compte des phénomènes de rétroaction sur la projection de départ : par exemple simuler la dépendance implique de postuler une mortalité différentielle entre personnes dépendantes et non-dépendantes et la combinaison de ces deux mortalités ne redonnera pas nécessairement la mortalité générale simulée dans le modèle de départ. De plus la programmation d'une prestation sous conditions de ressources peut s'avérer assez complexe et révéler des fragilités du modèle de départ quant à la projection des inégalités de revenu. Mais ce type d'ajout est de toute manière quasiment impossible à traiter par des approches agrégées et rend donc la microsimulation difficilement contournable.

Ces avantages sont toutefois contrebalancés par un inconvénient majeur, qui est le caractère stochastique du résultat obtenu : il n'est qu'une réalisation particulière du processus aléatoire gouvernant l'évolution de la population considérée, sur un échantillon de taille réduite. Il y a donc un arbitrage à faire entre l'avantage de pouvoir simuler des phénomènes complexes et l'inconvénient de ne pouvoir le faire qu'avec un résultat aléatoire. Si l'objectif est la projection démographique d'une structure simple telle que la structure par sexe et âge sur la base d'hypothèses de mortalité et de fécondité qui ne dépendent que de l'âge, il n'y a pas de débat sur la bonne façon de faire : la projection matricielle est préférable, et la reproduire par microsimulation ne fera que redonner le même résultat sous forme bruitée. Dans le cas extrême inverse de projections très complexes, la microsimulation est incontournable. Dans tous les cas intermédiaires, le choix est à apprécier au cas par cas.

Ceci étant, cet arbitrage peut tenir compte de ce qu'il existe des moyens de réduire le caractère bruité de la microsimulation. Deux premières solutions à peu près équivalentes sont l'accroissement de la taille des échantillons simulés, ou la réalisation de simulations multiples dont on présentera les résultats moyens. Elles sont évidemment coûteuses en temps de calcul : ce ne sont donc pas toujours des options. Mais on peut aussi chercher à contrôler la variabilité des résultats par d'autres voies. C'est l'objectif des techniques dites de réduction de variance et d'alignement, qui seront présentées plus en détail à la section 2.5.

1.3 Différents modes d'organisation

Au-delà de leur principe général commun, les modèles de microsimulation dynamique peuvent se différencier selon deux critères : la distinction entre modèles de période et de cohorte, déjà évoquée plus haut, et la distinction entre modèles « ouverts » et « fermés ». Les exemples présentés dans ce document relèveront de la catégorie des modèles de période de type « fermés », mais il est utile de situer cette façon de procéder par rapport aux deux autres options.

1.3.1 Modèles de période et de cohorte

La forme typique du modèle de période est de se présenter sous forme de deux boucles emboitées. Cela correspond au schéma récursif proposé par Orcutt. A chaque période, une boucle (ou éventuellement une succession de boucles) balaye l'ensemble des individus et modifie leur état en fonction des paramètres du modèle et, éventuellement, de l'état dans lequel se trouvent les autres individus de la population. Cette boucle est ensuite insérée dans une boucle temporelle allant de la date initiale à l'horizon de la simulation. Avec cette structure, on peut dire qu'un modèle de période fournit pour chaque date de la projection un équivalent actualisé du fichier initial. Il peut servir à fournir de nouvelles photographies de la population à toutes ces dates, avec la même richesse d'information que celle dont on dispose ou qu'on a reconstitué à la date initiale. Il s'agit là d'un autre élément de différentiation avec les méthodes de projection de type plus macro. Un modèle macro prédéfinit le type d'agrégats qu'il va projeter : une fois ce choix arrêté, la liste de variables dérivées qui pourront être reconstituées à partir de ces agrégats sera forcément limitée, alors qu'un modèle de microsimulation laisse la porte ouverte à des exploitations bien plus variées de ses résultats. En fait, la microsimulation peut se voir comme un outil de projection de l'équivalent d'un fichier d'enquête, qui peut être exploité à toutes les dates projetées comme le serait une enquête collectée en population réelle, les seules limites étant celles de la représentativité statistique.

En quoi les modèles de cohorte se différencient-ils de cette première approche ? Leur objectif est plutôt l'analyse des trajectoires et des inégalités ou de la redistribution au niveau intragénérationnel. Les modèles de période peuvent évidemment être exploités dans cet esprit puisque les résultats donnent à la fois les états des individus à chaque date, mais aussi leurs trajectoires biographiques. Mais cette approche par période ne se prête pas bien à la reconstitution de cycles de vie complets, sauf à réaliser la projection à horizon très lointain en se centrant sur les générations entrant dans la population en début de projection. Quand l'objectif final est plutôt l'analyse de trajectoires complètes sur cycle de vie, par exemple pour des analyses des effets redistributifs du système socio-fiscal sur l'ensemble de l'existence, on peut préférer travailler isolément sur une ou quelques cohortes. Au couple de boucles emboitées par période et individu on substitue un couple de boucles par âge et individu. Dans le cas particulier ou, par surcroît, les individus sont simulés indépendamment les uns des autres, rien n'interdit d'inverser l'ordre qui s'imposait pour les modèles de période, i.e. avoir la boucle individuelle comme boucle principale, puis, pour chaque individu, avoir une boucle sur l'âge qui permettra de simuler l'intégralité de son cycle de vie. Un cas particulier de modèles de cohorte sont les modèle de cohorte fictive consistant à simuler les trajectoires d'un ensemble d'individus qu'on fait naître la même année, et dont on projette les trajectoires avec le même type de lois de transitions que dans une simulation par période, mais avec des paramètres qui ne dépendront que de l'âge et non pas de la date, ceci valant en général aussi pour le paramétrage du système socio-fiscal. Ce genre de simulation évalue ce que seraient les situations des individus aux différents stades de leur cycle de vie dans un régime permanent fictif reproduisant indéfiniment les conditions de l'année initiale. Mais un modèle de cohorte peut tout aussi bien reconstituer des trajectoires de cohortes réelles soumises à un environnement économique et des législations qui évoluent au fur et à mesure qu'elles avancent en âge. Dans ce cas, leur construction est à peu près aussi coûteuse que celle de modèles de période.

1.3.2 Modèles ouverts et modèles fermés

L'autre grande dichotomie est celle qui oppose les modèles dits « ouverts » ou « fermés ». Elle intervient lorsque les individus simulés sont amenés à présenter des liens avec d'autres individus. Prenons le cas d'une approche par cohorte visant à simuler les effets sur cycle de vie d'une législation socio-fiscale donnée. Pour simuler les droits d'un individu à chaque phase de son cycle de vie, il faut simuler l'évolution de son environnement familial. Ceci sera fait en général du point de vue de l'individu. On lui simulera un conjoint ou des conjoints successifs ainsi que des enfants mais les individus additionnels ainsi générés ont un statut à part au sein de la simulation. Pour ces individus, on se bornera à simuler les évènements utiles à la simulation des individus de référence : par exemple, le devenir des enfants des individus de référence n'a pas besoin d'être simulé au-delà de leur prise d'autonomie. On parle de modèle ouvert pour caractériser cette approche qui combine la simulation d'une population d'individus de référence, qui est l'objet de la simulation, et une population d'individus périphériques servant uniquement à décrire l'environnement des individus de référence.

Le même principe peut être appliqué à une simulation par période, avec gestion de deux populations séparées. Par exemple, si le fichier de départ est un ensemble de ménages, un individu quittant un ménage pour entrer en union donne lieu à la création d'informations nouvelles sur un conjoint qu'on introduit dans la base à l'occasion de cette union. Mais traiter cet individu additionnel comme individu à part entière de la population viendrait fausser la structure démographique de l'échantillon. Une approche plus satisfaisante consiste alors à travailler de manière « fermée » : lorsqu'on simule une entrée en union pour un individu

de l'échantillon, on va chercher le conjoint au sein de la population simulée, et c'est uniquement par les naissances issues de ces unions que l'échantillon sera alimenté à sa base. On simule donc une sorte d'isolat représentatif de la population d'intérêt. Le modèle Destinie fonctionne de cette façon : plus exactement, il gère un fichier de données totalement individualisées mais qui comprend pour chaque individu les identifiants des autres individus de l'échantillon auxquels il est lié, et ces liens sont créés ou mis à jour au fur et à mesure de la projection. De cette manière, tout ce qui est simulé individuellement pour chaque individu de l'échantillon se répercute en tant que de besoin vers les individus qui lui sont liés : un décès peut déclencher le versement d'une pension de réversion, le passage d'un seuil d'âge pour un enfant conduit à l'arrêt de telle ou telle prestation perçue par ses parents... On précise que, dans cette logique, l'adjectif « fermé » ne s'entend pas au sens de l'absence de flux migratoires. Des individus additionnels peuvent entrer dans la population simulée autrement que par la naissance et une fois entrés, y figurer exactement au même titre que les individus présents en début de projection ou entrés par la naissance. On verra à la section 3 comment peuvent être gérées ces entrées/sorties du territoire.

1.4 Initialisation des données

En règle assez générale, tout modèle comprend à la fois le modèle lui-même et la base de données sur lequel il s'appuie. Par exemple, dans le cas d'un modèle macroéconométrique, un travail préliminaire important est la constitution de la base de séries macroéconomiques qui serviront à l'estimation des équations du modèle et que le modèle a ensuite vocation à projeter. Dans un modèle de microsimulation dynamique, l'équivalent est la constitution du fichier de données individuelles initiales que le modèle va ensuite faire vieillir et renouveler. Cette étape peut être assez lourde si elle inclut elle-même une phase importante d'imputation et/ou d'appariement de données. Les problèmes rencontrés sont communs aux modèles statiques et dynamiques.

Un premier cas est celui d'un modèle monosource s'appuyant sur une enquête ou une source administrative unique contenant l'ensemble des données individuelles requises. Par exemple, le modèle de fiscalité statique INES s'appuie sur l'enquête *Revenus fiscaux et sociaux* (ERFS) de l'Insee, les modèles dynamiques de retraite Prisme et Trajectoire s'appuient respectivement sur les fichiers de cotisants et d'allocataires de la CNAV et sur les échantillons interrégimes de cotisants et de retraités (EIC et EIR) de la Drees. Dans ce cas, une partie du travail de préparation des données a été en principe effectué en amont, lors de la mise en forme du fichier d'enquête ou de la conversion du fichier de données administratives de gestion en un fichier utilisable statistiquement, mais des ajustements supplémentaires peuvent être requis. Ainsi, le dispositif EIR/EIC ne couvre que certaines générations, or un modèle de microsimulation dynamique vise à projeter un échantillon représentatif de l'ensemble de la population retraitée, ce qui suppose de reconstituer des données pour les générations intermédiaires. Dans le cas du modèle Prisme, les données de gestion de la CNAV doivent être complétées pour les parties de carrière passées ne relevant pas du régime général ou par des données socio-démographiques telles que les dates de naissance des enfants ou l'âge de fin d'études, cette opération de complétion s'appuyant sur l'EIC et l'enquête *Famille*.

Le modèle Destinie est également un modèle s'appuyant sur une source unique, l'enquête *Patrimoine*, mais avec une part de données reconstituées beaucoup plus importante. En premier lieu, l'enquête *Patrimoine* ne donne qu'une partie des informations sur les carrières passées qui sont nécessaires au calcul des droits à retraite. Son apport principal est de fournir des historiques complets des situations sur le marché du travail,

à pas annuel. En revanche, en matière de salaires, seul le salaire courant est renseigné dans l'enquête. La séquence des salaires passés doit donc être reconstituée. Elle pourrait l'être par appariement avec d'autres sources, et notamment l'EIC mais, dans la version actuelle du modèle on procède plutôt par imputation, à l'aide des mêmes équations de salaires individuels que celles qui sont utilisées en projection, qui font dépendre ce salaire de l'âge, de l'expérience, du genre et du niveau d'éducation, ainsi que du type d'emploi. Evidemment, l'application rétrospective de ces équations de salaire donne des salaires passés qu'il faut ensuite recaler en niveau de sorte à retrouver, en moyenne, les niveaux de salaires moyens historiques de la comptabilité nationale.

La phase d'imputations initiales dans Destinie ne s'arrête pas à cela. Il s'y ajoute d'abord une réimputation initiale des montants de retraite. Ceci peut sembler paradoxal car cette donnée est en principe disponible dans l'enquête *Patrimoine*. Ce choix a deux raisons. D'une part, les pensions dans l'enquête *Patrimoine* ne sont connues qu'au niveau de l'ensemble du ménage, il serait donc de toute manière nécessaire d'imputer leur partage entre les deux conjoints. L'autre raison est que, quand bien même on disposerait des données, il ne s'agirait que de données déclaratives d'où un risque d'incohérence flux/stock au démarrage de la projection : partir de données déclarées en stock et enchaîner sur des pensions reconstituées d'après barême pour les nouveaux liquidants conduit à des évolutions transitoires biaisées dans le sens de la baisse ou de la hausse selon que les pensions déclarées sur-estiment ou sous-estiment les pensions vraies découlant des barèmes. Reconstituer les pensions initiales évite ce problème en garantissant l'homogénéité entre ce qui est décrit en stock et ce qui sera projeté en flux. Ceci veut dire que le travail de préparation de la base inclut aussi un travail de rassemblement des règles de calcul passées des droits à retraite, pas seulement de celles qui sont supposées s'appliquer en projection. Ce supplément de travail initial a une contrepartie intéressante : dès lors que les pensions du stock sont reconstituées, elles peuvent l'être de manière contrefactuelle avec d'autres barèmes que ceux qui ont été réellement appliqués, ce qui permet de reconstituer facilement, par différence l'effet des réformes passées.

Toujours dans Destinie, on ajuste également les données individuelles de sorte à rendre la structure par âge initiale aussi proche que possible de la structure par âge réelle de la population française. Enfin, une dernière opération est celle qu'on peut qualifier de « fermeture » du fichier de l'enquête *Patrimoine*. Comme expliqué à l'instant, Destinie utilise l'approche « fermée » dans laquelle les unions se forment entre individus de la population simulée, cette population se renouvelant par les naissances issues de ces unions. En projection, ceci permet de gérer totalement la dynamique des liens familiaux, que les individus apparentés résident ou non dans le même ménage. Soit par exemple un individu jeune observé comme vivant chez ses parents à la date de l'enquête. En projection, ce lien avec ces parents sera conservé même si l'individu part constituer un nouveau ménage. À long terme, le modèle permet donc de dire quelle proportion de la population a l'un ou l'autre de ses parents en vie et dans quelle configuration de ménages se trouvent ces parents. Symétriquement, pour une personne âgée, le modèle projette à chaque date la liste de ses enfants et ce que sont leurs statuts, matrimonial ou sur le marché du travail. Mais le problème qui reste à résoudre est que la structure du fichier initial n'est pas de ce type : le fichier de l'enquête *Patrimoine* est un fichier ménages, on sait pour chaque individu si ses parents sont encore en vie mais, s'il ne réside plus chez eux, il n'y a aucune raison que ces parents appartiennent eux-mêmes au fichier de l'enquête, contrairement à ce que le modèle va gérer en projection. L'opération dite de fermeture consiste donc, pour l'individu considéré, à aller chercher dans le fichier de l'enquête une ou des personnes âgées présentant les caractéristiques adéquates pour jouer le

role de parents de cet individu et à créer les liens fictifs correspondants. Ces opérations d'appariement internes au fichier de départ sont faites de manière progressive, avec affaiblissement progressif des critères d'adéquation de l'appariement, jusqu'à ce qu'on ait été capable de reconstituer approximativement l'ensemble des appariements requis.

Au total, même dans un modèle s'appuyant sur une base de données réelles, la part d'informations reconstituées peut être assez importante. Mais la part de la reconstitution peut être encore plus grande, avec des modèles se passant de tout appui sur des données individuelles réelles. On parle alors de population fictive. Le recours à des populations fictives est plutôt la norme dans le cadre des modèles d'agents lorsqu'ils visent à rendre compte de faits stylisés sur des populations qui peuvent être elles-mêmes stylisées. Lorsque les modèles de microsimulation recourent eux aussi à ce principe de la population fictive, ils s'efforcent de donner à ces populations le maximum de réalisme par le biais de calages initiaux sur les moyennes ou les distributions réelles des variables d'intérêt. Ceci peut être le cas aussi bien pour des modèles statiques que pour des modèles dynamiques. On peut en donner deux exemples.

- Dans la catégorie des modèles statiques, le modèle TAXIPP de l'Institut des Politiques Publiques (Bozio *et al.*, 2012) traite une population d'environ 800 000 individus "fictifs" ou "virtuels" mais dont les caractéristiques détaillées, décrites par une quarantaine de variables, sont tirées au hasard de manière à recouper au mieux les données réelles de plusieurs sources statistiques complémentaires : l'enquête *Emploi*, l'enquête *Budget des familles*, l'enquête *Revenus fiscaux et sociaux* ainsi que les déclarations fiscales de revenu et de fortune.

- Dans la catégorie des modèles dynamiques, le modèle de Barlet *et al.* (2009) simule l'impact de modifications des politiques d'emploi sur une population fictive présentant des caractéristiques proches de celles de la population française, mais correspondant à un régime d'équilibre permanent du marché du travail, hors perturbations conjoncturelles. L'objectif du modèle est en effet d'analyser les transitions entre régimes permanents correspondant à différentes configurations des politiques d'emploi, et non pas de projeter l'état du marché du travail à partir d'un état observé à une date t. Une telle configuration d'équilibre ne peut être tirée des données existantes telles que celles de l'enquête *Emploi* et la population est donc reconstituée et calée sur les caractéristiques d'un marché du travail en conjoncture "moyenne".

La génération de ce type de populations artificielles peut reposer sur différents types d'algorithmes dont l'examen dépasse le cadre de cette revue. D'autres exemples pourront être trouvés chez Gargiulo *et al.*, 2008, Harland *et al.*, 2012, Lenormand et Deffuant, 2012.

Toujours au chapitre de l'initialisation des données figure la question du choix de la taille de la population simulée. Ce choix dépend à la fois de la taille de la ou des bases de données mobilisées et du type d'utilisation envisagé. INES utilise les 90 000 individus de l'enquête *Revenus fiscaux et sociaux*. Le modèle Myriade de la CNAV gère environ 300 000 individus. La taille encore plus élevée de l'échantillon de TAXIPP s'explique par le souhait d'offrir des focus sur des tranches très étroites au sein de la population à très hauts revenus et, s'agissant d'une population fictive, aucune contrainte n'existe quant à cette taille autre que de capacité informatique et de vitesse de traitement mais ce dernier aspect n'est que secondaire pour un modèle statique. L'échantillon de Prisme est encore plus grand, bien qu'il repose sur des données réelles : il est obtenu par tirage au vingtième dans les bases de gestion de la CNAV, soit 4 millions d'individus. Cette très grande taille s'explique par le fait que le modèle est mobilisé pour des chiffrages détaillés de mesures portant sur

des petites sous-populations, y compris dans une perspective de projections financières à court-terme pour lesquelles l'attente de précision est forte.

Dans le cas de Destinie, l'objectif est plutôt la simulation des effets tendanciels à long terme de réformes d'assez grande ampleur, ce qui permet de se contenter d'une taille d'échantillon beaucoup plus réduite. Cette taille n'est pas forcément celle du fichier de l'enquête : on utilise alternativement des fichiers de taille soit plus petite soit plus grande. Un petit fichier au dix-millième de la population française soit seulement six à sept mille individus sert à des travaux de mise au point ou pour des projections exploratoires : ce sera la taille du fichier utilisé dans l'exemple de la troisième partie. Pour des exercices demandant davantage de précision, on utilise des fichiers de taille plus grande. L'échantillon de référence du modèle est un échantillon au millième de 60 000 individus et il en va de même pour PENSIPP qui, à ce stade, s'appuie sur les projections de carrières et de trajectoires démographiques fournies par Destinie. Augmenter la taille de l'échantillon peut se faire par clonage d'individus présents dans l'échantillon de base. Le clonage des individus de la base initiale ne réduit pas la variabilité des résultats initiaux mais, en projection, deux clones suivent des trajectoires différentes ce qui réduit donc la variabilité des résultats en projection. Le clonage ou au contraire la sélection d'une partie des individus servent aussi à prendre en compte la variabilité des poids de l'échantillon initial. Dans un modèle de microsimulation dynamique, il est en effet difficile de gérer des individus à pondérations inégales, surtout dans le cadre de l'approche fermée choisie par Destinie, car on voit mal comment gérer des liens interindividuels lorsque les individus ne pèsent pas tous du même poids : par exemple, on ne sait pas quel poids il faudrait donner aux couples composés de deux personnes à pondérations différentes et par conséquent aux enfants issus de ces couples[1]. Revenir à un échantillon équipondéré se fait en clonant autant de fois que nécessaire les individus sous-pondérés dans la base de départ ou au contraire en ne retenant qu'une partie des individus surpondérés.

En résumé, la construction de la base de données initiales est une étape essentielle de la construction d'un modèle de microsimulation, avec une grande variété de cas de figure, allant du cas où un fichier de données unique est déjà disponible avec la quasi-totalité des données individuelles que le modèle souhaite projeter, au cas extrême inverse où les données individuelles doivent être totalement reconstituées, en cohérence plus ou moins poussée avec des données agrégées. Entre ces deux cas extrêmes se trouve toute la gamme des cas intermédiaires où les imputations ne portent que sur une partie seulement des données. Les exemples introductifs de la section 2 concerneront la simulation d'une population artificielle, alors que la réécriture en R du module démographique du modèle Destinie, en section 3, sera illustratif de l'appui sur des données microdémographiques réelles, celles de l'enquête *Patrimoine*.

[1] La même contrainte explique la difficulté à construire des modèles de microsimulation dynamique appliqués à des populations d'entreprises puisque la grande hétérogénéité des tailles des entreprises impose en général des plans de sondage dans lesquels grandes et petites entreprises sont très inégalement pondérées. Ces pondérations inégales peuvent être conservées telles quelles dans des modèles de microsimulation statiques et il est courant de procéder à des chiffrages d'impact de réformes de la fiscalité des entreprises, sur ce type de fichiers, sans même que de tels exercices soient présentés comme des exercices de microsimulation. Il est plus complexe à gérer dans un cadre dynamique où les entreprises doivent pouvoir, en cours de simulation, changer de classe de taille, *a fortiori* lorsqu'on veut simuler la dynamique des appariements entre entreprises et employés, puisque ceci supposerait que les employés aient eux aussi des pondérations variables et évolutives en fonction des entreprises auxquelles ils appartiennent à chaque période de la simulation. On se reportera à Ballot (2002) et Ballot *et al.* (2013) pour des modèles gérant ces contraintes. Dans Barlet *et al.* (2009) elles avaient conduit à simuler des appariements entre individus et postes plutôt qu'entre individus et entreprises, mais avec pour conséquence de ne pas permettre de simuler des politiques différenciées par taille ou type d'entreprises.

1.5 Le calibrage

Une fois récupérées ou reconstituées les données microéconomiques et/ou microdémographiques initiales, une autre problématique est celle du du calibrage du modèle. Par calibrage, on entendra ici l'estimation de ses paramètres non observés. Il est à différencier du calage dont on verra de nombreux exemples dans la suite et notamment dans la troisième partie. L'estimation ou le calibrage servent à donner des valeurs de référence aux paramètres des différentes équations du modèle. C'est une étape indispensable de sa construction. Le calage est une opération optionnelle qui n'intervient que dans un second temps, lorsqu'on veut forcer le modèle à passer par certains points d'une trajectoire prédéfinie.

La problématique du calibrage rejoint celle de la validation du modèle. Le calibrage, lorsqu'il est nécessaire, se fait par confrontation entre les résultats du modèle et l'observation empirique : parvenir à un calibrage satisfaisant du modèle est donc une forme de validation. La problématique du calibrage peut aussi recouper celle de l'initialisation des variables, lorsqu'il faut attribuer des valeurs initiales à des variables non directement observables, par exemple des paramètres décrivant les préférences individuelles dans un modèle à comportements endogènes.

Néanmoins, tous les modèles n'accordent pas la même attention à cette problématique du calibrage. La raison est que les besoins varient d'un type de modèle à l'autre.

Tout d'abord, cette question du calibrage est évidemment sans objet pour des modèles de microsimulation qui sont à la fois statiques et comptables. Les seuls paramètres dont ils ont besoin sont ceux des barèmes fiscaux et sociaux dont ils doivent simuler les effets. Dans ces modèles, les sources d'écart à la réalité sont le défaut de représentativité des sources sur lesquelles ils s'appuient ou alors une description insuffisamment précise de ces barèmes qu'ils sont censés représenter. La correction de ces écarts ne relève pas vraiment d'une problématique de calibrage.

La situation est à peine différente pour des modèles dynamiques de type démographique ou quasi-démographique. Dans un tel cas de figure, l'essentiel du paramétrage consiste à définir les probabilités de survenue des différentes catégories d'évènements simulées par le modèle. Dans le cas le plus élémentaire, ces probabilités correspondent à des statistiques usuelles, telles que les données des tables de mortalité, et le paramétrage du modèle consiste à faire des hypothèses sur l'évolution à venir de ces paramètres. De telles hypothèses ne sont pas validables ou calibrables *ex ante*. Tout au plus peut-on porter des jugements sur leur plausibilité, mais ce n'est qu'*ex post* qu'on saura si elles étaient pertinentes pour rendre compte des évolutions démographiques à venir.

Des problématiques d'estimation ne commencent à apparaître de façon plus marquée que lorsqu'on complexifie le modèle pour y faire jouer des déterminismes démographiques plus sophistiqués que la seule dépendance par rapport au sexe ou à l'âge. L'intérêt de l'approche par microsimulation est, de fait, d'autoriser ces dépendances plus complexes, par exemple des mortalités différentielles par CS, niveau de qualification ou état de santé -si le modèle simule ces variables- ou des probabilités de naissances dépendant non seulement de l'âge de la mère, mais aussi du nombre d'enfants déjà nés, du statut d'activité, ou à nouveau de la CS ou de la qualification. De même, les transitions sur le marché du travail seront déterminées par des probabilités de passage d'un statut à un autre conditionné également par l'âge, l'ancienneté dans le statut de départ, et toujours la CS ou la qualification. Rassembler ces éléments représente un travail important pour la qualité du modèle, mais il ne s'agit toujours pas d'une problématique de calibrage du modèle proprement dit car ces paramètres peuvent être évalués hors modèle par des analyses statistiques *ad hoc* sur des données d'enquête

ou des sources administratives. Par exemple, dans le cas du modèle Destinie, les probabilités de survenue des différents types d'évènements démographiques ont fait l'objet d'un travail d'estimation préalable sur la base de l'échantillon démographique permanent de l'Insee (Duée, 2005) : le module démographique simplifié présenté en partie 3 s'appuiera en partie sur ces paramétrages, en complément du recours aux données des projections standard. Les transitions sur le marché du travail de ce modèle ont été, de leur côté, estimées sur la base des enquêtes *Emploi*. On peut également s'appuyer sur des estimations hors modèle pour le paramétrage de composantes du modèle telles que les équations de salaire. De même, les coefficients d'un modèle de comportement de liquidation inséré dans un modèle de simulation des retraites peuvent avoir été estimés hors modèle puis injectés dans ce modèle.

En revanche, le calibrage ne peut plus se faire en amont du modèle lorsqu'on a affaire à des paramètres inobservés dont l'impact ne se mesure que dans le cadre du fonctionnement global du modèle. Par exemple, dans un modèle appliqué au marché du travail, les paramètres du processus d'appariement entre postes et demandeurs d'emploi ne peuvent être évalués qu'en faisant tourner le modèle et en examinant ce qu'ils impliquent pour diverses variables observées telles que les distributions des durées passées au chômage. Cette situation va être plus typique des modèles d'agents, et ceci explique l'émergence seulement récente de cette problématique dans le cadre des modèles de microsimulation. Elle a plusieurs spécificités qu'on peut essayer de formaliser de la façon suivante, illustrée en encadré par quelques exemples plus précis. On va considérer que le modèle comprend un certain nombre de paramètres inobservés qui peuvent être soit des paramètres macro ou sans variabilité interindividuelle qu'on notera θ_m, soit des paramètres avec variabilité interindividuelle θ_i. Le modèle s'écrit ensuite comme un processus de génération de données D^{sim} qui sont pour l'essentiel des données micro mais qui peuvent aussi inclure des variables macro -par exemple un prix dans un modèle de microsimulation d'un marché-, soit $(D_m^{sim}, D_i^{sim}) = M(\theta_m, \theta_i)$. L'objectif est dès lors de trouver le jeu (θ_m, θ_i) conduisant à des données simulées qui soient aussi proches que possible d'un jeu de données observées (D_m^{obs}, D_i^{obs}). On voit tout de suite que ce rapprochement va poser trois problèmes. Le premier est que le nombre de paramètres est très élevé, le second est que le nombre de dimensions est encore plus élevé pour les données simulées et les données observées et le troisième est enfin que le modèle $M(.)$ n'a pas de forme analytique.

Face à ces difficultés, on est d'abord contraint de réduire la dimension de l'espace des observations. En général, on ne cherche pas à contrôler l'adéquation de l'ensemble des résultats micro du modèle avec des données micro-détaillées individu par individu. Le calibrage et/ou la validation porteront sur des résumés de ces variables qu'on notera $S(D_i^{sim})$ et $S(D_i^{obs})$, par exemple un ensemble de moments des distributions des D_i. Autrement dit les outputs micro sont eux-mêmes convertis en outputs de type agrégés ou semi-agrégés. On notera respectivement $\mathscr{D}^{sim}$ et $\mathscr{D}^{obs}$ ces ensembles combinant données macro D_m et synthèses de données micro $S(D_i)$.

Ensuite, l'absence de forme analytique pour la fonction M exclut le recours à des méthodes d'estimation nécessitant de connaître cette forme analytique. On exclut donc une méthode telle que le maximum de vraisemblance. Elle supposerait de savoir évaluer la probabilité d'observer le jeu $\mathscr{D}^{obs}$ sachant (θ_m, θ_i), ce qui n'est en général pas possible, sauf à essayer de reconstituer la forme de la distribution de $\mathscr{D}$ en tirant un grand nombre de réalisations du modèle pour chaque valeur envisagée des paramètres : cette démarche est très coûteuse car le temps requis pour une simulation unique du modèle est non négligeable, or il s'agirait ici de répliquer un grand nombre de simulations pour un grand nombre de jeux de valeurs des paramètres.

Encadré : Quelques exemples de calibrage

Quatre exemples peuvent servir à illustrer le cadre d'analyse général du calibrage proposé dans le texte, en allant du plus simple au plus complexe.

1) Paramétrisation du choix du départ en retraite dans Destinie : On est dans le cas d'un modèle adossé à une base de données réelles, qui est ici l'enquête *Patrimoine*. À la date initiale, on sait dans l'enquête si les individus sont à la retraite ou pas et à quel âge ils y sont partis. On sait aussi reconstituer à quelle retraite ils auront ou auraient eu droit pour tous les âges de départ qui leur sont ou leur auraient été accessibles. Le ou les paramètres individuels qu'on veut imputer dans le modèle sont soit la préférence pour le loisir (hypothèse de départ en retraite dès l'atteinte d'une cible de taux de remplacement) soit le triplet constitué de la préférence pour le loisir, de la préférence pour le présent et de l'élasticité de substitution intertemporelle (version Stock et Wise). L'ensemble θ^i est donc soit un vecteur simple, soit une matrice qu'on veut imputer de manière cohérente avec ce qu'on sait des différents individus. Imputer ces paramètres est utile à la fois pour les individus encore en activité dont on veut projeter les comportements et pour les individus déjà partis au démarrage de la projection, si on veut simuler les conséquences rétrospectives de contrefactuels neutralisant tout ou partie des réformes passées. À noter que pour les individus qui ne sont pas encore entrés dans la fenêtre de départ (les moins de 60 ans) on n'a aucune information qui vienne contraindre l'imputation : pour ceux-ci, les θ^i seront tirés librement dans les distributions déterminées par leurs moments $S_\theta(\theta^i)$.

2) Fluctuation des cours sur un marché financier (Winker et Gilli, 2001). Il s'agit d'un modèle très simple du marché d'un actif financier avec deux types d'acteurs : des fondamentalistes considérant que le prix doit progressivement revenir vers une valeur fondamentale fixe supposée connue, et des chartistes qui extrapolent la tendance du prix de la période précédente. Les individus peuvent passer aléatoirement d'un type à l'autre selon une probabilité ε et aussi en fonction d'échanges à fréquence δ avec d'autres participants dont ils adoptent le comportement. Il y a également le fait qu'un fondamentaliste peut choisir de se comporter en chartiste s'il pense que, du fait de leur poids, ce sont eux qui déterminent la tendance du marché. Les paramètres du modèle autres que ε et δ sont fixés *a priori* de sorte que le calibrage porte uniquement sur $\theta^m = (\delta, \varepsilon)$. Le calibrage se fait par inférence indirecte. Ce modèle purement illustratif vise à rendre compte des caractéristiques du taux de change DM/US $, résumées par deux paramètres : le coefficient d'hétéroscédasticité d'un modèle ARCH estimé sur cette série, et son coefficient de kurtosis. On notera que, bien qu'il s'agisse d'un modèle d'agents, on ne cherche à rendre compte d'aucune caractéristique observée de ces agents, même globale. L'objectif est uniquement de rendre compte d'une variable macro résultant de l'interaction entre ces agents, c'est-à-dire un output de la forme D_m^{sim}. Comme il n'y a que deux paramètres, l'inférence indirecte peut se faire par inspection graphique de l'écart quadratique entre outputs simulés et valeurs cibles.

3) Modèle du marché du travail (Barlet et al.*, 2009)* : on est dans le cas d'un modèle dynamique visant à reproduire les principales caractéristiques du fonctionnement du marché du travail mais qui n'est pas calé individu par individu sur données micro. Il y a à la fois des paramètres inobservés micro θ^i et des paramètres θ^m de type macro ou, ce qui revient au même, les paramètres qui auraient pu être micro mais qu'on a supposé sans variance interindividuelle, pour ne pas alourdir à l'excès le modèle. Parmi les paramètres micro inobservés figurent les productivités individuelles. On notera que, pour ces paramètres, le problème d'imputation est un problème d'imputation de valeurs initiales en début de simulation. Ces productivités évoluent ensuite selon des processus aléatoires, avec des lois caractérisées par d'autres paramètres de type plutôt macro, en l'occurrence le coefficient de rendement de *l'expérience*, et la moyenne et la variance des chocs de productivité idiosyncratiques. D'autres exemples de paramètres macro introduits dans le modèle déterminent le lien entre taux de marge de l'employeur et risque de licenciement, ou le comportement de négociation salariale de la part des *insiders*. Compte tenu du grand nombre de paramètres, une partie sont fixés de manière exogène au modèle. Le calibrage proprement dit ne porte que sur un petit nombre d'entre eux, par inférence indirecte, avec pour cible de rendre compte au mieux de quelques caractéristiques principales du marché du travail, à savoir des taux de chômage et les distributions des salaires par catégorie sociale.

4) Calibrage ABC d'un modèle simulant des dynamiques démographiques locales (Lenormand et al. *2012).* Il s'agit d'un modèle de population artificielle visant à reproduire les dynamiques démographiques de communes évoluant par renouvellement naturel et sous l'effet de flux migratoires. La modélisation inclut une formation des unions ainsi que des mobilités par rapport à l'emploi, affectées par une dynamique de création d'emplois locaux (Huet et Deffuant, 2011). Le calibrage est fait par un algorithme ABC limité à seulement quatre paramètres du modèle : le nombre d'enfants par femme, la probabilité de changement de résidence, les probabilités de *formation* et de dissolution des unions. Le modèle initialise des populations fictives calées sur les données du recensement de 1990, puis il simule l'évolution de ces populations sur 16 années, le calibrage s'obtenant en testant l'adéquation aux données des recensements de 1999 et 2006. Cette adéquation est testée par un indicateur de distance basé sur huit données synthétiques, à savoir les nombres d'habitants, leur distribution par âge, la distribution par types de ménages et le flux migratoire net, dans chaque commune, à la fois en 1999 et 2006. Le recours à l'approche ABC implique évidemment un volume de calculs très conséquent : environ 10 millions de simulations du modèle, une simulation unitaire prenant 1,4 secondes, dont seulement 8 000 satisfaisant le critère d'acceptation, c'est-à-dire un écart suffisamment faible entre statistiques synthétiques observées et simulées. Ce sont ces 8 000 quadruplets qui servent à établir la distribution *a posteriori*.

Plutôt que de tenter cette démarche très lourde, on peut traiter le problème en cherchant le jeu (θ_m, θ_i) qui minimise une distance $d[\mathscr{D}^{sim}, \mathscr{D}^{obs}]$ entre les statistiques synthétiques issues des données simulées et observées, ce qui correspond au cadre général des méthodes d'inférence indirecte. Mais, même en le simplifiant de la sorte, le problème nécessite qu'on ait beaucoup réduit le nombre de paramètres à estimer. Par exemple, pour les paramètres individuels, il n'est en général pas envisageable que ce soit le choix de chaque paramètre individuel qui fasse l'objet d'un calibrage : le calibrage va plutôt porter sur certaines caractéristiques de la distribution de ces paramètres qu'on pourra noter $S_\theta(\theta_i)$, les valeurs individuelles des θ_i étant ensuite tirées dans ces distributions, la notation S_θ étant utilisée pour distinguer cette fonction résumant les paramètres individuels du modèle et la fonction S servant à résumer les données produites par la simulation.

Les problèmes rencontrés sont à peu près les mêmes si on attaque le problème de manière bayésienne. À première vue, le calibrage d'un modèle de microsimulation semble être un bon domaine d'application de la démarche bayésienne. Puisqu'on a affaire à des modèles à nombre élevé de paramètres, on souhaite se servir du maximum d'*a priori* sur ces paramètres et on aimerait le faire de manière plus flexible et plus rationnelle que la démarche consistant à fixer arbitrairement certains d'entre eux. Mais la démarche baysésienne usuelle suppose que l'on sache évaluer la vraisemblance des données observées pour chaque jeu possible des paramètres (θ_m, θ_i), ce qui est à nouveau impossible. L'alternative est le calcul bayésien approché (ABC pour *approximate bayesian computing*), dont l'idée générale est de fournir une distribution *a posteriori* des paramètres en multipliant les tirages dans leur loi *a priori*, et en ne retenant des jeux ainsi tirés que ceux qui conduisent à des outputs simulés suffisamment proches de l'observation, c'est-à-dire satisfaisant une condition $d[\mathscr{D}_i^{sim}, \mathscr{D}^{obs}] < \varepsilon$ avec ε le seuil d'acceptation. Mais cette approche n'est pas non plus sans difficultés. Un résultat satisfaisant suppose en effet de fixer le seuil ε suffisamment bas, or un seuil bas implique un grand nombre de simulations du modèle sortant en échec, i.e. avec $d[\mathscr{D}_i^{sim}, \mathscr{D}^{obs}] > \varepsilon$, pour un faible nombre de succès. Il faut donc mettre en place des stratégies permettant de minimiser ces échecs, en essayant de concentrer l'exploration de l'espace des paramètres sur les zones qui s'avèrent les plus intéressantes. Même en mettant en œuvre ce type de stratégie, on est contraint par le fait que le temps requis pour une simulation unitaire du modèle n'est pas infinitésimal. Les méthodes ABC ont été

développées à la base pour l'estimation de modèles sans forme analytique mais dont les temps de simulation informatique restent extrêmement faibles, ce qui autorise un grand nombre de tirage de type essai/erreur. Pour des modèles plus lourds, les contraintes de délai de calcul deviennent dirimantes et imposent de limiter la démarche à l'exploration des valeurs possibles pour au plus un nombre très restreint de paramètres, en utilisant de l'information exogène pour fixer arbitrairement les valeurs des autres paramètres. Un exemple de ce type de démarche est l'exemple 4 de l'encadré 1 (Lenormand *et al.*, 2012).

Enfin, on précise que limiter le calibrage à des synthèses des paramètres individuels $S_\theta(\theta_i)$ peut ne pas suffire et qu'il peut y avoir des cas où il est quand même nécessaire de calibrer spécifiquement chaque valeur θ_i. C'est ici que la question du calibrage rejoint celle de l'initialisation des données individuelles. Le calibrage global convient dans des modèles de population totalement fictive : dans ces modèles, on impute les θ_i, de manière totalement libre dans les lois de caractéristiques $S_\theta(\theta_i)$ puis on regarde si la population fictive ainsi simulée et son comportement reproduisent bien les observations $\mathscr{D}^{obs}$ dont on dispose. Mais, dans le cas d'un modèle adossé à des données microéconomiques réelles, on peut avoir le besoin supplémentaire d'une cohérence entre les paramètres θ_i et les observations dont on dispose au niveau de chaque individu. Tel devrait être le cas pour le calibrage des préférences pour la retraite dans le modèle Destinie qui constitue l'exemple 1 de l'encadré : pour les individus en âge de liquider ou d'avoir liquidé, il est normal de chercher à leur imputer des préférences micro cohérentes avec leurs comportements observés. Il n'y a que pour les individus pas encore arrivés à l'âge minimum de la retraite qu'on peut imputer librement. De la même manière, si on voulait reprendre le modèle de marché du travail donné en exemple 3 de l'encadré, mais en l'adossant à une source statistique telle que l'enquête *Emploi*, il faudrait imputer aux individus des productivités inobservées cohérentes avec ce que l'enquête nous dit de leur statut d'emploi et de leurs salaires.

Toutes ces questions dépassent largement le cadre de cette note : on présentera juste un exemple simple de calibrage à deux paramètres en section 2.10. On verra que le recours à un logiciel statistique tel que R offre des facilités intéressantes pour des modèles qui ont besoin de tels calibrages.

1.6 Le choix d'un langage de programmation

Quels sont les autres critères pouvant plaider en faveur d'un langage tel que R pour la programmation de modèles de microsimulation dynamique ? Ces modèles ne font en général pas appel à des concepts de modélisation complexes -le principe de la méthode est très simple-, mais ils nécessitent l'écriture d'un nombre important de lignes de code. La facilité à développer et maintenir ce code dépend des choix de programmation initiaux et on peut distinguer trois grandes options.

La première option est de développer directement dans un langage généraliste en programmant soi-même l'ensemble des fonctionnalités élémentaires dont le modèle a besoin. C'est le choix qui avait été retenu pour les deux versions successives du modèle Destinie, écrites respectivement en turbo-pascal et en Perl. Beaucoup d'autres modèles sont directement développés en langage C++, tels le modèle Myriade de la CNAF.

Une deuxième option est d'utiliser des plateformes spécifiquement dédiées à la microsimulation, c'est-à-dire des biblothèques dans lesquelles sont préprogrammées les opérations de microsimulation les plus usuelles. On peut citer ModGen développé en C++ par Statistique Canada depuis les années 1980[2], LIAM et son successeur LIAM-2 développé en Python (O'Donoghue *et al.*, 2009, Liégeois et Dekkers, 2011) ou bien MIC-CORE (Zinn *et al.*, 2009) ou encore, tout récemment JAMSIM (Mannion *et al.*, 2012). Un très grand nombre

[2]Voir sa documentation à l'adresse www.statcan.gc.ca/microsimulation/modgen/modgen-fra.htm.

d'outils ont également été développés pour la construction de modèles d'agents : un survey remontant à 2009 en dénombrait pas moins d'une cinquantaine dont l'examen sort évidemment totalement du cadre de cette note (Nikolai et Madey, 2009). L'intérêt de cette option est de permettre au modélisateur de se concentrer directement sur le contenu de son modèle mais au prix d'une appropriation préalable de ces plateformes.

La troisième est de s'appuyer sur un logiciel statistique. Le modèle Ines de l'Insee et de la Drees est ainsi développé sous SAS, de même que le modèle Prisme de la CNAV. Le modèle TAXIPP a été développé sous Stata.

Le choix entre ces différents types d'outils dépend de plusieurs facteurs. Les logiciels dédiés à la microsimulation permettent de se concenter directement sur le contenu du modèle mais ils supposent une phase d'appropriation préalable et, ayant souvent été développés pour mettre en œuvre une approche particulière de la méthode, on n'a pas toujours la garantie qu'ils se prêteront bien aux spécificités de l'application qu'on envisage. À l'extrême inverse, les langages généralistes n'enferment dans aucune structure de programmation *a priori*, mais imposent de reprogrammer soi-même un certain nombre de fonctionnalités, ne serait-ce que de simples procédures de tabulation.

Le recours à un logiciel statistique donne pour sa part accès à beaucoup de fonctionnalités prédéfinies par ces outils : génération de nombres aléatoires, tabulation et procédures graphiques... Il a l'intérêt de permettre aux concepteurs de conserver les habitudes de programmation acquises sur des logiciels de ce type. Leur inconvénient est de ne pas toujours se prêter facilement à toutes les manipulations de données requises par la microsimulation. Par exemple, SAS a l'avantage de savoir gérer de très gros fichiers, mais il est surtout adapté au traitement en bloc de tableaux statistiques. Il convient donc bien à des travaux de microsimulation statique consistant à appliquer en une seule fois des traitements globaux à l'ensemble des individus d'un fichier, soumis à des conditionnements simples, sans interactions entre individus. Pour les manipulations plus complexes requises par la microsimulation dynamique, travailler sous SAS oblige à la programmation de macros à la programmation moins intuitive que des étapes data standards. Comme Stata, il s'agit par ailleurs d'un logiciel sous licence ce qui limite la portabilité des programmes. A l'inverse, et ce sera la raison de le mettre ici en avant, le logiciel R est à la fois en accès libre et permet de passer assez facilement d'un mode de traitement des données « en bloc » à un mode de traitement observation par observation, comme on le fait spontanément dans un langage généraliste.

Pour départager entre ces différents styles de programmation, un critère supplémentaire est celui de la vitesse d'exécution. Le tableau 3 présente ainsi les résultats d'un petit test comparatif du délai d'exécution d'une même opération typique d'un modèle de microsimulation dynamique. Elle consiste à simuler une population de 10 000 individus caractérisés par une variable x valant 0 ou 1. Cette population est simulée sur 100 périodes successives, la simulation pour chaque période consistant à attribuer à x la valeur 1 avec une probabilité uniforme de 5% et la valeur 0 dans tous les autres cas. Comme on pouvait s'y attendre, C++ est de très loin le plus rapide, avec un temps d'exécution de 1,4 centième de seconde. Les vitesses d'exécution sont environ 20 fois plus élevées avec les langages interprétés tels que Python ou Perl. La réalisation de la même opération par SAS prend près de 2 secondes, essentiellement en raison de l'obligation de passer par une macro. Les résultats obtenus avec R sont pour leur part très contrastés, car on a testé deux façons différentes de réaliser la même opération sous R : soit sous forme de deux boucles emboitées sur la période et l'ensemble des individus, comme on le fait sous C++, Python et Perl, soit en remplaçant la seconde boucle sur l'ensemble des individus par un ensemble d'instructions vectorielles du type qu'on a au sein d'une étape data

	C++	Python	Perl	SAS	R	
					Par boucles	Manipulation en bloc
Temps CPU en secondes	0,014	0,278	0,282	1,998	3,667	0,061

Table 3: Temps de réponse de différents langages. Analyse comparative

Note : Chacune des simulations considère une population de 10000 individus caractérisés par une variable x initialisée à 0. On itère sur 100 périodes la mise à un de cette variable avec une probabilité fixe de 5%. Sous SAS, l'itération sur 100 périodes suppose le recours à une macro. Pour la simulation sous R, deux variantes sont traitées, selon que les données individuelles sont traitées à chaque période au sein d'une boucle individuelle ou par une instruction vectorielle en bloc. Les simulations ont été effectuées sur MacBook Pro, processeur 2,4 GHz, 4 Go de Ram.

de SAS. Sans atteindre la rapidité de C++, le second mode de programmation conduit à une performance sensiblement meilleure que celle de Python et Perl : R est quatre fois plus lent que C++ mais entre quatre et cinq fois plus rapide que Python ou Perl, et donc beaucoup plus rapide que SAS. En revanche, utiliser R sans aucun recours aux opérations vectorielles conduit à une performance beaucoup moins satisfaisante, encore moins satisfaisante qu'avec SAS.

La façon d'utiliser le langage R influe donc beaucoup sur sa performance, mais l'argument de la rapidité d'exécution n'est cependant pas le seul. Il est décisif lorsqu'on met en œuvre un modèle travaillant sur un très grand nombre d'individus, ou lorsqu'on fait face à des problèmes de calibrage complexes nécessitant de nombreuses simulations successives du même modèle. Dans ces cas, le recours à C++ ou à un code R soigneusement optimisé apparaissent préférables. Si on n'est pas dans ce cas de figure, c'est plutôt le confort de programmation et la lisibilité et la portabilité du code qui doivent primer, et ces critères sont évidemment plus subjectifs que celui de la rapidité d'exécution. Les exemples qui suivront montreront qu'un des intérêts du langage R est de permettre des syntaxes relativement lisibles et compactes. Il se prétait donc bien à la présentation détaillée d'exemples complets permettant de voir concrètement le fonctionnement de la méthode. Les mêmes arguments peuvent plaider pour le recours à ce langage dans le cadre de projets en vraie grandeur, mais le choix d'un langage pour un projet de grande envergure est un choix complexe qui peut faire intervenir un très grand nombre d'autres considérations.

2 Un exemple élémentaire : modélisation de transitions sur le marché du travail.

Cette section se propose d'exposer des principes généraux pour la construction d'un modèle de microsimulation dynamique en langage R, à l'aide d'un exemple rudimentaire. Cet exemple sera la simulation des transitions entre l'emploi et le chômage. Elle se fera sur une population totalement fictive, de taille fixe et, dans un premier temps, sans vieillissement ni renouvellement. À ce stade, l'objectif n'est pas du tout d'illustrer l'apport de la microsimulation. Pour un modèle aussi simple, la simulation n'apporte que peu de choses par rapport à l'étude analytique des propriétés du modèle, laquelle est elle-même très simple. Dans une population où les probabilités d'entrée et de sortie du chômage sont des paramètres fixes p_e et p_s, on sait que le taux de chômage d'équilibre doit vérifier l'égalité entre flux d'entrées et flux de sortie $(1-U)\,p_e = Up_s$ soit $U = p_e/(p_e+p_s)$. Ce taux de chômage d'équilibre est une fonction croissante de la probabilité d'y entrer et décroissante de la probabilité d'en sortir. On connait aussi la forme analytique de l'évolution du taux de chômage lorsqu'on part d'une valeur $U(0)$ différente de cette valeur d'équilibre. En temps continu, la dynamique du taux de chômage est en effet déterminée par l'équation :

$$\frac{dU}{dt} = -p_s U + p_e(1-U) = p_e - (p_e+p_s)U$$

de solution :

$$U(t) = \frac{p_e}{p_e+p_s} + \left[U(0) - \frac{p_e}{p_e+p_s}\right] \mathrm{e}^{-(p_e+p_s)t} \tag{1}$$

La microsimulation ne fera que confirmer ces propriétés, à quelques complexifications près qui n'apparaîtront qu'en fin de section. L'exemple a été construit dans le seul but d'illustrer les principes de base de la méthode. Pour ce qui concerne le langage R, on fera l'hypothèse que la syntaxe de base en est connue, et on ne détaillera donc que des éléments de programmation moins élémentaires, tels que la gestion de tableaux multidimensionnels ou la programmation de fonctions.

2.1 Un premier exemple : simulation de transitions emploi chômage.

L'exemple développé ci-dessous est donc notre premier exemple de programme de microsimulation. Comme annoncé plus haut en section 1.4, tous les exemples de cette section 2 porteront sur des populations fictives dont les caractéristiques initiales sont donc générées par le modèle lui-même. La population considérée est de 1000 personnes. Au départ de la simulation, 90% de ces personnes sont dans l'état d'actif occupé codé 1 et les 10% restantes sont au chômage, codé 2. On simule ensuite 25 périodes successives pendant lesquelles le risque de tomber au chômage pour un actif occupé est de 2,5% contre 50% pour la probabilité de retour à l'emploi d'un chômeur.

```
# Programme Exemple_1.R

taille              <- 1000
statut              <- numeric(taille)
txcho               <- numeric(25)
statut              <- 1+rbinom(taille,1,0.1)
```

```
proba_sortie_chomage <- 0.5
proba_entree_chomage <- 0.025
for (t in 1:25)
{
  txcho[t] <- 100*sum (statut==2)/taille
  for (i in 1:taille)
  {
    if (statut[i]==1)
    {
      if (runif(1)<proba_entree_chomage)
      {
        statut[i]<-2
      }
    }
    else if (statut[i]==2)
    {
      if (runif(1)<proba_sortie_chomage)
      {
        statut[i]<-1
      }
    }
  }
}
```

Ce premier exemple permet d'illustrer la structure-type d'un modèle de microsimulation, incluant la phase d'initialisation des variables micro et des paramètres, puis les deux boucles imbriquées sur la date et l'ensemble des individus simulés avec, dans la boucle individuelle, les transitions conditionnées par l'état courant. Pour les lecteurs non familiers de R, on relèvera la forme de l'instruction d'affectation, qui s'écrit `<-`[3]. Les vecteurs et leur longueur font l'objet d'une déclaration préalable du type `x <- numeric(n)`[4]. On relève aussi la syntaxe des générateurs d'aléatoires. De manière générale une instruction `rxxxx(n)` génère un vecteur de `n` valeurs aléatoires tirées selon la loi `xxxx`. L'instruction `runif(1)` simule donc un tirage unique selon une loi uniforme sur [0,1], `rbinom(taille,1,0.1)` simule 1000 tirages binomiaux de paramètres 1 et 0,1, c'est-à-dire 1000 tirages bernouillens selon la probabilité p=0,1. On mentionne aussi le recours à la notion dite de masque logique dans l'instruction `txcho[t] <- 100*sum(statut==2)/taille` qui utilise un comptage direct des individus remplissant la condition `statut==2` pour le calcul du taux de chômage exprimé en %. Ce mode de sélection des observations est un des atouts importants du langage R et il réapparaitra de manière régulière dans la suite, alternativement avec l'instruction `which` qui permet de sélectionner les indices des individus remplissant une condition donnée : davantage de détails sur ces deux modes de sélection des données en annexe 1.

[3] R autorise aussi le signe = utilisé dans la majorité des autres langages mais la forme <- est la forme standard dont on verra plus loin en section 2.8 la variante dite de superaffectation notée <<-. Cette variante permet à des fonctions R de modifier les valeurs de variables globales extérieures à ces fonctions.

[4] Une instruction équivalente est `x <- vector(mode="numeric", length=n)`. Dans un cas comme dans l'autre, les éléments de ces vecteurs sont initialisés à zéro. On précise par ailleurs que, dans ce cas des vecteurs, la taille déclarée n'est pas contraignante. Une instruction `statut[2000]<-2` à l'intérieur du programme porterait à 2000 la taille du vecteur `statut` en laissant à la modalité `NA` (donnée manquante) les positions 1001 à 1999. Mais on n'exploitera pas ici cette possibilité d'extension dynamique qui, de toute manière, n'existe pas pour les objets de type `matrix` ou `array` qui seront mobilisés plus loin. Ne pas utiliser cette possibilité impose d'avoir une idée *a priori* du nombre maximum d'individus que le modèle peut être amené à gérer. Il peut être très supérieur au nombre d'individus initial, surtout lorsque les positions laissées vacantes par les individus qui quittent la population ne sont pas réutilisées pour gérer les individus qui y arrivent, comme ce sera le cas dans l'adaptation en R du modèle Destinie présentée en troisième partie de ce document.

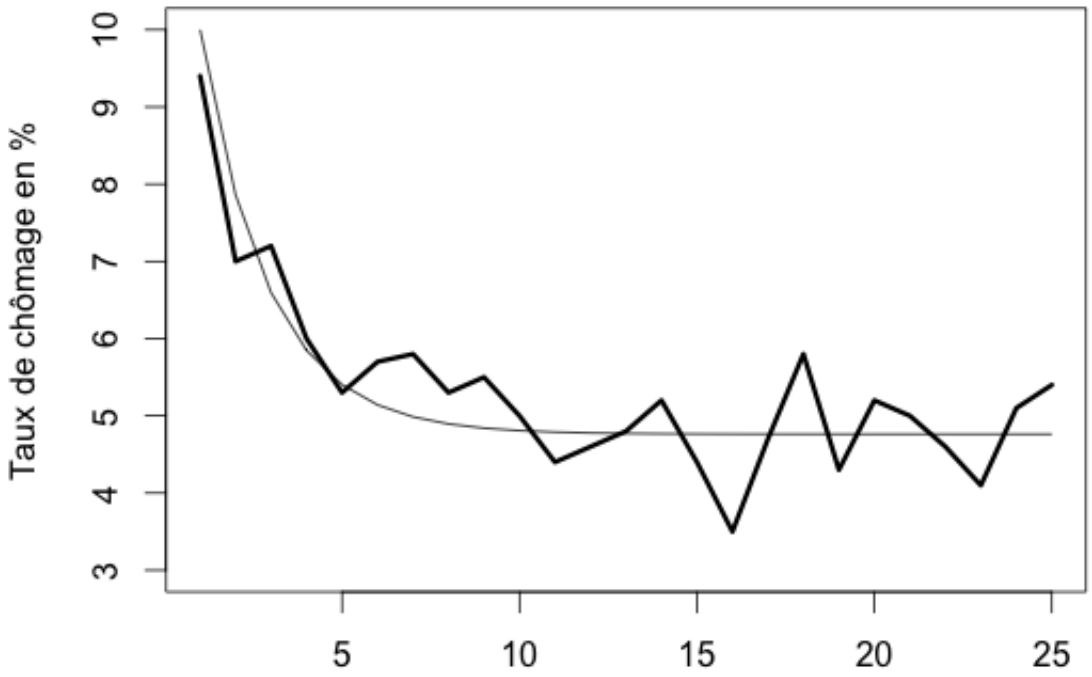


Figure 1: Résultat du programme `Exemple_1.R`. Evolution du taux de chômage simulé (trait épais) comparé à son évolution théorique (trait fin).

Le graphique 1 donne la dynamique simulée avec ce programme pour le taux de chômage (trait épais). À partir d'un niveau initial de l'ordre de 10% , on redescend rapidement vers un niveau d'environ 5%. Cette valeur d'équilibre correspond bien à la formule $U = p_e/(p_e + p_s) = 0,025/(0,525) = 0,0476$, et la convergence se fait bien autour de la courbe théorique de l'expression (1) donnée en trait fin sur le même graphique.

2.2 Utiliser la manipulation globale des vecteurs

Ce premier exemple n'a pas utilisé le style de programmation qui est général recommandé aux utilisateurs de R. À l'exclusion de l'appel à `rbinom` qui génère en une seule fois l'ensemble des statuts initiaux, les autres calculs sont faits à l'intérieur de boucles traitant séparément chaque individu. Or, comme on l'a déjà annoncé en section 1.6, une utilisation plus efficace du langage consiste à minimiser le recours à ces boucles car elles sont plus lentes que les instructions de manipulation de variables en bloc. L'autre avantage de la manipulation globale est qu'elle permet d'avoir des codes plus compacts. Tout au long de ce document, on verra qu'on alternera les deux modes de travail : travailler individu par individu à l'intérieur de boucles est difficilement évitable lorsque les opérations à effectuer sur chaque individu sont complexes mais, pour des changements d'état aussi simples que ceux qu'on simule ici, le traitement en bloc est une option intéressante, si l'on sait comment gérer la sélection d'individus à l'intérieur d'instructions globales, en recourant à nouveau aux notions de filtres ou de masques logiques décrites plus précisément en annexe 1.

La séquence suivante reproduit ainsi ce que faisait le premier programme mais en remplaçant la boucle

sur `i` et les blocs conditionnels au statut courant par seulement deux instructions.

```
# Programme Exemple_2.R

taille               <- 1000
statut               <- numeric(taille)
txcho                <- numeric(25)
statut               <- 1+rbinom(taille,1,0.1)
proba_sortie_chomage <- 0.5
proba_entree_chomage <- 0.025

for (t in 1:25)
{
  txcho[t] <- 100*sum(statut==2)/taille
  statut[statut==1 & runif(taille)<proba_entree_chomage] <- 2
  statut[statut==2 & runif(taille)<proba_sortie_chomage] <- 1
}
```

Par exemple, dans la première des deux instructions de redéfinition du statut, la séquence `statut==1 & runif(taille)<proba_entree_chomage` définit un nouveau vecteur de booléens qui sert de masque logique pour la sélection des observations du vecteur `statut` auxquelles va être affectée la nouvelle valeur de 2. Cette séquence fait intervenir au passage un autre vecteur intermédiaire non nommé, celui qui est généré par l'appel `runif(taille)` qui peut être utilisé dans une instruction combinée avec le vecteur `statut` puisqu'il est de même longueur. Ce second programme est effectivement bien plus rapide que le premier : sur la même machine que celle utilisée pour le test comparatif du tableau 3, il tourne en 0,014 seconde -hors sortie graphique- contre 0,150 seconde pour le premier. L'écart relatif est moindre que celui que montrait le tableau 3 car la pénalité induite par le fonctionnement en boucle est une fonction croissante du nombre d'individus traités, mais il reste très important. Si le critère de la vitesse d'exécution est important, il faut donc chercher à maximiser le recours à ces instructions en bloc.

Ceci ayant été rappelé, deux remarques peuvent être faites quand à cette nouvelle écriture.

Tout d'abord, dans l'esprit de réduire encore davantage le temps d'exécution, on peut se demander s'il était vraiment nécessaire de tirer deux séquences entières de 1 000 nombres aléatoires, et donc envisager plutôt le genre de séquence suivant :

```
alea <- runif(taille)
statut[statut==1 & alea<proba_entree_chomage)] <- 2 # instruction A
statut[statut==2 & alea<proba_sortie_chomage)] <- 1 # instruction B
```

Cette séquence conduit cependant à un résultat erroné. En effet, puisque `proba_entree_chomage` est supérieur à `proba_sortie_chomage`, tous les individus entrés au chômage au titre de l'instruction `A` vérifient *a fortiori* la condition `alea<proba_sortie_chomage` et ressortent donc aussitôt du chômage au titre de l'instruction `B`. De ce fait le modèle ne génère plus aucune entrée au chômage d'une année sur l'autre et, de proche en proche, le taux de chômage tend vers zéro. Cet exemple montre l'importance de bien maitriser la façon dont sont gérées les séquences de tirage aléatoire. En règle générale, il vaut mieux utiliser trop de tirages successifs que de risquer des tirages non indépendants.

Même en évitant cette erreur, il subsiste une différence avec le programme précédent : dans le programme

précédent, les individus mis au chômage une année donnée ne pouvaient pas sortir du chômage la même année alors que ce sera le cas ici, le second tirage se faisant sur l'ensemble de chômeurs après les mises au chômage générées par la première instruction. Ce point n'est pas secondaire : il en résulte une valeur limite du chômage deux fois moindre (figure 2). En effet, dans cette nouvelle formulation, une moitié des individus tombés au chômage une année donnée en ressortent aussitôt et n'apparaissent jamais comme chômeurs. Le résultat aurait été différent en intervertissant la simulation des entrées et des sorties. Dans ce cas, on interdit certes les rechutes immédiates au chômage pour les individus qui viennent de retrouver un emploi, mais la probabilité de tomber au chômage étant plus faible, on retombe sur une valeur plus proche de celle de la première simulation et du modèle théorique.

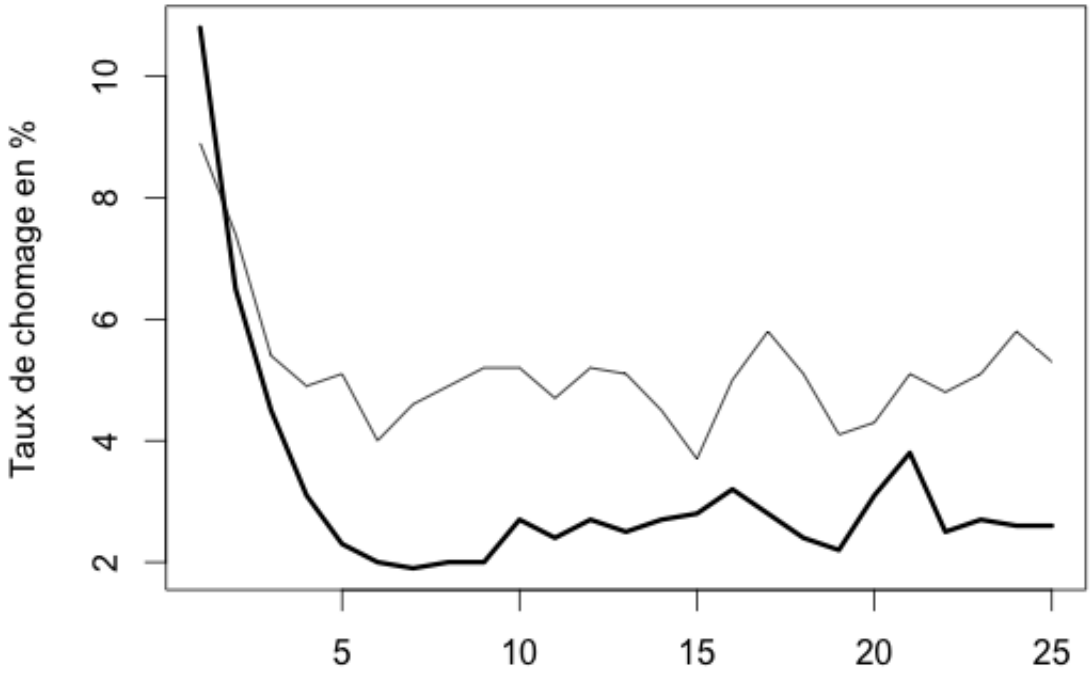


Figure 2: Résultats du programme `Exemple_2.R`. Evolution du taux de chômage selon que les entrées en chômage sont simulées avant (trait épais) ou après (trait fin) les sorties.

Les techniques permettant de mieux gérer cette question des ordres d'apparition des évènements ne seront pas approfondies dans ce document introductif mais on peut en donner un bref aperçu (Willekens, 2009). En général, la stratégie est de réduire le pas de simulation de manière à réduire la fréquence de l'ensemble des évènements. À la limite, certains modèles adoptent un traitement continu du temps : lorsqu'un individu est exposé à plusieurs risques concurrents, des modèles de durée sont utilisés pour tirer au hasard la date d'occurence potentielle de chacun d'entre eux (par exemple un licenciement ou un passage volontaire en inactivité) et c'est le plus proche de ces différents évènements qui est alors simulé, après quoi on resimule de la même manière la réalisation des évènements auquel l'individu se retrouve exposé près cette transition.

Mais cette façon de résoudre le problème alourdit sensiblement la programmation et n'est applicable qu'à des modèles simulant les trajectoires individuelles indépendamment les unes des autres, comme le font plutôt les modèles de cohorte. Lorsque le modèle simule des individus interdépendants, il a besoin de points de rendez-vous réguliers auxquels évaluer les situations de l'ensemble des individus et les interactions qui en résultent, ce qui exclut la simulation en temps continu. Dans ce cas, on est donc réduit à la stratégie consistant à réduire le pas de simulation, ce qui ne pose pas de problème de programmation, mais allonge évidemment les temps de calcul.

Plus généralement, le problème qu'on vient de soulever illustre un point de vigilance général que requièrent ces modèles. Les résultats peuvent être sensibles à des détails apparemment secondaires de spécification et, ici, c'est le fait de disposer d'un point de référence analytique qui a permis de détecter le problème. Malheureusement, pour des modèles en vraie grandeur, de telles références théoriques sont en principe indisponibles puisque c'est précisément l'interêt de la méthode d'apporter des réponses à des problèmes de modélisation qu'on ne sait pas résoudre analytiquement. Mais tout ce dont on dispose et qui peut contribuer à des validations partielles du modèle est bienvenu. À la limite, ceci peut prendre la forme d'un modèle du modèle aidant à reconstituer plus directement certains de ses agrégats. Dans le cas du module démographique du programme Destinie présenté en section 3, on verra que ce modèle du modèle sera naturellement fourni par les projections démographiques classiques, au point d'utiliser celui-ci comme instrument de calage de la microsimulation, ce qui a l'intérêt de rattraper mécaniquement les éventuelles limites de la programmation au niveau micro. Mais cette technique n'est pas non plus sans danger : le calage sur données macro d'un modèle mal spécifié peut donner l'apparence d'un modèle qui fonctionne, alors que ce calage a pu avoir pour impact de déplacer les conséquences de l'erreur de spécification vers d'autres variables ne donnant pas lieu à calage. Nous rementionnerons ce point plus loin en section 2.5.

2.3 Garder la mémoire des états antérieurs : le recours aux matrices

Dans les deux exemples qu'on vient de présenter, la mémoire de la variable `statut` n'est pas conservée d'une année sur l'autre. Elle ne sert qu'à conserver le statut courant. Mais de nombreuses applications peuvent nécessiter de garder trace de ces statuts passés. Cela sera le cas si ces informations rétrospectives sont nécessaires à la simulation. Par exemple, on peut faire dépendre les entrées-sorties du chômage de la fréquence des épisodes de chômage passés. Ou, encore, dans un modèle de retraite, on va avoir besoin de la séquence des salaires et statuts d'activité passés pour calculer la retraite au moment de la liquidation. La conservation des statuts passés peut être aussi utile à fin d'exploitation des résultats : par exemple, ce peut être un des objectifs du modèle de produire des données simulées sur les durées passées dans tel ou tel état et leur répartition au sein de la population.

Pour ce faire, il suffit de substituer au vecteur `statut` une matrice. La déclaration se fait sous la forme

```
statut <- matrix(nrow=1000,ncol=25)
```

Une fois ceci fait, on peut accéder soit à la matrice dans son entier, soit aux éléments individuels `statut[i,t]`, soit encore à des coupes qui peuvent être verticales ou horizontales. Ainsi

```
statut[,10]
```

sera le vecteur des statuts des différents individus à la date 10, qu'on pourra manipuler comme on manipulait, à la date 10 le vecteur simple `statut` des deux exemples précédents. Alternativement

```
statut[100,]
```

donnera la séquence des statuts de l'individu 100 aux différentes dates et

```
statut[100,10:20]
```

donnera la même séquence mais uniquement entre les dates 10 et 20.

Sur cette base, on réécrit le programme précédent de la manière suivante :

```
# Programme Exemple_3.R

taille               <- 1000
statut               <- matrix(nrow=taille,ncol=25)
txcho                <- numeric(25)
statut[,1]           <- 1+rbinom(taille,1,0.1)
proba_sortie_chomage <- 0.5
proba_entree_chomage <- 0.025

for (t in 1:25)
{
  txcho[t] <- 100*sum(statut[,t]==2)/taille
  if (t<25)
  {
    statut[,t+1] <- statut[,t]
    statut[statut[,t]==1 & runif(taille)<proba_entree_chomage,t+1] <- 2
    statut[statut[,t]==2 & runif(taille)<proba_sortie_chomage,t+1] <- 1
  }
}
```

Les points importants à souligner dans ce programme modifié sont les suivants :

- Il faut penser à faire porter l'initalisation sur la première colonne du vecteur statut, soit `statut[,1]`
- Il faut prévoir dans la boucle une actualisation du statut pour les individus qui ne changent pas de statut entre les deux dates successives. C'est fait ici par l'affectation globale `statut[,t+1]<-statut[,t]`, dont le résultat sera ensuite modifié pour les individus qui passent d'un état à l'autre entre les deux dates.
- On relève aussi la complexification de l'instruction d'affectation pour ces individus qui changent de statut : le test de statut initial se fait sur la colonne `t` de la matrice statut alors que l'affectation se fait sur sa colonne `t+1`, d'où l'instruction :

```
statut[statut[,t]==1 & runif(1000)<proba_entree_chomage),t+1] <- 2
```

Une fois que ceci a été fait divers calculs sont possibles sur la matrice stockant l'ensemble des statuts successifs. Par exemple, la séquence

```
duree_cho <- numeric(taille)
for (i in 1:taille)
{
   duree_cho[i] <- length(which(statut[i,]==2))
}
```

fournira directement les durées passées au chômage par les différents individus au cours des 25 années de simulation, dont l'histogramme est fourni par la figure 3.

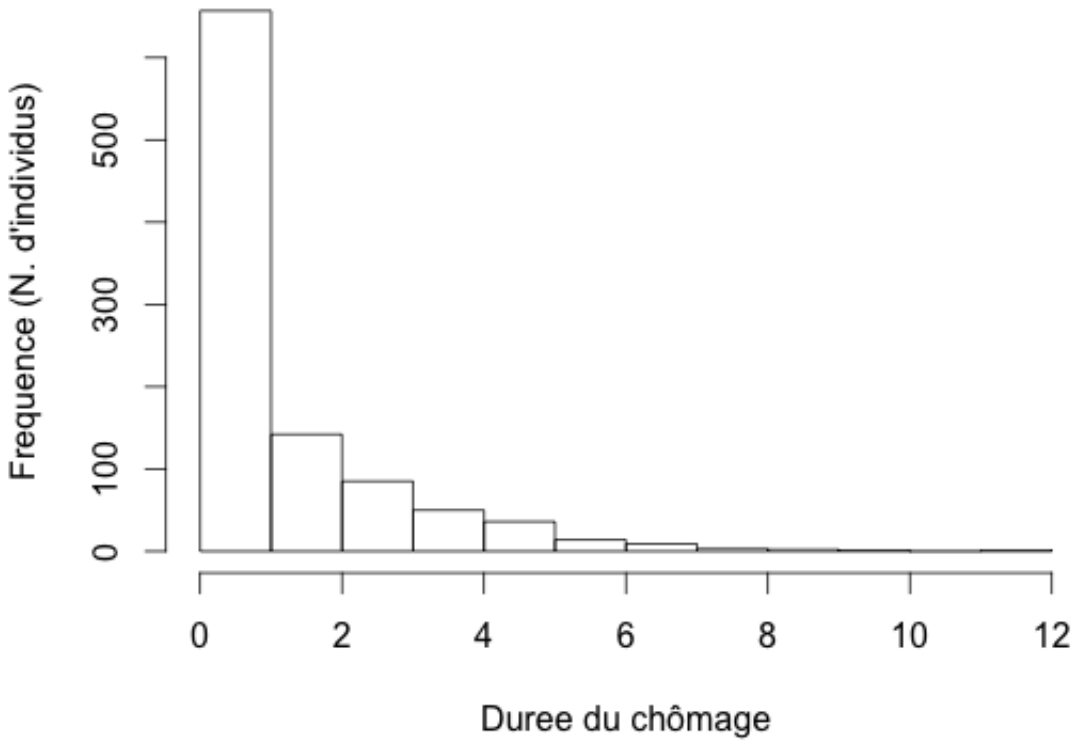


Figure 3: Résultat du programme `Exemple_3.R`. Distribution des durées passées en chômage

2.4 Un premier remède à la variabilité stochastique : accroître la taille de l'échantillon simulé ou multiplier les simulations

Comme on l'a rappelé en section 1, la microsimulation a l'avantage de permettre de traiter de phénomènes complexes en les simulant au plus près de la façon dont se comportent ou évoluent les individus, mais, en contrepartie, ses résultats sont affectés d'une variabilité stochastique qui apparaissait bien sur les figures 1 et 2. Les conséquences de cette variabilité ne doivent pas être exagérées : les statistiques qu'on manipule quotidiennement sont aussi affectées d'aléas d'échantillonnage. Avoir le même problème sur des résultats de projections n'est ni plus ni moins génant que pour l'observation courante, mais il reste néanmoins souhaitable de réduire au minimum l'ampleur de cet aléa, lorsqu'il y a des moyens de le faire.

Dans ce but, deux stratégies élémentaires consistent soit à travailler sur des échantillons de taille plus importante, soit à multiplier les simulations et travailler sur des moyennes de leurs résultats. Les deux méthodes sont à peu près équivalentes. Leur défaut commun est de pouvoir devenir assez rapidement coûteuses en temps de calcul, mais elles ne posent pas de problèmes conceptuels particuliers. On propose néanmoins un programme mettant en l'œuvre l'une et l'autre car ceci va donner l'occasion d'introduire la façon dont R permet de programmer des fonctions.

On va présenter le code de ce nouveau programme en deux parties. La première est la déclaration d'une fonction qui reprend l'essentiel du programme `Exemple_3.R` mais avec une taille d'échantillon para-

métrable. Ce segment de programme est le suivant :

```
# Programme Exemple_4.R: définition de la fonction simul

simul <- function(taille,horizon=25,
                  proba_sortie_chomage=0.5,
                  proba_entree_chomage=0.025)
{
  statut     <- matrix(nrow=taille,ncol=horizon)
  txcho      <- numeric(horizon)
  statut[,1] <- 1+rbinom(taille,1,0.1)
  for (t in 1:horizon)
  {
    txcho[t] <- 100*sum(statut[,t]==2)*/taille
    if (t<horizon)
    {
      statut[,t+1] <- statut[,t]
      statut[statut[,t]==1 & runif(taille)<proba_entree_chomage,t+1] <- 2
      statut[statut[,t]==2 & runif(taille)<proba_sortie_chomage,t+1] <- 1
    }
  }
  return (txcho)
}
```

Cet exemple montre qu'une fonction se déclare comme n'importe quel autre objet R, c'est-à-dire par son nom suivi d'une instruction d'affectation `<-function()` où la liste fournie entre parenthèses est la liste des paramètres d'appel de la fonction. Le corps de la fonction suit entre deux accolades. Et les données qui sont retournées par la fonction sont les données figurant entre parenthèses, en général en fin de programme, comme argument de l'instruction `return`. Ici, le programme retourne la séquence des taux de chômage simulés, mais il pourrait aussi bien retourner l'intégralité du tableau des statuts successifs des différents individus simulés.

On notera que les paramètres de la fonction sont de deux types : des paramètres de valeur non prédéfinie (ici le paramètre `taille`) auxquels il sera donc indispensable d'affecter une valeur au moment de l'appel et des paramètres auquels la fonction affecte une valeur par défaut, à l'aide cette fois du signe = plutôt que <-. Il s'agit ici des paramètres `horizon`, `proba_entree_chomage` et `proba_sortie_chomage`. Pour ces paramètres, on ne spécifiera de valeur lors de l'appel que si on veut des valeurs différentes des valeurs par défaut. En utilisant les noms des paramètres, il est par ailleurs possible de les renseigner dans un ordre différent de celui dans lequel ils sont déclarés dans la fonction. On pourra ainsi avoir des appels du type :

```
simul(1000)
simul(1000, 30)
simul(1000, horizon=30)
simul(1000, proba_sortie_chomage=0.4)
simul(horizon=30, taille=1000)
```

Par ailleurs, on constate qu'il y a déclaration d'un certain nombre de variables propres à la fonction, principalement les variables `statut` et `txcho`. Comme les paramètres, ces variables sont des variables

internes à la fonction qui, le cas échéant, cachent localement les variables de même nom ayant pu être définies à l'extérieur de la fonction.

On va en avoir un exemple immédiat avec la façon dont la suite du programme utilise cette fonction, fournie ci-dessous.

```
# Programme Exemple_4.R: corps de programme

txcho <- matrix(nrow=11,ncol=25)

# 10 simulations successives et stockage de la moyenne en
# position 11 de la matrice txcho
for (sim in 1:10)
{
  txcho[sim,] <- simul(1000)
}
for (t in 1:25)
{
  txcho[11,t]<-mean((txcho[1:10,t]))
}
```

On voit que ce bloc de programme déclare lui aussi une variable `txcho`, qui est cette fois une matrice, et qui va servir à stocker les résultats de 10 simulations successives dont on fait ensuite la moyenne, en position 11 du même tableau. C'est l'objet des deux premières boucles, la seconde recourant à la fonction `mean` qui produit une moyenne simple des éléments du vecteur passé en argument. Cette variable `txcho` est bien gérée indépendamment de l'objet du même nom qui est manipulé à l'intérieur de la fonction `simul`. Les résultats de ce programme sont fournis par la figure 4 : de manière assez prévisible, itérer 10 simulations ou simuler sur une population 10 fois plus grande donne à peu près le même résultat en termes de réduction de la variance, et cette réduction reste loin d'être totale, même si on se rapproche beaucoup de la courbe théorique qui était donnée sur la figure 1. Privilégier l'une ou l'autre des deux méthodes dépend de l'objectif choisi : dans certains cas, la variance des résultats simulés est une donnée qu'on cherche aussi à évaluer, par exemple si on s'intéresse à la variabilité potentielle de tel ou tel agrégat sur une petite population qu'on microsimule exhaustivement, auquel cas on privilégiera plutôt la multiplication des simulations plutôt que la simulation unique sur une population plus nombreuse.

2.5 Les remèdes au problème de la variabilité stochastique : réduction de variance et alignement

Accroître la taille des échantillons simulés ou multiplier les simulations sont deux techniques simples mais qui ont l'inconvénient de devenir rapidement très coûteuses en temps de calcul sur des modèles un peu conséquents. Deux techniques alternatives sont la réduction de variance ou le calage, également qualifié d'alignement, dont on va d'abord présenter les principes généraux, en s'appuyant sur Morrison (2006) ainsi que Li et O'Donoghue (2011) et dont on verra l'application à notre modèle simple de marché du travail dans la section suivante, avant qu'elles ne soient reprises de façons plus systématique et plus réaliste dans la présentation du module démographique du modèle Destinie 2.

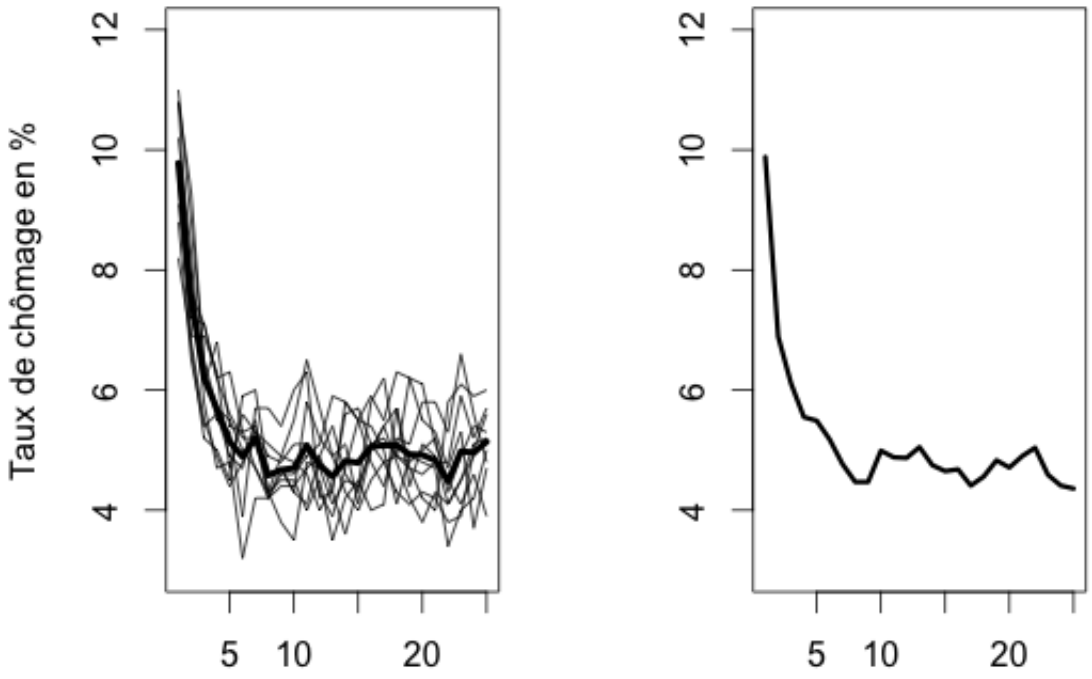


Figure 4: Résultats du programme `Exemple_4.R`. Moyenne de simulations multiples (à gauche) et simulation sur échantillon plus large (à droite)

2.5.1 Réduction de variance : simuler des nombres d'évenènements conformes aux nombres espérés

Le principe de la réduction de variance est le suivant. Soit la simulation d'un événement particulier à l'intérieur du modèle. Soit $i = 1, .., N$ l'indice désignant les individus et p_i la probabilité de connaître l'événement pour l'individu i. À partir de ces probabilités, la méthode de microsimulation par défaut consiste à tirer pour chaque individu un aléa u_i compris entre zéro et un et à ne faire se réaliser l'événement que pour les individus pour lesquels $u_i < p_i$. Le nombre d'évènements qu'on obtient dans ce cas sera en moyenne de Σp_i mais avec une variance $\Sigma p_i(1 - p_i)$, et donc un coefficient de variation $\sum p_i / \sqrt{\sum p_i(1 - p_i)} \simeq 1/\sqrt{\sum p_i}$ d'autant plus grand que l'évènement simulé est rare et la population simulée petite, alors que la simulation déterministe simulerait un nombre de transitions exactement égal à Σp_i. L'objectif est de réduire cette variabilité qui n'est pas informative. Il suffit pour cela de contraindre le tirage aléatoire de sorte à ce qu'il donne exactement Σp_i résultats positifs. Dit autrement, ce qu'on va demander au modèle de microsimulation est de désigner les individus qui vont effectivement connaître l'événement, et non pas leur nombre total qu'on préfère fixer de manière déterministe.

Que faire pour obtenir un nombre de tirages aussi peu dispersé que possible autour de Σp_i ? Si les probabilités p_i étaient égales pour tous les individus, il suffirait de réordonner aléatoirement la liste des N individus exposés à la probabilité fixe p de survenue de l'évènement, et d'en retenir les Np premiers. Cette méthode n'est pas applicable lorsque le risque de réalisation de l'évènement varie d'un individu à l'autre. Dans ce cas, on peut recourir à l'algorithme de tirage systématique usuellement utilisé pour le tirage d'échantillons

de sondage de taille fixe à probabilités d'inclusion inégales (voir par exemple Tillé, 2001; Ardilly, 2006). Cet algorithme est le suivant :

- On initialise un compteur U à une valeur u_0 tirée uniformément entre zéro et un.
- Ensuite, pour chaque individu i, on ajoute p_i à U. À chaque fois que cet ajoût fait franchir une valeur entière, on ajoute i à la liste des individus tirés. De manière totalement équivalente, le test peut consister à retenir i et à décrémenter le compteur de un à chaque fois que ce compteur franchit l'unité.

Dans la littérature consacrée à la microsimulation, cet algorithme est aussi qualifié de *sidewalk algorithm*, l'image étant celle d'une marche à pas de longueurs variables sur un dallage régulier, à partir d'une position initiale aléatoire (Morrison, 2006). On devine comment il donne des probabilités de tirage conformes aux probabilités d'inclusion : la probabilité de franchir un nombre entier à l'étape i est d'autant plus grande qu'on incrémente le compteur U d'une quantité p_i importante. Elle est égale à un si cette probabilité p_i est elle même égale à un. Elle est nulle dans le cas extrême inverse. Le fait d'initialiser le processus par un premier tirage uniforme entre zéro et un permet, pour sa part, de donner des chances positives d'être tirés aux premiers individus de la liste. Par exemple, la probabilité d'être tiré pour le premier individu de la liste correspondra à la probabilité que l'aléa initial u_0 soit supérieur à $1 - p_1$. Elle sera donc bien égale à p_1. Quant au nombre total d'individus qui seront tirés, il correspond au nombre de franchissements de valeurs entières par le compteur U et il correspondra donc à $int(u_0 + \Sigma p_i)$, en notant $int(x)$ la partie entière de x. Ce nombre de franchissements sera ainsi égal à $int(\Sigma p_i)$ avec une probabilité de $int(\Sigma p_i) + 1 - \Sigma p_i$ et à $int(\Sigma p_i) + 1$ avec la probabilité complémentaire : si par exemple le nombre moyen d'évènements attendus est de $\Sigma p_i = 6,2$, le tirage donnera 6 individus dans 80 % des cas et 7 individus dans les 20 % restants. On gère donc bien sans biais le fait que les évènements ne peuvent être simulés que sur des nombres entiers d'agents.

Cette méthode présente néanmoins une limite, qui est que les probabilités d'être sélectionné ne sont pas indépendantes entre individus adjacents : elle ne respecte pas les probabilités d'inclusion du second ordre. Le fait que l'individu i ait été tiré conduit à une très faible probabilité d'être tiré pour l'individu situé juste après. Si on reprend le cas particulier d'un tirage équiprobable, par exemple avec $p_i = p = 0,1$, on voit par exemple que la méthode va consister à tirer un individu au hasard parmi les dix premiers, puis à prendre systématiquement chaque dixième individu parmi les individus suivants. Si les individus sont rangés dans la base selon une séquence non aléatoire, il peut en résulter des biais systématiques dans la composition de la population tirée. Pour résoudre ce problème, on peut faire précéder le tirage d'une permutation aléatoire de l'ensemble des individus susceptibles d'être tirés. Ceci sera fait d'office dans la fonction de tirage qui sera présentée plus loin.

2.5.2 Alignement: s'ajuster sur des cibles déterministes exogènes

Le calage ou l'alignement sont des généralisations de cette démarche au cas où on veut que le nombre d'évènements tirés respecte une autre cible que Σp_i. Dans quel cas peut-on souhaiter un tel alignement ? Un exemple est le cas du calage de la microsimulation sur les flux donnés par la projection démographique déterministe classique dont le principe a été rappelé en section 1.2. Dans cette projection déterministe par la méthode dite « des composants », on a vu que le flux de naissances annuel ne dépend que des effectifs et des taux de fécondité féminins par âge. De son côté, un modèle de microsimulation fera en général intervenir des déterminants bien plus nombreux, parce qu'il cherche à rendre compte de caractéristiques plus fines de

la distribution des tailles de familles selon les caractéristiques des parents. On pourrait considérer que ce déterminisme plus riche fait la supériorité du modèle de microsimulation et qu'il n'y a pas lieu d'en caler les résultats sur ceux de la méthode traditionnelle. Le calage peut même s'avérer dangereux s'il est seulement utilisé comme moyen de rattraper les résultats d'un modèle non calé qui a été mal spécifié (Klevmarken, 1998). Une précaution minimale est de n'appliquer le calage que sur un modèle dont on aura d'abord testé le comportement non calé, pour s'assurer que le calage ne jouera qu'un rôle résiduel (Wolf, 2001; Bacon et Pennec, 2009). Mais il existe de nombreuses raisons qui plaident pour un usage raisonné du calage. Par exemple, les projections construites par la méthode des composants ont souvent le statut de projection de référence, et on peut vouloir un modèle de microsimulation cohérent avec ces projections, considérant que l'apport de la microsimulation est davantage la prévision des structures fines des ménages que la prospective d'ensemble sur la dynamique démographique. Dans ce cas, le modèle ne prétend plus être le mieux placé pour prévoir combien de naissances vont survenir dans l'année -il laisse ce soin à la projection de calage-, mais il propose une répartition individuelle réaliste de ces naissances entre les différents ménages susceptibles d'en avoir une. Du même coup, le flux global d'évènements simulés perdra son caractère aléatoire. C'est uniquement l'identité des individus ou ménages concernés qui sera stochastique.

Comment procède-t-on pour ce faire ? Notons K la cible du nombre d'évènements à simuler. Une démarche consiste à appliquer l'algorithme de tirage précédent à une transformée $f(p_i)$ des probabilités initales p_i, telle que $\Sigma f(p_i)$ soit égal à K. On pourrait se borner à simplement multiplier l'ensemble des p_i par $K/\Sigma p_i$. On peut souvent s'en contenter mais, si K est très supérieur à Σp_i et si certains des p_i sont proches de un, la transformation peut conduire à des $f(p_i)$ supérieurs à un. Il faut donc une transformation qui garantisse de rester sur le support [0,1]. Une transformation qui respecte cette contrainte est la transformation :

$$f(p) = kp/(1 + (k-1)p) \tag{2}$$

Elle laisse bien à 1 ou 0 les probabilités égales à ces valeurs et, pour les individus tels que $0 < p < 1$ elle préserve les rapports des *odds-ratio*, autrement dit, si on a les probabilités de tirages p_i et p_j pour deux individus, alors on a :

$$[f(p_i)/(1 - f(p_i))]/[f(p_j)/(1 - f(p_j))] = [p_i/(1 - p_i)]/[p_j/(1 - p_j)]$$

La seule difficulté sera que la valeur de k nécessaire à l'alignement n'a pas d'expression analytique et doit donc être obtenue par résolution numérique. Il reste ensuite à effectuer le tirage avec ces nouvelles probabilités et à gérer les deux cas particuliers où cette procédure ne peut aboutir :

- Le premier cas est celui où le nombre d'individus à $p_i > 0$ est inférieur à la cible. Dans ce cas, on peut imaginer de tirer un nombre additionnel d'individus pour qui les probabilités d'inclusion *a priori* sont nulles, mais avec le risque de simuler des évènements totalement non pertinents. Dans la fonction d'alignement qui sera proposée plus loin, on préfèrera laisser apparente l'impossibilité de satisfaire la cible.

- L'autre cas est le cas symétrique où le nombre d'individus tels que $p_i = 1$ est supérieur à la cible. Dans ce cas, on pourrait choisir symétriquement de ne pas respecter la cible et de retourner l'ensemble des individus pour qui l'évènement est présumé certain. Le programme qui sera présenté plus loin fait

néanmoins le choix de ne retourner qu'une sélection aléatoire de K individus parmi les individus tels que $p_i = 1$. En pratique, ce cas est peu suspectible d'apparaître : s'il est fréquent de prédéfinir des probabilités d'occurence nulles pour un évènement -par exemple aucune naissance en dehors des âges biologiquement envisageables-, il est rare de spécifier des probabilités unitaires, où alors c'est qu'il s'agit d'évènements déterministes -par exemple le départ à la retraite à l'âge maximum légal- qui se simulent sans passer par des tirages probabilistes.

Il existe une autre méthode pour obtenir le même alignement sur une cible K différente de la somme des p_i, il s'agit de l'algorithme dit d'alignement par tri. Pour chaque individu i, il consiste à tirer un alea u_i uniforme entre 0 et 1 et à calculer la différence $v_i = f(u_i) - f(p_i)$ où f est la transformation logit $f(x) = ln(x/(1-x))$. On ordonne alors les individus selon les valeurs croissantes de v_i et on retient les K premiers. Quelles vont-être les propriétés de cette méthode ? Elle va se traduire par la définition d'un seuil de sélection S avec tirage des individus tels que $f(u_i) - f(p_i) < S$. Il n'est pas nécessaire d'avoir une expression explicite de S pour connaitre les propriétés des probabilités de tirage qui vont en résulter. La probabilité de tirage de i va être :

$$Prob\left[u_i < f^{-1}\left(f(p_i) + S\right)\right]$$

qui s'écrit encore :

$$Prob\left[u_i < \frac{e^S p_i/(1-p_i)}{1 + e^S p_i/(1-p_i)}\right] = \frac{e^S p_i}{1 + p_i(e^S - 1)}$$

ce qui correspond exactement à la transformation (2) avec $k = e^S$.

2.6 Réduction de variance et alignement : une fonction pour les tirages contraints

L'ensemble des modes de tirage qu'on vient de présenter ont été ici rassemblés dans une fonction unique `tirage` dont on va maintenant présenter le fonctionnement. Cette fonction suppose prédéfini le vecteur de probabilités individuelles de connaître l'événement considéré. Ce vecteur peut avoir été déterminé de différentes manières. Par exemple, pour la mortalité, les probabilités découleront dans le cas le plus simple de la table de mortalité par sexe et âge. Mais elles peuvent aussi provenir d'un modèle plus ou moins complexe, par exemple un modèle de transitions entre états faisant intervenir différents facteurs explicatifs, soit constants dans le temps (sexe, niveau d'éducation) ou variables et donc simulés en d'autres endroits du modèle (âge, nombre d'enfants, durée passée dans l'état courant et/ou dans certains autres états antérieurs...). Ce vecteur peut aussi comprendre des probabilités non déterminées (ayant la valeur `NA`) pour les individus hors champ pour l'événement considéré : ces probabilités seront traitées comme nulles.

Les arguments de la fonction sont ce vecteur et deux arguments optionnels `cible` et `algo`. Le paramètre `cible` peut prendre soit une valeur littérale soit une valeur numérique, non nécessairement entière et il définit de quelle manière on souhaite contraindre le tirage. La valeur par défaut est la valeur `cible="sommeprob"` signalant que le retour va être une liste d'identifiants tirés selon les probabilités passées en argument en nombre aussi proche que possible de la somme de ces probabilités. Le seul élément de variance qui subsistera dans la longueur de la liste tirée correspondra à l'arrondi aléatoire de cette somme de probabilités. Cet arrondi aléatoire évite le biais de sous-simulation de l'événement considéré qu'on aurait avec un arrondi à l'entier inférieur, biais qui sera d'autant plus important en termes relatifs qu'on simule un événement peu fréquent.

L'autre possibilité d'option littérale est l'option `cible="aucune"` qui conduit à un tirage poissonien non contraint, i.e. chaque individu est tiré librement en comparant sa probabilité individuelle de tirage à un nombre pseudo aléatoire compris entre 0 et 1. Sinon, si la valeur fournie à `cible` est une valeur numérique, on sera dans le cas d'une demande d'alignement. Le tirage sera de longueur égale à cette taille, toujours à un arrondi aléatoire près, avec l'ajustement requis sur le profil de probabilités individuelles, déformées à la hausse ou à la baisse selon que la cible est supérieure ou inférieure à la somme de ces probabilités de tirages. Le paramètre `algo` définit pour sa part la méthode utilisée pour l'alignement ou la réduction de variance, avec les deux modalités `"sidewalk"` et `"sorting"`, la seconde étant la modalité par défaut.

A titre d'illustration, on a mobilisé cette fonction pour le test suivant. On se donne une population de 10 000 individus, pour lesquels on prédéfinit un vecteur `proba` qui vaudra un pour 100 d'entre eux, qui aura une valeur aléatoire comprise entre 0 et 1 pour 900 autres, et qui sera nulle pour le reste de la population. La probabilité moyenne sur cette population est donc de l'ordre de 5,5%. On procède alors à des tirages successifs selon cette probabilité, sous trois modalités successives : un tirage poissonien, un tirage avec simple réduction de variance, et un tirage avec alignement sur une cible égale à la moitié de l'espérance du nombre d'évènements simulés. En cas de tirage non poissonien, la méthode est l'alignement par tri. Les trois appels sont donc :

```
liste <- tirage(proba,cible="aucune")                       # tirage non contraint
liste <- tirage(proba,cible="sommeprob",algo="sorting")     # tirage avec réduction de variance
liste <- tirage(proba,cible=0.5*sum(proba),algo="sorting") # tirage avec cible
```

Chaque tirage a été réédité 1000 fois pour calculer la distribution de la taille de l'échantillon tiré (partie gauche de la figure 5) et les probabilités d'inclusion *a posteriori* de chaque individu en fonction des probabilités *ex ante* données par le vecteur `proba` (partie droite).

Dans les deux premiers cas, les graphiques de droite montrent la correspondance entre probabilités d'inclusion *ex ante* et *ex post*, alors que celui du troisième cas montre la déformation imposée aux probabilités d'inclusion pour parvenir à un échantillon de la taille désirée, le graphique reproduisant la forme de la fonction de transformation (2) après évaluation de son paramètre k, qui est ici implicite puisqu'on a choisi la méthode d'alignement par tri. Des résultats totalement équivalents sont obtenus par l'option `"sidewalk"`. En fait, le principal élément qui peut différencier les deux méthodes est la durée d'exécution, qui est évidemment un critère important lorsque les calages interviennent de manière assez répétitive dans le modèle. Un problème de la méthode d'alignement par tri est qu'elle impose un tri de la population à risque d'être sélectionnée. L'autre algorithme impose pour sa part un tatônnement pour le calcul du paramètre k, mais celui-ci est assez rapide. Ce deuxième algorithme devrait donc être plus rapide, mais on a vu qu'il était préférable de le faire précéder d'un réarrangement aléatoire de la population traitée pour limiter les conséquences du fait que l'algorithme donne en général une probabilité très faible et souvent nulle de tirage joint de deux individus traités consécutivement, et c'est ce que fait la fonction `tirage`. Cette option implique donc elle aussi un tri et c'est cette modalité `"sidewalk"` qui s'avère finalement la plus coûteuse. Les deux sont néanmoins assez rapides : l'ensemble des 1 000 tirages avec alignement sur 10 000 individus prennent respectivement 0,9 et 1,3 seconde avec les deux options `"sorting"` et `"sidewalk"`[5].

[5]Il serait possible d'accélérer l'une et l'autre des deux méthodes. Le tri préalable dans la méthode du sidewalk est fait ici par la méthode naïve consistant à générer une variable aléatoire temporaire sur l'ensemble de la population et à trier selon cette variable (voir plus loin la fonction `permut`), or il existe des algorithmes plus rapides (algorithme de Fisher-Yates). De même, dans la méthode d'alignement par tri, repérer les K premiers individus triés selon $f(u_i) - f(p_i)$ ne nécessite pas un tri complet de la liste : il existe des algorithmes qui en tiennent compte. Néanmoins, implémenter ces algorithmes sous R tend plutôt à

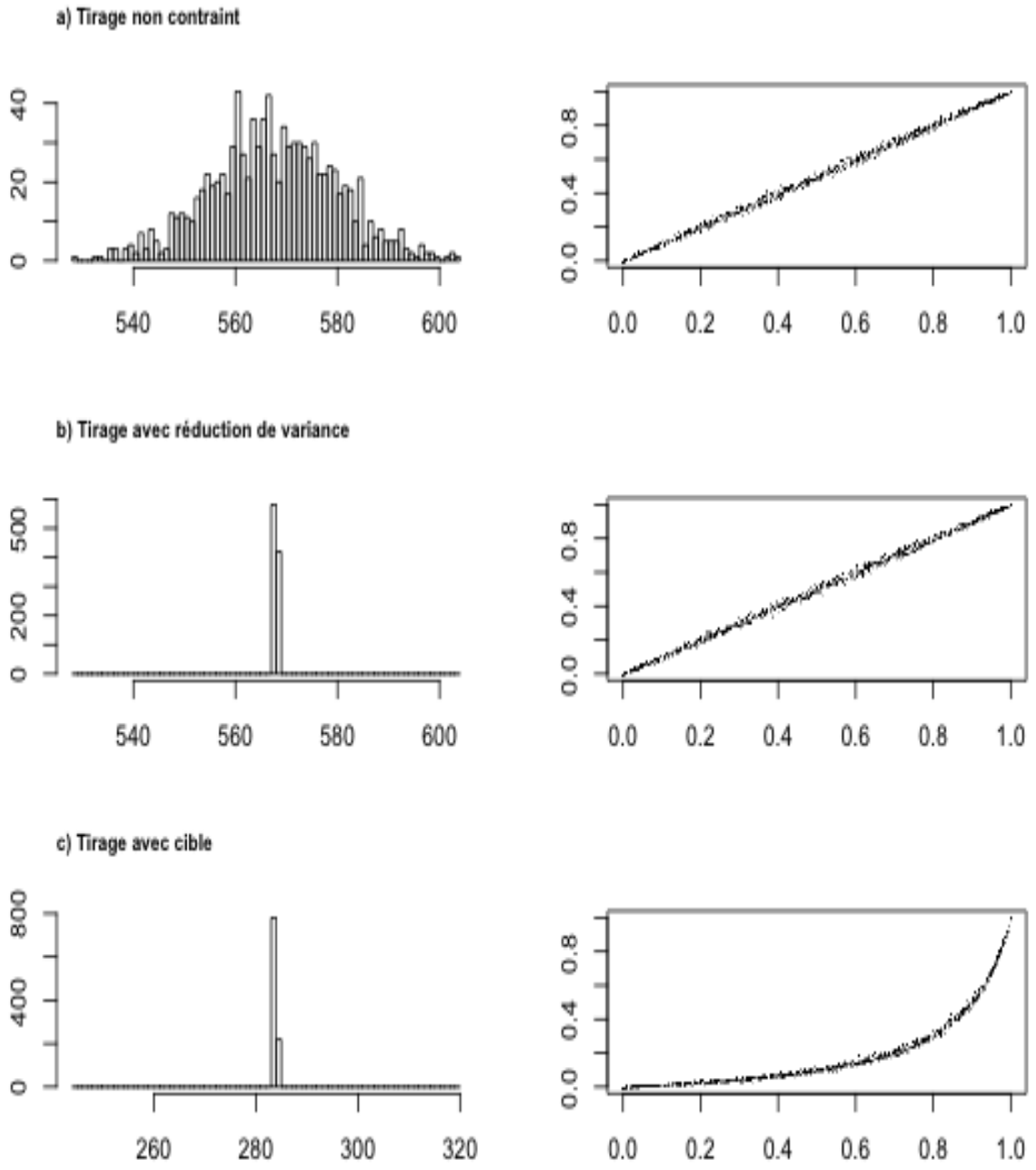


Figure 5: résultats des trois formes d'appel de la fonction `tirage` répliqués 1000 fois (graphiques de gauche : distribution du nombre d'évènements tirés, graphiques de droite : probabilités de tirage constatées *ex post* en fonction des probabilités individuelles *ex ante*)

On peut voir ce que donne la mise en œuvre de cette méthode de réduction de la variance dans notre exemple de simulation des flux de transition emploi-chômage. Le programme ci-dessous reprend le programme `Exemple_3.R` en introduisant le tirage avec réduction de variance sous l'option par défaut `"sorting"`.

```
# Programme Exemple_5.R
setwd("~/Documents/Microsimulation R/Outils")
source("OutilsMS.R")
statut <- matrix(nrow=1000,ncol=25)
txcho  <- numeric(25)

# génération du statut initial
proba <- numeric(1000)
proba <- rep(0.1,1000)
statut[,1] <- rep(1,1000)
statut[tirage(proba)1] <- 2
proba_sortie_chomage <- 0.5
proba_entree_chomage <- 0.025

for (t in 1:25)
{
  txcho[t] <- sum(statut[,t]==2)/10
  if (t<25)
  {
    statut[,t+1] <- statut[,t]
    statut[tirage((statut[,t]==1)*proba_entree_chomage),t+1] <- 2
    statut[tirage((statut[,t]==2)*proba_sortie_chomage),t+1] <- 1
  }
}

duree_cho <- numeric(1000)
for (i in 1:1000)
{
  duree_cho[i] <- sum(statut[i,]==2)
}
```

On voit que, dans cet exemple, il n'a même pas été nécessaire de générer explicitement les vecteurs de probabilités passés en argument à la fonction `tirage`. Un tel vecteur est généré de manière implicite par la séquence `(statut[,t]==1)*proba_entree_chomage` qui retourne un vecteur donnant la probabilité d'entrée au chômage pour l'ensemble des individus dont le statut en `t` est égal à 1 et qui sera à la modalité `NA` pour les autres, modalité qui est traitée comme une probabilité nulle par la fonction `tirage`.

Le résultat de ce programme est donné par la figure 6. Cette méthode de réduction de variance conduit évidemment à un lissage quasi-total du taux de chômage et la courbe reproduit parfaitement le profil théorique de (1), aux effets d'arrondi près. Évidemment, sur un exemple aussi simple, le résultat ainsi atteint en matière de taux de chômage n'apporte rien par rapport à ce qu'aurait donné une simulation purement déterministe du même modèle. Mais, même sur cet exemple simple, la microsimulation conserve l'avantage de fournir

dégrader la performance : la raison est à nouveau la lenteur des boucles sous R. Ils ne peuvent faire gagner du temps que si on les programme de la façon dont est programmée l'algorithme de tri par défaut de R, c'est-à-dire en langage C. R permet ce genre d'incorporation de code étranger qui peut être nécessaire pour des modèles nécessitant une grande rapidité d'éxécution, mais ceci dépasse le cadre de cette note.

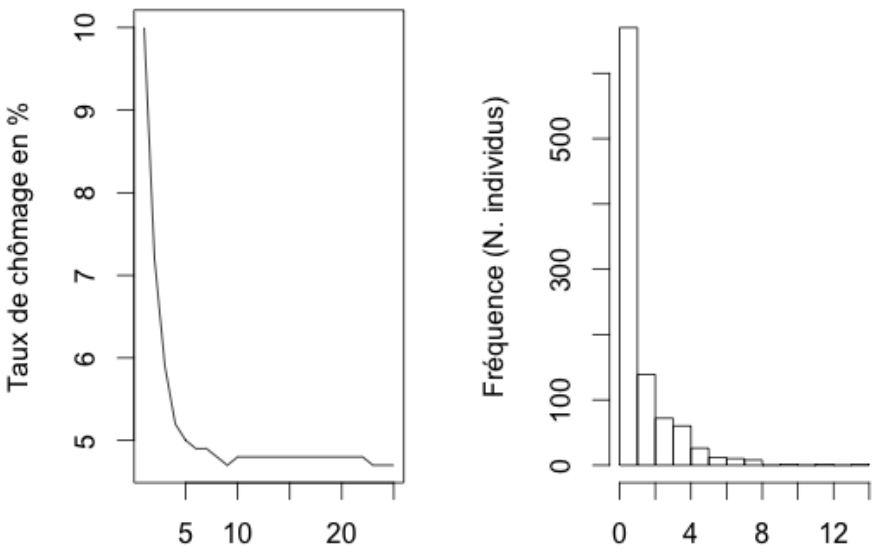


Figure 6: Résultats du programme `Exemple_5.R` : taux de chômage en fonction du temps (à gauche) et distribution des durées passées en chômage (à droite)

d'autres résultats, tels que la distribution des durées passées en chômage. Elle est donnée sur la partie droite de la figure, qui présente les mêmes caractéristiques que la distribution estimée sans calage qui était donnée sur la figure 3. Elle offre aussi la possibilité de différencier au niveau individuel les degrés d'exposition aux différents types d'évènements ou de servir de base à d'autres calculs nécessitant l'appui sur données micro. Sur cet exemple très simple, il peut par exemple s'agir de calculs de prestations chômage conditonnées par la durée du chômage. On verra plus loin ce type d'apport, mais avant cela, il est utile d'introduire quelques compléments sur l'utilisation des fonctions, à la fois leur regroupement dans des fichiers partageables par plusieurs programmes, et la possibilité, pour une fonction, de modifier directement les variables individuelles qui font l'objet de la simulation sans en faire des valeurs de retour de cette fonction.

2.7 Regrouper les fonctions de base dans un fichier d'intérêt général

La fonction `tirage` qui vient d'être présentée jouera un rôle central dans la plupart des programmes qui suivront. Tel qu'il est envisagé dans ce document, un modèle de microsimulation dynamique consiste pour l'essentiel, à chaque date de la simulation, à réaliser un ensemble de tirages de sous-populations aux membres desquels sont ensuite appliqués différents types de traitements. Mais d'autres opérations sur ces listes d'individus peuvent s'avérer nécessaires : tri, sélection de sous-listes... S'agissant d'opérations potentiellement communes à un grand nombre de modèles, il est intéressant de les regrouper dans un fichier unique pouvant être partagé par des programmes différents.

Pour éviter la construction de programmes trop longs et faciliter le partage de telles fonctions entre programmes, on les regroupe dans des fichiers sources séparés qui peuvent aussi servir à la déclaration et à l'initialisation de variables également d'intérêt général. La suite des applications présentées dans cette note utiliseront ainsi les programmes d'un fichier `OutilsMS.R` qui, outre la fonctions `tirage`, incluera les

définitions de variables et fonctions suivantes :

- `taux_sondage` : une grandeur scalaire qui est le taux de sondage de la population simulée (initialisé à 1 par défaut)

- `arr_alea` : une fonction reproduisant sur un scalaire quelconque le type d'arrondi aléatoire qu'on a vu être généré spontanément par la fonction tirage. Par exemple :

```
arr_alea(10.2) # retourne 10 dans 80% des cas et 11 dans les 20% restant
```

- `reduc` : une fonction convertissant un effectif de la population réelle en un nombre d'individus de la population simulée. Cette fonction est notamment utile pour des tirages avec cible lorsqu'on veut mettre à la bonne échelle des cibles exprimées en effectifs de la population réelle totale, par exemple :

```
cible_deces_a_simuler <- reduc(deces_totaux)
```

Cette fonction se limite en fait à une multiplication par `taux_sondage` mais en se chargeant automatiquement de l'arrondi aléatoire.

- `augm` : une fonction effectuant l'opération inverse, utilisant l'inverse du taux de sondage pour convertir les résultats de comptages en effectifs réels. Par exemple, si une valeur positive de la variable `statut` permet de sélectionner les individus présents dans la population à un instant donné, un chiffre de la population totale sera donné par :

```
poptot <- augm(sum(statut>0))
```

- `tri` : la fonction triant une liste d'individus selon les valeurs croissantes ou décroissantes d'une variable[6], avec pour exemples d'appel :

```
liste <- tri(liste, age) # tri par age croissant des membres
liste <- tri(liste, age, type = "decroissant") # tri par age décroissant
```

- `permut` : une fonction permutant une liste d'identifiants dans un ordre aléatoire[7], soit tout simplement :

```
liste <- permut(liste)
```

- `extr` : une fonction retournant un extrait d'une liste, selon plusieurs modalités :

```
liste <- extr(liste,n) # extraction des n premiers éléments
liste <- extr(liste,-n) # exclusion des n premiers éléments
liste <- extr(liste,keep=age>20) # conservation des i tels que age[i]>20
liste <- extr(liste,drop=sexe==1 & age>20) # exclusion hommes > 20 ans
```

Par exemple, l'application de proche en proche d'un traitement aux différents membres d'une liste pourra prendre la forme :

[6] Cette fonction est différente de la fonction prédéfinie `sort`, qui se contenterait de reclasser les éléments de la liste selon leurs propres valeurs plutôt que selon les valeurs d'une variable indicée par les éléments de cette liste. Par exemple, si x est le vecteur (1,3,2), `sort(x)` retourne le vecteur (1,2,3).

[7] Comme indiqué plus haut en 1.3, un appel à cette fonction `permut` est inclus dans `tirage` pour éviter tout risque de biais de l'algorithme de tirage séquentiel utilisé par cette fonction.

```
liste <- permut(liste)
while(length(liste>0))
{
   i <- liste[1]
   # actions sur individu i ...
   extr(liste,-1)
}
```

où l'appel préalable de `permut` garantit que le traitement de la liste se fera dans un ordre aléatoire.

Pour pouvoir utiliser ces fonctions sous l'environnement RStudio décrit en annexe 3, il suffit soit de provoquer une exécution du fichier `OutilsMS.R` qui chargera les objets décrits dans ce fichier dans l'espace de travail et les rendra disponibles depuis n'importe quel autre fichier de programme .R ou depuis la fenêtre « console ». En particulier les fonctions ainsi chargées peuvent être appelées interactivement depuis cette fenêtre pour en tester le fonctionnement sur des données quelconques, par exemple en phase de déboguage du programme de simulation. Alternativement, le chargement de `OutilsMS.R` peut être forcé depuis le programme de simulation principal par une instruction

```
setwd("~/Documents/Microsimulation R/Outils")
source("OutilsMS.R")
```

servant à la fois à définir le répertoire de travail dans lequel aller chercher le fichier et déclenchant son chargement, comme ceci était fait plus haut en tête du programme `Exemple_5.R` [8].

2.8 Vers un modèle plus complet (a) : une seconde bibliothèque d'utilitaires

Muni de cette première bibliothèque, on va maintenant évoluer vers un modèle qui restera encore très simple mais qui donnera une idée plus complète des ingrédients d'un modèle en vraie grandeur, en mobilisant également une seconde bibliothèque d'outils plus spécifiques au problème considéré. Comme les précédents, ce modèle simulera les transitions sur le marché du travail entre deux statuts, l'emploi et le chômage, avec des pertes d'emploi survenant aléatoirement selon une probabilité exogène `proba_sortie_emploi` mais avec les différences suivantes :

- Jusqu'ici, la structure démographique de la population simulée était totalement ignorée. Les individus présents dans l'échantillon l'étaient de manière perpétuelle, sans début ni fin de vie active. On va désormais supposer que les individus de l'échantillon n'y sont présents que de leur entrée en activité à leur départ en retraite, qu'on supposera se faire pour simplifier à respectivement 20 et 60 ans. Comme le modèle n'a pas vocation à simuler autre chose qu'une population stationnaire, ceci pourra être simulé en gérant une variable individuelle `age`, et en rétinitialisant les caractéristiques de l'individu `i` aux valeurs d'entrée dans la vie active à chaque fois que `age[i]` atteindra 60 ans.

- Contrairement aux pertes d'emploi qui continueront à se faire à probabilité fixe, les retours à l'emploi se feront de manière à satisfaire une cible d'emploi déterminée de manière macroéconomique. Dans ce modèle très simple, cette cible d'emploi sera constante et exogène, mais on imaginera facilement comment elle pourrait éventuellement être endogénéisée en fonction des autres variables générées par le

[8] Pour des biblothèques d'outils plus complètes et destinées à être partagées plus largement, R offre aussi la possibilité de constituer des packages, dont on trouve de nombreux exemples sur le site de la communauté R (`www.cran.org`) mais dont la création est un peu plus complexe, puisqu'elle suppose notamment la rédaction d'une documentation selon un format spécifique. La création de packages ne sera pas abordée dans cette note.

modèle : par exemple en la faisant dépendre du taux de profitabilité moyen sur les emplois existants, ou de politiques affectant la demande. Les employeurs effectueront chaque année les embauches requises pour satisfaire cette cible d'emploi. Ce que fera le modèle consistera surtout à déterminer qui seront les bénéficiaires de ces recrutements au sein du stock de personnes sans emploi. L'idée sera de supposer une employabilité qui décroit avec l'ancienneté du chômage : les demandeurs d'emploi seront classés selon une variable d'employabilité corrélée négativement à cette ancienneté, avec une intensité qui fera l'objet d'un calibrage.

- Enfin, même si cet apport sera ici symbolique, le modèle gérera une variable `salaire` et une variable `alloc` donnant respectivement le salaire et le taux d'allocation chômage pour, respectivement, les individus en emploi et les individus au chômage.

Pour ce modèle, on construit une seconde bibliothèque de fonctions, complémentaire à la bibliothèque `OutilsMS.R`, mais spécifique à ce modèle. Elle inclut à la fois des définitions de variables et de fonctions. Les variables sont soit des variables individuelles "à mémoire" de type individu × date (`statut`, `salaire` et `alloc`), soit des variables individuelles temporaires ne dépendant que de `i` (`age` et `anciennete`), soit enfin des paramètres scalaires : il n'y aura ici qu'un seul paramètre de cette dernière catégorie, qui est un niveau de salaire minimum. Les fonctions peuvent être elles aussi regroupées en plusieurs catégories représentatives des types de fonctions à envisager pour des programmes en vraie grandeur :

- Une fonction initialise les valeurs individuelles. Ceci est fait dans le programme et non en amont du programme puisqu'on est ici dans un cas de population fictive. Cette fonction `initialisation` ne peut prétendre donner à la population simulée une structure de régime permanent. C'est plutôt au modèle d'évaluer vers quel type de régime permanent conduit le maintien indéfini des valeurs de ses paramètres, avant de choquer ces paramètres et de simuler l'impact transitoire et de long terme de leurs changements. Mais, pour réduire le délai de convergence du système vers cet état d'équilibre de départ, on essaye évidemment d'initialiser ces caractéristiques individuelles à des valeurs proches de celles auxquelles on s'attend pour ce régime permanent.

- Une fonction modifiant globalement l'ensemble des caractéristiques individuelles entre la date t-1 et la date t, hormis ce qui découlera des transitions entre état. Il s'agit de la fonction `vieillissement`. Elle incrémente l'âge et l'ancienneté dans le statut courant, elle fait évoluer le salaire selon une règle sommaire de progression à l'ancienneté et l'allocation chômage selon une règle symétrique, tout aussi sommaire, de dégressivité. La seconde partie de cette fonction gère également le remplacement de chaque individu partant en retraite par un individu héritant de son identifiant et dont les caractéristiques sont réinitialisées à des caractéristiques de début de carrière.

- Une fonction auxiliaire opérant au niveau individu × date qui, elle, ne modifie aucune caractéristique des individus, mais retourne un indicateur dérivé de leurs caractéristiques : il s'agit de la fonction `salaire_reference` qui retourne le dernier salaire de la dernière période d'emploi si l'individu en a déjà connu une, ou le salaire minimum. Cette information est utilisée pour déterminer soit l'allocation-chômage en cas de licenciement, soit le premier salaire dans le nouveau poste, en cas d'embauche ou de réembauche. Par exemple, en cas de réembauche, on suppose ici une décote forfaitaire de 20% par rapport au dernier salaire dans l'emploi antérieur, qui pourrait évidemment être spécifiée plus finement dans un modèle plus réaliste.

- Enfin deux fonctions travaillant elles aussi au niveau individu × date. Il s'agit respectivement des fonctions `entree_chomage` et `entree_emploi` qui simulent les conséquences d'une sortie d'emploi ou d'une embauche, avec modification du statut, remise à zéro du compteur d'ancienneté dans le statut, et mise à jour des variables `salaire` et `alloc`.

Le contenu complet de cette bibliothèque est détaillé ci-dessous.

```
setwd("~/Documents/Microsimulation R/Outils")
source("OutilsMS.R")

taux_sondage          <- 1/1000
taille                <- 1000
horizon               <- 200
age                   <- numeric(taille)
anciennete            <- numeric(taille)
statut                <- matrix(nrow=taille,ncol=horizon)
salaire               <- matrix(nrow=taille,ncol=horizon)
alloc                 <- matrix(nrow=taille,ncol=horizon)
salaire_minimum       <- 1000

initialisation <- function()
{
   age               <<- floor(runif(taille,min=20,max=60))
   statut[,1]        <<- rmult(taille,c(1,2),c(0.95,0.05))
   salaire[,1]       <<- (statut[,1]==1)*salaire_minimum
   alloc[,1]         <<- (statut[,1]==2)*salaire_minimum*0.7
   liste              <- which(statut[,1]==1)
   alloc[liste]      <<- 0
   salaire[liste]    <<- salaire_minimum
   anciennete[liste] <<- age[liste]-20
   liste              <- which(statut[,1]==2)
   alloc[liste]      <<- salaire_minimum*0.7
   salaire[liste]    <<- 0
   anciennete[liste] <<- min(age[liste]-20,1)
}

vieillissement  <- function(t)
{
  # Incrementation compteurs et évolutions des revenus
  age           <<- age+1
  anciennete    <<- anciennete+1
  statut[,t]    <<- statut[,t-1]
  salaire[,t]   <<- salaire[,t-1]*(1+0.05*exp(-(age-20)))
  alloc[,t]     <<- alloc[,t-1]*0.9

  # Si age de la retraite, remplacement par un nouvel individu initialement
  # classé comme demandeur d'emploi
  liste <- which(age==60)
  age[liste]        <<- 20
  anciennete[liste] <<- 0
  statut[liste,t]   <<- 2
  salaire[liste,t]  <<- 0
  alloc[liste,t]    <<- 0
```

```
}

salaire_reference <- function(i,t)
{
  annees_emploi <- which(statut[i,(max((t-age[i]+20),1):t)]==1)
  if (length(annees_emploi)==0) {return (salaire_minimum)}
  else                          {return (salaire[max(annees_emploi)])}
}

entree_chomage <- function(i,t)
{
  statut[i,t]     <<- 2
  salaire[i,t]    <<- 0
  alloc[i,t]      <<- 0.7*salaire_reference(i,t)
  anciennete[i]   <<- 0
}

entree_emploi <- function(i,t)
{
  statut[i,t]     <<- 1
  salaire[i,t]    <<- 0.8*salaire_reference(i,t)
  alloc[i,t]      <<- 0
  anciennete[i]   <<- 0
}
```

On notera que la plupart de ces fonctions ont la caractéristique de travailler directement sur les variables individuelles définies par la bibliothèque, grâce à l'instruction dite de superaffectation `<<-`, variante de l'instruction d'affectation simple `<-`, dont le principe est décrit plus en détail en annexe 2. Cette superaffectation `<<-` permet notamment aux deux fonctions `entree_chomage` et `entree_emploi` de modifier directement les différentes variables microéconomiques sans utiliser l'instruction `return`. On notera par ailleurs que ces deux fonctions ont `i` et `t` comme argument d'appel. Ce choix peut effectivement se discuter. On aurait pu imaginer de se dispenser de ce passage d'argument, ce qui aurait permis des séquences du type

```
for (t in 1:horizon)
{
  for (i in liste_entrees_chomage)
  {
    entree_chomage
  }
}
```

la fonction `entrée_chomage` s'appuyant directement sur les valeurs courantes des indices de boucle `i` et `t` sans qu'il y ait besoin de les lui transmettre. On a préféré ici forcer cette transmission, pour prévoir les cas où l'utilisateur souhaiterait utiliser d'autres noms d'indices pour ses boucles individuelles et temporelles, ou pour que ces fonctions soient utilisables dans d'autres contextes.

Le fichier réunissant l'ensemble de ces éléments est le fichier `OutilsModeleMdT.R` qui va alors pouvoir être utilisé conjointement au fichier `OutilsMS.R` par le programme de simulation proprement dit.

2.9 Vers un modèle plus complet (b) : programme appelant et calibrage

Sur ces bases, le programme de microsimulation proprement dit est celui qui est donné dans le pavé suivant :

```
# Programme Exemple_6.R
setwd("~/Documents/Microsimulation R/Outils")
source("OutilsMS.R")
source("OutilsModeleMdT.R")

# Variables de comptage
txcho       <- numeric(horizon)
anc_cho     <- numeric(horizon)
anc_emp     <- numeric(horizon)
sal_moy     <- numeric(horizon)
alloc_moy   <- numeric(horizon)

# Programme de simulation
simul <- function(proba_entree_chomage,alpha,cible_emploi)
{
  for (t in 1:horizon)
  {
    if (t==1) {initialisation()}
    else      {vieillissement(t)}

    # simulation des ruptures d'emploi
    liste <- tirage((statut[,t]==1)*proba_entree_chomage)
    for (i in liste) {entree_chomage(i,t)}

    # Simulation des nouvelles embauches
    embauches <- 0.5*(reduc(cible_emploi)-sum(statut[,t]==1))
    employabilite <- alpha*anciennete +rnorm(taille)
    liste <- which(statut[,t]==2)
    liste <- tri(liste,employabilite,type="decroissant")
    liste <- extr(liste,embauches)
    for (i in liste) {entree_emploi(i,t)}

    txcho[t]     <<- 100*sum(statut[,t]==2)/taille
    anc_cho[t]   <<- mean(anciennete[which(statut[,t]==2)])
    anc_emp[t]   <<- mean(anciennete[which(statut[,t]==1)])
    sal_moy[t]   <<- mean(salaire[statut[,t]==1,t])
    alloc_moy[t] <<- mean(alloc[statut[,t]==2,t])
  }
}
```

Grâce aux fonctions préprogrammées dans `OutilsModeleMdT.R`, ce programme se présente de manière assez compacte permettant de bien visualiser son organisation. Quelques variables globales d'output sont tout d'abord définies. Il s'agira de moyennes évaluées à chaque date de la simulation : le taux de chômage, les anciennetés moyennes en emploi et dans le chômage, le salaire moyen et l'allocation chômage moyenne. Puis, comme dans `Exemple_4.R`, le programme de simulation proprement dit se présente sous forme d'une fonction `simul`, à trois arguments : la probabilité de séparation, le paramètre `alpha` qui va gouverner l'effet de l'ancienneté dans le chômage sur les chances de retour à l'emploi, et enfin la cible exogène d'emploi, `cible_emploi`. La raison de cette écriture sous forme de fonction est qu'elle va faciliter le calibrage des

deux premiers de ces paramètres supposés non directement observables. Dans le cas de la probabilité de séparation, cette hypothèse est un peu artificielle : dans la réalité, on dispose de statistiques sur les taux de séparation qui devraient permettre de se dispenser de ce calibrage. En revanche, l'absence d'information *a priori* est bien ce à quoi on serait confronté dans la réalité pour le paramètre `alpha`. Ce paramètre intervient dans le calcul de la variable `employabilite`. Elle permet de classer les demandeurs d'emploi par ordre de priorité pour les embauches. Seuls les premiers chômeurs reviendront à l'emploi, après tri de l'ensemble de ces chômeurs selon les valeurs décroissantes de cette variable. Dans le cas particulier `alpha`=0, la variable `employabilite` est une variable aléatoire normale, ce qui signifie une sélection totalement aléatoire de ces chômeurs. Ensuite, plus `alpha` devient grand, plus l'ancienneté du chômage pèse négativement sur l'employabilité, ce qui doit conduire à une polarisation entre des chômeurs de courte durée qui reviennent facilement à l'emploi et des chômeurs de longue durée qui tendent à y rester.

Le calibrage des deux paramètres `proba_entree_chomage` et `alpha` va se faire selon le type de principe présenté en section 1.5, en cherchant à optimiser l'adéquation du modèle à un jeu de caractéristiques macro. On aurait pu introduire le taux de chômage parmi ces cibles macro à respecter mais ceci est superflu puisqu'il est contraint mécaniquement par le paramètre `cible_emploi`. Les paramètres qu'on va chercher à ajuster au mieux seront plutôt les anciennetés moyennes dans l'emploi et dans le chômage, qui sont deux caractéristiques résumant bien le turnover auquel on assiste sur le marché du travail. Une valeur élevée pour `alpha` est supposée augmenter l'ancienneté moyenne dans le chômage. Une valeur élevée de `proba_entree_chomage` a pour sa part deux effets : elle réduit l'ancienneté moyenne en emploi, et elle réduit aussi l'ancienneté moyenne des chômeurs, puisque beaucoup de séparations à emploi global donné conduisent à de forts flux de chomeurs de courte durée à qui un plus grand nombre d'embauches donne plus de chances de retrouver un emploi rapidement. Si on avait conservé un modèle avec agents à durée de vie infinie, ce paramètre n'aurait pas eu besoin de calibrage : on aurait pu l'estimer directement comme inverse de l'ancienneté moyenne en emploi[9]. Mais les durées en emploi sont ici tronquées par les départs en retraite et cette relation ne s'applique pas, d'où l'utilité du calibrage.

Pour ce calibrage, l'adéquation entre valeurs simulées et valeurs cibles des deux anciennetés moyennes ne doit évidemment s'apprécier qu'une fois que le modèle a suffisamment itéré pour avoir rejoint son état d'équilibre dynamique stochastique : par équilibre dynamique stochastique, on entend un régime dans lequel aucun individu pris isolément n'est en équilibre -les positions individuelles évoluent constamment- mais où les distributions des caractéristiques individuelles sont stables. Sur un modèle avec renouvellement démographique et dans lequel les situations initiales sont tirées de manière partiellement arbitraire, la convergence peut prendre un certain temps. On ne calcule donc les anciennetés moyennes en emploi et dans le chômage qu'au bout de plusieurs dizaines de périodes -en pratique sur les 50 dernières années d'une simulation poussée jusqu'en t=200-. Cet artifice vise uniquement à s'assurer qu'on a bien convergé vers le régime permanent, que l'utilisateur du modèle pourra ensuite perturber en modifiant les paramètres de la simulation. C'est sur ce régime permanent qu'on calcule ensuite un score d'ajustement qui sera ici la somme quadratique des écarts relatifs entre ces moyennes et des valeurs cibles « observées » fixées respectivement à 10 et 0,66 (soit huit

[9]Plus précisément, pour des séparations arrivant avec la probabilité instantanée λ, la probabilité d'être encore en emploi à l'ancienneté x est en $e^{-\lambda x}$ et l'ancienneté moyenne du stock est donc $\int_0^\infty x\mathrm{e}^{-\lambda x}\mathrm{d}x / \int_0^\infty \mathrm{e}^{-\lambda x}\mathrm{d}x = 1/\lambda$. Elle se trouve être égale à l'espérance de la durée passée en emploi $\int_0^\infty x\lambda\mathrm{e}^{-\lambda x}\mathrm{d}x$, mais cette dernière relation n'a rien de général. Si $e(d)$ et $V(d)$ sont l'espérance et la variance de la durée de séjour, l'ancienneté moyenne du stock est $e(d)/2 + V(d)/2e(d)$. Elle est égale à la moitié de la durée moyenne de séjour lorsque les départs sont concentrés en un point unique.

mois)[10]. Ce score est l'équivalent de la fonction $d[S(D_i^{sim}), S(D_i^{obs})]$ de la section 1.5.

Un code R permettant la minimisation de ce score par balayage est le suivant :

```
grille_proba <- seq(0.02,0.1,by=.01)
grille_alpha <- seq(0,2,by=.2)
anc_cho_moy  <- matrix(nrow=length(grille_proba),ncol=length(grille_alpha))
anc_emp_moy  <- matrix(nrow=length(grille_proba),ncol=length(grille_alpha))
score        <- matrix(nrow=length(grille_proba),ncol=length(grille_alpha))

for (i_p in (1:length(grille_proba)))
{
   for (i_a in (1:length(grille_alpha)))
   {
       simul(grille_proba[i_p],grille_alpha[i_a],950000)
       anc_cho_moy[i_p,i_a] <- mean(anc_cho[horizon-50:horizon])
       anc_emp_moy[i_p,i_a] <- mean(anc_emp[horizon-50:horizon])
       score[i_p,i_a] <- (anc_cho_moy[i_p,i_a]/0.66-1)**2+
                         (anc_emp_moy[i_p,i_a]/10  -1)**2
   }
}
```

Ce code définit deux vecteurs de valeurs de balayage pour `proba_entree_chomage` et `alpha`, procède à la simulation pour tous les couples de valeurs correspondants et calcule les anciennetés moyennes en emploi et en chômage qui en découlent, ainsi que le score d'ajustement correspondant.

Le résultat de ce balayage peut ensuite être inspecté à l'aide des instructions `contour` et `persp` dont les syntaxes détaillées sont fournies en annexe 3. La figure donnant le score en courbes de niveau montre que l'adéquation aux valeurs cibles est la meilleure pour une combinaison `proba_entree_chomage` $\approx$ 0,06 et `alpha` $\approx$ 0,6. Les graphiques en 3D illustrent le lien positif entre `alpha` et la durée du chômage et les effets négatifs de `proba_entree_chomage` à la fois sur l'ancienneté moyenne en emploi et dans la chômage. On vérifie que la valeur optimale de `proba_entree_chomage` n'est pas l'inverse de l'ancienneté moyenne en emploi sur laquelle on a cherché à s'aligner. Du fait de la troncation en fin de carrière, il faut un taux de séparation plus faible que 1/10 pour rendre compte d'une ancienneté moyenne en emploi de 10 années : c'est un des apports de la microsimulation avec structure démographique de prendre en compte ce facteur.

Une fois obtenues ces valeurs préférées des deux paramètres, il suffit de les utiliser pour relancer une nouvelle fois la simulation pour en examiner les autres résultats. On peut ensuite passer à la simulation de variantes portant sur ces paramètres ou d'autres éléments du modèle. Ceci peut éventuellement se faire sous forme interactive, l'environnement `R Studio` décrit en annexe 3 permettant de lancer en ligne des exécutions de `simul` et d'en inspecter directement les résultats de diverses manières : examen de résultats individuels, génération de nouveaux agrégats, sorties graphiques. Certains types de variantes peuvent d'ailleurs être produites sans relancer l'intégralité de la simulation. Par exemple, si on accepte une hypothèse d'absence des effets de l'indemnisation du chômage sur sa durée, rien n'interdit de recalculer individuellement la variable individuelle `alloc` pour un autre barème d'indemnisation, par exemple avec une autre formule de dégressivité ou encore un niveau d'indemnisation initial dépendant de la durée de la dernière séquence d'emploi ou du

[10]Dans ce modèle à pas annuel, la notion de durée est évidemment très approximative. Une durée moyenne inférieure à l'unité nécessite la présente d'anciennetés nulles et ces anciennetés nulles correspondent ici à des personnes au chômage en début de période t après y être tombées à la période t-1, dont l'ancienneté réelle est en fait positive. Les durées passées en chômage sont mesurées de façon plus satisfaisante dans un modèle à pas infra-annuel.

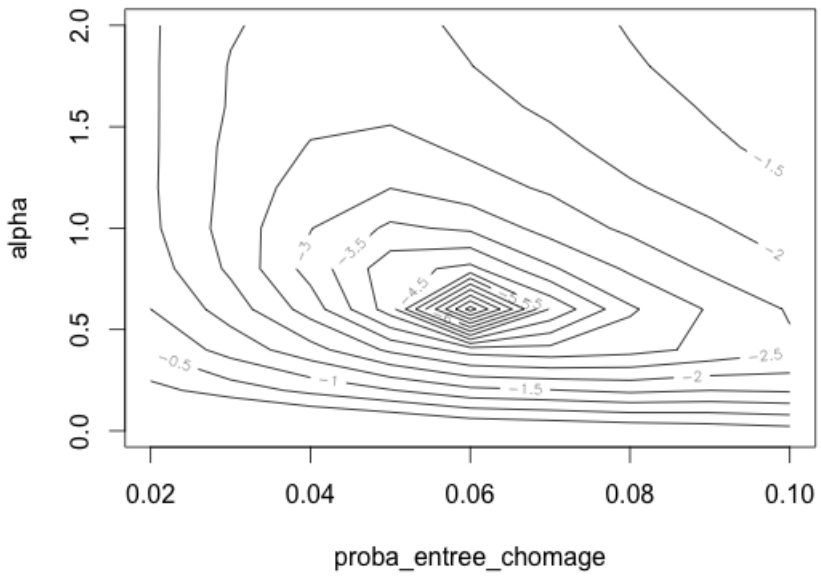


Figure 7: Calibrage des paramètres `alpha` et `proba_entree_chomage`.

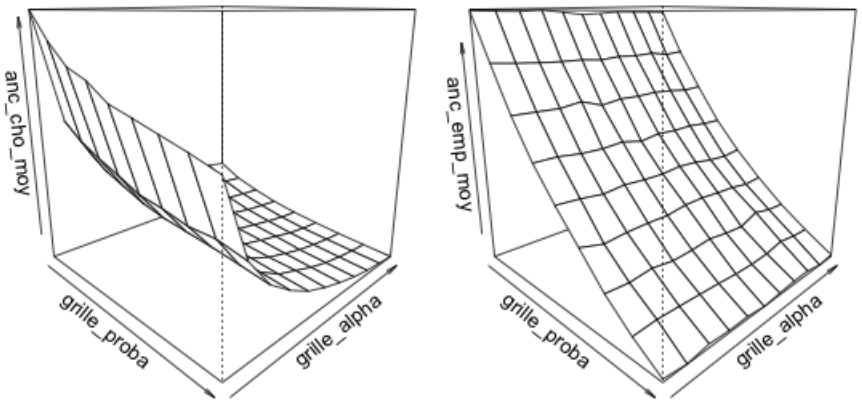


Figure 8: Impact des paramètres `proba_entree_chomage` et `alpha` sur les anciennetés moyennes dans le chômage et en emploi

salaire atteint avant l'entrée en chômage : tout ceci est possible dès lors qu'on a pris soin de stocker l'ensemble des historiques de statut et de salaire dans des variables "à mémoire".

Sur toutes ces bases, on voit assez bien de quelle manière le modèle peut être enrichi, sans remise en cause de sa structure globale, par exemple selon les lignes qu'on trouvera dans Barlet *et al.* (2009) ou Benoteau (2011). Diverses pistes sont les suivantes :

- La possibilité de faire varier la taille de la population active soit en allongeant ses bornes, soit en augmentant les flux d'entrée. Ceci suppose évidemment de renoncer au système de partage d'identifiants entre individus successifs, mais le coût est uniquement en termes d'espace de stockage, sans modification radicale de l'organisation du programme

- La gestion de variables additionnelles, à savoir un niveau de qualification associé à chaque individu et une différentiation des types d'emplois selon la qualification demandée.

- La gestion d'une variable décrivant la productivité individuelle et d'une variable donnant le coût global du travail de chaque salarié, incluant un calcul des charges sociales selon barème. On en dérive le taux de marge réalisé par l'employeur sur chaque poste. Ces éléments permettent d'endogénéiser la dynamique salariale : les salaires individuels croissent en fonction de la productivité individuelle, selon un mécanisme d'ajustement progressif qui se ralentit au fur et à mesure que la marge laissée à l'employeur se réduit. L'intensité de cet ajustement est également modulée par une variable d'environnement macro-économique reconstituée à partir du modèle, qui est le taux de chômage global. L'introduction de cet effet Phillips illustre la possibilité de combiner déterminants micro ou macro des évènements individuels : dans un modèle de microsimulation, les comportements micro peuvent très facilement prendre en compte l'évolution de variables macro. Il suffit de demander au modèle de les calculer à chaque date et de les injecter dans les équations déterminant les évolutions individuelles.

- Les dynamiques relatives de la productivité et du coût du travail permettent aussi de rajouter un facteur supplémentaire de rupture des périodes d'emploi, lorsque des chocs de productivité font baisser le taux de marge au-dessous d'un seuil critique. Le modèle combine ainsi un flux de séparations intervenant de manière exogène, comme dans le modèle qu'on vient d'examiner, et un flux de séparations économiques endogènes découlant des chocs de productivité.

- Un modèle plus complet peut aussi inclure une simulation plus riche de la demande globale de travail. Le nombre d'embauches proposées ne viserait plus à combler l'écart à un emploi cible parfaitement constant comme cela a été le cas ici : on peut poser que cet emploi cible est lui aussi influencé par des variables d'environnement macro-économique telles que la profitabilité moyenne constatée les années antérieures ou une variable de tensions sur la demande de biens. Il est également possible que l'ensemble des postes offerts ne soient pas immédiatement pourvus, ce qui permet au modèle de simuler également une dynamique des emplois vacants.

Une fois ceci fait, le modèle peut être utilisé pour la simulation de diverses variantes de politique économique (allègements de charges, salaire minimum...) dont on peut reconstituer les impacts sur un grand nombre de variables tels que des taux de chômage par qualification, des distributions détaillées des salaires.... Ces évaluations qui peuvent soit s'additionner soit être confrontées aux divers résultats d'évaluation *ex post* dont on dispose sur le même sujet, selon la stratégie évoquée en fin de la section 1.1.

Ce n'est toutefois pas selon cette direction qu'on va poursuivre ici. La section 3 va présenter un modèle en quasi-vraie grandeur mais de type plus purement démographique, puisqu'il va s'agir d'une réécriture, en R, d'une partie du bloc démographique du modèle de projection des retraites Destinie.

3 Un exemple plus détaillé : une version simplifiée du bloc démographique du modèle Destinie

3.1 Introduction

L'exemple qui a précédé a permis d'introduire les principaux ingrédients d'un modèle de microsimulation sur des exemples très stylisés, sans aller jusqu'à la présentation d'un modèle en vraie grandeur. Le but de cette troisième partie est de présenter un exemple plus systématique. Il s'agit d'un premier essai de réécriture, en R, du cœur du volet démographique du modèle Destinie 2 de l'Insee. La démarche a consisté à transposer en R les principes de programmation retenus pour Destinie 2, programmé en langage Perl, mais en bénéficiant des facilités offertes par R qui, comme on le verra, permettent à ce module de se présenter sous forme très compacte.

Revenons d'abord sur les objectifs généraux du modèle Destinie. Comme on l'a expliqué en section 1.2, la microsimulation ne présente aucune valeur ajoutée si elle se borne à reproduire les résultats de projections démographiques obtenues par la méthode déterministe usuelle des composants. Elle n'est utile que si l'on vise d'autres objectifs. D'une part si on veut que les résultats des projections démographiques servent de base à d'autres types de projections pour lesquelles la microsimulation est incontournable : c'est le cas de la projection des règles d'un système de retraite complexe. D'autre part si on veut enrichir la représentation des phénomènes démographiques, par exemple si on veut simuler les structures familiales, ou faire dépendre la fécondité d'autres facteurs que de l'âge, ou encore si on veut simuler une mortalité différentielle... Tous ces aspects sont présents dans le modèle Destinie. Son objectif principal est la simulation des droits à retraite et de leur incidence sur le niveau de vie des retraités : ceci nécessite de recourir à la microsimulation en y incluant une projection de la façon dont les retraités sont regroupés en ménages, à la fois pour savoir comment s'agrègent les droits individuels des conjoints lorsqu'ils sont tous deux en vie, et pour la simulation des conséquences du veuvage ou des droits familiaux.

Le bloc démographique constitue donc la base d'un tel modèle et c'est sur ce bloc qu'on va se concentrer ici. On va d'abord préciser quelles sont les données que simule le modèle et quelles sont celles dont il a besoin en entrée. Puis on présentera un certain nombre de fonctions mises au point pour le traitement de ces données, regroupées dans un fichier `OutilsDestinie.R` qui inclut aussi la prédéclaration de la majorité des variables utilisées par le modèle. Ce fichier sera le pendant de la bibliothèque `OutilsModeleMDT.R` du dernier exemple de la section précédente. On présentera ensuite le détail du programme de simulation proprement dit, découpé en plusieurs étapes : la lecture des données micro initiales et des données de calage, la simulation de la formation et la dissolution des unions puis les simulations successives des naissances, des migrations et de la mortalité.

3.2 Principes d'organisation des données et conventions d'indiçage

3.2.1 Variables micro

Une étape préalable à la modélisation est le choix d'une structure d'organisation des données, qui dépend des objectifs du modèle. L'objectif est ici de gérer les trajectoires individuelles d'un ensemble d'individus et leurs relations familiales. Ceci se fait selon l'approche dite « fermée » décrite en section 1.3.2. On rappelle que ce terme n'est pas à entendre au sens d'absence de migration puisque le modèle simule des flux migratoires. Il

signifie que l'ensemble de la formation des couples se fait entre membres de la population simulée et que ce sont les enfants issus de ces unions qui assurent le renouvellement de la population à sa base. On a expliqué que ceci suppose une « fermeture » du fichier de données de l'enquête *Patrimoine* mais on ne reviendra pas ici sur cette étape de préparation des données supposée avoir été réalisée en amont.

Comment gérer ce type de population ? On peut opter pour une représentation dans laquelle chaque individu est décrit par un enregistrement regroupant ses différentes caractéristiques. Comme dans les exemples de la section précédente, on retient ici une autre approche, qu'on peut qualifier d'approche en représentation vectorielle consistant à définir autant de vecteurs que de caractéristiques des individus, en les initialisant à une taille suffisamment large pour gérer l'ensemble des individus que la simulation va être appelée à traiter. On définit par exemple des vecteurs colonnes `sexe` et `age`, ce qui permettra d'accéder au sexe et à l'âge de l'individu `i` par `sexe[i]` et `age[i]`. Avec cette représentation, on peut directement mettre en œuvre la plupart des procédures statistiques de R sur les variables concernées, soit pour des calculs intermédiaires, soit pour la production de résultats finaux, par exemple des calculs d'âge moyen ou de sex ratio sur tout ou partie de la population.

Selon le même principe, on va introduire des variables stockant des identifiants des personnes liées à l'individu. Les vecteurs `pere` et `mere` donneront les numéros des père et mère de chaque individu, ce qui autorisera la notation intuitive `age[pere[i]]` pour obtenir directement l'âge du père de l'individu `i`.

Cette réprésentation vectorielle est cependant insuffisante dans certains cas. Elle suffit à des données qui sont des caractéristiques permanentes des individus, telles que le sexe ou l'identité des parents biologiques. Pour les données variables au cours du temps, un stockage unidimensionnel signifie qu'on ne peut connaître que la valeur courante de la variable et aucune de ses valeurs antérieures. Ceci n'est pas problématique pour l'âge dont les valeurs passées peuvent être reconstituées en cas de besoin. En revanche, en vue des calculs de retraite, il faut garder trace de la séquence complète des statuts passés par rapport à l'emploi. Même si le module présenté ici ne concerne que la démographie, on va donc prévoir une telle variable `statut` indicée par `i` et par `t`, qui va également nous servir à contrôler la présence effective de l'individu dans la population à chaque date. Il s'agira d'une variable indicée à la fois par la date et par l'identifiant individuel, avec les différentes possibilités suivantes : 1 codera un individu en vie à la date t et présent sur le territoire, -1 correspondra à un individu pas encore né à la date `t`, -2 à un individu résidant à l'étranger, qui pourra être soit un futur migrant soit une personne ayant quitté le territoire, -3 correspondra enfin à un individu décédé, 0 correspondra à une position entièrement vacante qui n'est affectée ou pré-affectée à aucun individu.

Ce codage implique qu'une position `i` sera attribuée une fois pour toutes à un et un seul individu : les positions libérées par les décès ne sont pas réutilisées pour stocker les informations relatives à de nouveaux individus, contrairement à ce que faisait le dernier exemple de la section précédente. Ceci permet une programmation plus propre. Par ailleurs, conserver une information relative à des individus décédés peut être utile à certaines extensions du modèle : par exemple, ceci permet au modèle de conserver la trace des liens entre frères et sœurs même après le décès de leurs parents communs, via l'information sur les enfants de ces parents.

Avec ce mode de stockage, le vecteur simple des statuts courants de l'ensemble des individus simulés à un stade ou à un autre du programme est donné par `statut[,t]`, et un filtrage sur les individus présents dans la population à cette date `t` s'écrira `statut[,t]>0`. Par exemple, l'âge moyen des hommes présents à la

date `t` s'écrira `mean(age[which(statut[,t]>0 & sexe==1)`[11]. Pour le développement d'un module marché du travail, il resterait ensuite à éclater le code 1 en autant de codes positifs qu'on veut distinguer d'états sur le marché du travail. Par exemple si 2 est le code retenu pour la situation de chômeur, on pourra avoir les instructions du type suivant, utilisant le tableau bidimensionnel aussi bien en colonne qu'en ligne

```
liste_chomeurs <- which(statut[,t]==2)
duree_chom[i]  <- sum(statut[i,1:t]==2)
annees_chom    <- which(statut[i,]==2)
```

la première instruction donnant la liste des individus au chômage à la date `t`, la seconde retournant, pour l'individu `i`, le nombre d'années passées au chômage jusqu'à la date `t` incluse, et la dernière donnant les numéros de ces années passées en chômage.

Ce mode de stockage interdit évidemment que `t` puisse varier sur un intervalle trop large. Dans les simulations de la section précédente où les dates étaient fictives, le problème de l'intervalle de variation de `t` ne se posait pas : on le faisait varier de 1 à l'horizon retenu. S'agissant de projections pour des dates réelles, on ne peut utiliser tels quels les millésimes effectifs, ce qui imposerait des tableaux à plus de 2000 colonnes dont la plupart seraient sans objet. Mais prendre la convention `t=1` pour l'année de début de projection n'est pas souhaitable non plus. Pour des calculs de retraite, on a besoin de données rétrospectives pour des individus déjà âgés. Pour ce faire, on a ici choisi la convention de numéroter les millésimes modulo 1900, c'est-à-dire affecter l'indice 1 à l'année 1901, ce qui est suffisant pour stocker des biographies rétrospectives de personnes ayant atteint 109 ans en 2010. Ainsi, `statut[298,110]` est le statut de l'individu de rang 298 en 2010. Dans la mesure où la taille des matrices doit être prédéfinie, on a fixé une borne supérieure aux variations de `t` et aussi à celles de `i`, soit respectivement `horizon` et `taille_max`, avec, en pratique, `horizon = 160` correspondant à l'horizon 2060 des dernières projections de population de l'Insee sur lesquelles le modèle va être calé. S'agissant de la valeur de `taille_max`, pour un échantillon au 10 000ème de la population française, il faut prévoir le stockage d'environ 6700 personnes présentes en début de projection, puis les positions à occuper par tous les individus à naître ou à venir résider sur le territoire. Dans ce qui suivra, ce paramètre sera fixé à 15000.

Le même principe de stockage sera utilisé pour une variable servant à enregistrer la séquence des situations matrimoniales : si l'individu est en union, cette variable est positive et contient l'identifiant du conjoint, sinon, elle est négative avec les codes -1 pour un(e) célibataire, -2 pour un(e) personne séparé(e) et -3 pour un veuf ou une veuve. Cette variable sera notée `conjoint`, ce qui autorisera les expressions suivantes :

```
liste_celib     <- which(conjoint[,t]==-1)
liste_conjoints <- unique(conjoint[which(conjoint[i,1:t]>0])
duree_union     <- sum(conjoint[i,]==j)
annees_veuvage  <- which(conjoint[i,]==-3)
```

[11] Pour information, la version Perl du modèle Destinie 2 utilise une autre modalité de stockage de cette variable, en fonction de l'indice `i` et de l'âge plutôt que de l'indice `i` et de la date. Elle est plus économe en espace mémoire mais elle ne permet pas ce genre de syntaxe puisqu'une tranche verticale au sein d'un tableau par âge et individu ne correspond à aucune période d'observation particulière. Pour contourner cette difficulté, Destinie gère en fait deux variables : la variable `statut_` par identifiant et âge donnant les statuts successifs de l'individu tout au long de son cycle de vie, et une variable `statut` ne dépendant que de `i` et gérant le statut courant de l'ensemble des individus de la population, à utiliser pour les tabulations transversales. Avec un tel système, le programme de simulation doit évidemment être tel que, à chaque date, on vérifie `statut[i]==statut_[i,age[i]]`.

la première donnant la liste des individus célibataires, la seconde donnant la liste des identifiants des conjoints successifs de l'individu `i` (en utilisant l'instruction `unique` qui supprime l'ensemble des doublons d'un vecteur), la troisième la durée de son union avec l'individu `j` et la quatrième donnant enfin la liste des années passées par `i` dans l'état de veuvage.

Le double indiçage sera enfin utilisé pour stocker une autre information, mais sans dimension temporelle. Il s'agira de la liste des identifiants des enfants biologiques de chaque individu. Le lien de filiation étant une caractéristique permanente liant deux individus, il n'y a pas lieu d'en enregistrer des variations au cours du temps. Le second indice ne correspondra donc pas à la date, mais au rang de l'enfant : `enf[i,e]` sera l'identifiant du `e`-ième enfant de l'individu `i`, `enf[i,]`correspondra à la liste des identifiants de l'ensemble des enfants de `i`, qu'ils soient en vie ou non à l'instant courant, ce qui peut ensuite être contrôlé en examinant la valeur de `statut[enf[i,e]]`. Avec cette notation, `length(which(enf[i,]))` donnera le nombre total d'enfants de `i`, mais, compte tenu de l'utilité d'accéder directement à cette information, on va également prévoir une variable stockant cette information, qui sera notée `n_enf[i]`: on verra plus loin l'utilisation de cette variable lors de la simulation des naissances. Par ailleurs, du fait de la contrainte de prédéfinir les dimensions des objets matriciels, on a dû imposer une borne supérieure au nombre d'enfants possible pour un même individu soit `nb_enf_max`.

Compte tenu de ces éléments, la section du module `OutilsDestinie.R` consacrée aux déclarations de paramètres et variables se présente de la manière suivante, avec quelques paramètres supplémentaires que sont la date de début des simulations, la valeur plafond pour l'âge, et des intitulés en clair pour les codes des variables `statut` et `conjoint` plus faciles à mémoriser que les valeurs numériques.

```
# Paramètres généraux de la microsimulation
taille_max   <- 15000
nb_enf_max   <- 6
age_max      <- 108
t_deb        <- 107
t_fin        <- 160

# Codes
non_ne    <- -1
hors_terr <- -2
decede    <- -3
present   <-  1
celib     <- -1
separ     <- -2
veuf      <- -3

# Declaration variables micro
anaiss     <- numeric(taille_max)
adeces     <- numeric(taille_max)
age        <- numeric(taille_max)
sexe       <- numeric(taille_max)
findet     <- numeric(taille_max)
pere       <- numeric(taille_max)
mere       <- numeric(taille_max)
n_enf      <- numeric(taille_max)
enf        <- matrix (0,nrow=taille_max,ncol=nb_enf_max)
conjoint   <- matrix (0,nrow=taille_max,ncol=t_fin     )
```

```
statut     <- matrix (0,nrow=taille_max,ncol=t_fin     )
```

D'autres variables travaillant au niveau des individus pourront être utilisées pour des calculs intermédiaires et seront déclarées dans le programme appelant, notamment des listes d'identifiants d'individus auxquels tel ou tel traitement doit être appliqué : ces listes pourront découler soit de l'appel à la fonction `tirage`, soit de toute autre forme de filtrage de la population. Pour un évènement simple, une seule liste a en principe besoin d'être construite, mais il peut en falloir plusieurs, par exemple dans le cas de la simulation des mises en couple qui nécessitera de constituer deux listes de conjoints potentiels entre lesquels se feront les appariements.

3.2.2 Variables macro

Les autres variables principales qui vont être fournies en entrée du programme vont être les données issues des projections démographiques établies par la méthode traditionnelle des composants, qui vont soit servir de données de calage pour la microsimulation, soit de données de référence auxquelles les résultats de la simulation pourront être comparés. Ces données se présentent toutes sous forme de tableaux par âge et période ou par sexe, âge et période. Ils seront systématiquement indicés dans cet ordre. Lorsqu'il y a plus de deux indices, le type requis par R est le type `array` et non plus `matrix`. On partage avec l'organisation des tableaux `statut` et `conjoint` le principe d'avoir la date `t` comme dernier indice. Ces données vont être les populations par sexe et âge détaillés, les flux de migration nette par sexe et âge et enfin les décès par sexe et âge.

Des variables de même type seront également calculées en sortie du programme. Pour celles qui constituent des inputs, leur lecture se fera à l'aide d'une fonction `lec_tab` décrite à la section suivante. Ces variables auraient pu être prédéclarées et initialisées dans la bibliothèque `OutilsDestinie.R`, mais, dans la mesure où leur liste ou leur contenu sont susceptibles de varier d'une version à l'autre de la microsimulation, on verra plus loin que leur déclaration et leur initialisation se font dans le programme appelant.

Un autre point technique doit être précisé sur l'indiçage de ces variables. Le fait de ne pouvoir démarrer l'indiçage que par la valeur 1 crée une contrainte pour la gestion de l'âge, puisque ceci interdit d'identifier en clair une population d'âge zéro. On résoud ce problème en raisonnant en termes d'âge atteint dans l'année. Plus précisément, une quantité telle que `pop[s,a,t]` sera supposée contenir l'effectif, au premier janvier de l'année `t`, des gens de sexe `s` qui atteindront leur `a`-ieme anniversaire au cours de cette année `t`. Les individus nés au cours de l'année `t-1` arrivent donc au 1er janvier de l'année `t` avec un âge conventionnellement fixé à un an, même si leur âge en années révolues est encore égal à zéro. En raison de la même contrainte, une variable de flux telle que `deces[s,a,t]` va servir à mesurer le flux de décès, au cours de l'année `t`, des gens qui auraient atteint l'âge `a` au 1er janvier de l'année suivante : `deces[s,1,t]` mesure donc le flux de décès, en cours d'année, parmi les naissances de cette même année `t`. Ces conventions n'empêcheront pas la variable individuelle `age[i]` de prendre temporairement la valeur zéro. L'année de leur naissance, les individus auront effectivement un âge nul, et c'est au moment du passage au 1er janvier de l'année suivante que leur âge sera incrémenté à un pour les faire figurer dans la première case des tableaux tels que `pop[s,a,t]`.

	Rôle	Arguments	Retour
(a) Gestion d'individus et des liens familiaux			
`clone`	Création d'une copie de l'individu i en position j	`i, j`	-
`delete`	Délétion de l'individu i (remise à zéro)	`i`	-
`liste_enf`	Liste des enfants de l'individu i	`i, option`	Liste d'identifiants
`nb_enf`	Nombre d'enfants de l'individu i	`i, option`	Scalaire
(b) Probabilités d'évènements démographiques			
`proba_union`	Calcul des probabilités individuelles de mise en couple à la date t	`t`	Vecteur de valeurs individuelles
`proba_separ`	Calcul des probabilités individuelles de séparation à la date t	`t`	Vecteur de valeurs individuelles
`proba_naissance`	Calcul des probabilités individuelles de naissance d'un enfant à la date t	`t`	Vecteur de valeurs individuelles
(c) Lecture de données			
`init`	Initialisation des données individuelles	`nom_fichier`	-
`lec_tab`	Lecture de tableau par âge et période (données de calage)	`nom_fichier`	Tableau âge/période
(d) Tabulations			
`pyram`	Effectifs de la population totale ou d'une sous-population, par âge simple ou regroupé	`filtre, pas`	Tableau par âge
`profil`	Moyenne ou autre statistique d'une variable selon l'âge, sur la population totale ou une sous-population	`variable, filtre, type de statistique, pas`	Tableau par âge

Table 4: Listes des fonctions de la bibliothèque `OutilsDestinie.R`.

3.3 Fonctions prédéfinies

Le reste du fichier `OutilsDestinie.R` est consacré à spécifier un ensemble de fonctions listées sur le tableau 4 et qui sont de quatre types : (a) diverses fonctions élémentaires de gestion des indivdus ou de leurs liens familiaux, (b) des fonctions permettant de calculer les probabilités de réalisation de certains évènements démographiques, (c) des fonctions permettant la lecture de données en entrée du modèle et enfin (d) des fonctions de tabulation permettant d'en alimenter les sorties. On va surtout détailler les fonctions des trois derniers groupes.

3.3.1 Probabilités d'évènements démographiques

Les fonctions calculant des probabilités d'évènements démographiques permettent de construire les vecteurs de probabilités de tirage dont a besoin la fonction `tirage`. Trois fonctions de ce type sont proposées. Il s'agit des fonctions `proba_union`, `proba_separ`, `proba_naissance` avec la date pour argument d'appel car ces fonctions ont besoin de se référer aux caractéristiques individuelles observées à une date donnée, par

exemple le fait d'être déjà en union ou non. Toutes ces probabilités doivent être vues comme des probabilités *ex ante* : les réalisations effectives prendront ou non en compte d'éventuels calages.

Dans le cas présent, ces fonctions ont été programmées et paramétrées en adaptant les estimations réalisées par Duée (2005) pour l'alimentation de Destinie[12]. Il ne s'agit que d'exemples de la forme que peuvent prendre de telles fonctions, avec des paramètres dont les valeurs numériques sont incorporées en dur, mais pourraient évidemment être introduites de façon plus flexible. On donne ci-dessous le code de la plus simple d'entre elles, la fonction `proba_separ` :

```
proba_separ <- function(t)
# Les probas sont appliquées à la femme
{
  p <- rep(0,taille_max)
  for (i in 1: taille_max)
  {
    if (statut[i,t]>0 && sexe[i]==2 && conjoint[i,t]>0)
    {
      d     <- t-min(which(conjoint[i,]==conjoint[i,t]))     # Durée de l'union
      n     <- length(intersect(liste_enf(i,t),
                                liste_enf(conjoint[i,t],t))) # Nbre d'enfants nés de l'union
      n_aut <- n_enf[i]-n                                    # Enfants autre union
      p[i]  <- -2.92-0.06*d-0.04*(age[i]-d)+0.58*(n==0)+0.21*(n==1)+0.13*(n>3)+0.41*(n_aut>0)
      p[i]  <- exp(p[i])/(1+exp(p[i]))
    }
  }
  return (p)
}
```

On relèvera qu'on a choisi d'affecter les probabilités de séparation à la femme, mais ce choix est conventionnel : cette probabilité peut être évaluée en fonction des caractéristiques propres de l'un ou l'autre des deux conjoints ou de leurs caractéristiques communes. Le fait que les calculs puissent être un peu complexes explique qu'ils soient menés individu par individu et non pas à l'aide d'instructions vectorielles en bloc. Les variables individuelles qui servent à évaluer la probabilité de rupture sont la durée de l'union `d`, l'âge de la femme à l'entrée en union (`age[i]-d`), le nombre d'enfants issus de l'union et la présence d'autres enfants de la femme non issus de cette union. La durée de l'union est calculée en allant rechercher la date la plus ancienne pour laquelle le conjoint de l'individu `i` est égal à son conjoint courant. Le calcul du nombre d'enfants issus de l'union mobilise pour sa part une des fonctions du groupe (a) du tableau 4, la fonction `liste_enf <- function(i,t,option="tous")` qui retourne la liste des identifiants des enfants de l'individu `i` à la date `t` : par défaut l'ensemble de ces enfants, ou seulement les enfants encore en vie (si `option="vivants"`). Ainsi, les appels `liste_enf(i,t)` et `liste_enf(conjoint[i,t],t)` donnent respectivement, pour la date `t`, les enfants de `i` et de la personne qui est son conjoint à cette même date `t`, et l'appel à la fonction R `intersect` permet de ne retenir que les identifiants figurant dans les deux listes.

3.3.2 Lecture de données

Les données à lire en entrée sont de deux types. Il faut d'une part initialiser l'ensemble des données individuelles. On suppose celles-ci stockées dans un fichier unique et leur lecture se fera par la fonction `init`, qui

[12] Ces estimations avaient été conduites sur les données biographiques de l'enquête *Histoires familiales* de 1999.

procède aussi à diverses imputations supplémentaires de variables non disponibles dans ce fichier mais qui sont nécessaires au fonctionnement du programme : dans cet exemple, il s'agit des valeurs rétrospectives de la variable `conjoint`, nécessaires au calcul des durées d'union qui déterminent les probabilités de survenue de divers évènements démographiques. Les autres données à lire en entrée sont celles qui serviront au calage du modèle qui sont tirées des projections démographiques standards, tels que les effectifs, les flux migratoires ou les decès par sexe, âge et période, qui ont été décrites à la section 3.2.2. Toutes ces données se présentent comme des tableaux age × période ou période × age et la fonction `lec_tab` permet de les lire dans des fichiers de type `.csv` correspondant à un des types de fichiers gérés par Excel, facilement lisible sous R.

3.3.3 Tabulations

Concernant enfin les tabulations, on dispose évidemment de tout l'ensemble de fonctions déjà programmées en R, mais deux types de tabulations non standard sont d'usage récurrent dans le modèle et font l'objet des fonctions `pyram` et `profil`. Les deux sortent des profils par âge relatifs à une sous-population définie par un filtre : la première sort un simple comptage, alors que la seconde sort le profil par âge d'une statistique univariée, par exemple un salaire moyen selon l'âge, ou encore sa médiane ou son écart-type, toujours par âge. Les deux fonctions sortent ces résultats soit selon l'âge simple, soit selon un âge regroupé. Des exemples d'appel sont les suivants, dont un exemple montrant comment cette fonction permet aussi de calculer directement des taux par âge :

```
pyramide[,t]    <- pyram(t)
pyrquinq[,t]    <- pyram(t,pas=5)
nb_maries[s,,t] <- pyram(t,sexe==s & matri==2,pas=5)
tx_maries[s,,t] <- pyram(t,sexe==s & matri==2)/pyram(t,sexe==s)
sal_moy[,t]     <- profil(salaire,t,(sexe==2),pas=5)
sd_sal[,t]      <- profil(salaire,t,(sexe==2),pas=5,type="sd")
```

Dans le cas de la fonction `profil` les statistiques pouvant être calculées selon l'âge sont la somme, le minimum, le maximum, la médiane ou l'écart-type de la variable passée en argument, avec respectivement les valeurs `"sum"`, `"min"`, `"max"`, `"median"` et `"sd"` pour le paramètre `type`.

3.4 Le premier bloc du modèle : lecture des paramètres et des données initiales

Une fois détaillé le contenu de la bibliothèque `OutilsDestinie.R`, on peut aborder le contenu du programme de simulation proprement dit. Comme on l'a fait jusqu'ici, on en donne et on en commente le code source intégral, bloc par bloc, mais en continuant d'exclure les appels de procédures graphiques, reportés en annexe 3.

L'en tête du programme comprend les éléments suivants. Après les appels aux bibliothèques `OutilsMS.R` et `OutilsDestinie.R` et la définition du répertoire de travail, le programme déclare les différentes données de calage puis lit leurs valeurs dans les fichiers correspondants à l'aide de la fonction `lec_tab`. Il définit ensuite quelques autres variables macro telles que celles qui serviront à stocker les pyramides des âges simulées et il définit enfin trois vecteurs `liste`, `liste_h` et `liste_f` qui serviront à stocker des listes d'individus à traiter à différents endroits du programme.

```
source("~/Documents/Microsimulation R/Outils/OutilsMS.R")
source("~/Documents/Microsimulation R/Destinie/OutilsDestinie.R")
```

```
setwd("~/Documents/Microsimulation R/Destinie")

# Définition et lecture des données de projection
pop          <- array(0,dim=c(2,age_max,t_fin))  # Effectifs selon l'âge au 1/1/t
deces        <- array(0,dim=c(2,age_max,t_fin))  # Flux selon l'âge au 1/1/t+1
mig          <- array(0,dim=c(2,age_max,t_fin))  # Flux selon l'âge au 1/1/t+1
naiss        <- array(0,dim=c(age_max,t_fin))    # Flux naissances par age de la mere
sex_ratio    <- 0.488                            # Proportion de filles à la naissance
pop[1,,]     <- lec_tab("PopHommes.csv"     ,age="colonne")
pop[2,,]     <- lec_tab("PopFemmes.csv"     ,age="colonne")
deces[1,,]   <- lec_tab("DecesHommes.csv"   ,age="colonne")
deces[2,,]   <- lec_tab("DecesFemmes.csv"   ,age="colonne")
mig[1,,]     <- lec_tab("SoldeMigHommes.csv",age="colonne")
mig[2,,]     <- lec_tab("SoldeMigFemmes.csv",age="colonne")
naiss        <- lec_tab("Naissances.csv"    ,age="colonne")


# Données intermédiaires et d'output
popsim       <- array(dim=c(2,age_max,t_fin))
en_couple    <- array(dim=c(2,age_max,t_fin))
ratio        <- array(dim=c(t_fin))
ratiosim     <- array(dim=c(t_fin))

# variables micro annexes
liste        <- numeric(taille_max)
liste_h      <- numeric(taille_max)
liste_f      <- numeric(taille_max)

# Initialisation
set.seed(123456789)
init("popini_new.csv")
taux_sondage <- length(which(statut[,t_deb]>0))/sum(pop[,,t_deb])
```

Le bloc s'achève par l'initialisation du générateur d'aléatoires et celle des données micro par la fonction `init`, immédiatement suivie du calcul du taux de sondage apparent du fichier, par comparaison entre l'effectif des projections de référence et le nombre d'individus vivants et présents à la date initiale `t_deb`. Ce paramètre `taux_sondage` permettra soit de convertir les effectifs de calage en effectifs de la microsimulation, à l'aide de la fonction `reduc`, soit au contraire de convertir les résultats de comptages issus du modèle en variables en vraie grandeur, à l'aide de la fonction `augm`, appartenant toutes deux à l'autre bibliothèque `OutilsMS.R`.

3.5 Début de la boucle temporelle

Une fois déclarées les variables et lues les données initiales ou de calage, le programme débute la boucle de simulation annuelle. On rappelle que les données présentes en début de boucle sont des données au 1er janvier de l'année considérée, avec des âges individuels qui sont les âges qui vont être atteints dans l'année. La première étape de la boucle consiste à prendre la photographie de la population à cette date du premier janvier, et la suite de la boucle temporelle consistera à modifier les caractéristiques des différents individus pour les mettre dans ce que sera leur état au 1er janvier suivant. Cette mise à jour n'a évidemment pas lieu

d'être si on est arrivée en fin de projection, d'où l'instruction `break` qui fait sortir de la boucle dans le cas `t==t_fin`.

```
# Début de la simulation

for (t in t_deb:t_fin)
{
  # Calcul données en début d'année pour sorties graphiques et sortie si date finale
  for (s in 1:2)
  {
    popsim[s,,t]    <- pyram(statut[,t]>0 & sexe==s)
    en_couple[s,,t] <- pyram(statut[,t]>0 & conjoint[,t]>0 & sexe==s)
  }
  ratiosim[t] <- sum(popsim[,60:age_max,t])/sum(popsim[,20:59,t])
  ratio[t]    <- sum(pop   [,60:age_max,t])/sum(pop   [,20:59,t])

  if (t==t_fin) {break}

  # Prolongement par défaut des variables statut et conjoint
  statut[,t+1]   <- statut[,t]
  conjoint[,t+1] <- conjoint[,t]
```

Si cette date terminale n'est pas atteinte, l'étape suivante consiste à donner immédiatement des valeurs par défaut aux positions `t+1` des deux variables individuelles indexées par le temps, à savoir `statut` et `conjoint`. Par défaut, elles sont prolongées à leurs valeurs courantes, et on peut considérer que le reste du programme va essentiellement consister à corriger ce prolongement par défaut à chaque fois qu'il doit l'être, en même temps que la mise à jour des autres variables individuelles non indicées par le temps..

3.6 Simulation des séparations et des mises en couple

Les deux blocs suivants vont commencer par mettre à jour la variable `conjoint` en simulant successivement les séparations et les mises en couple. On débute par la simulation des séparations. Un vecteur de probabilités de séparation est établi individu par individu par appel de la fonction `proba_separ` vue plus haut et directement passé comme argument à la fonction `tirage` pour construire la liste de femmes en couple qui vont effectivement connaître une séparation dans l'année, après quoi il suffit de donner la valeur `separ` à la variable `conjoint` à la fois pour ces femmes et pour leurs conjoints.

```
  # Simulation des séparations
  liste <- tirage(proba_separ(t))
  conjoint[conjoint[liste,t],t+1] <- separ
  conjoint[liste,t+1]             <- separ

  # Simulation des mises en couples
  liste_f <- tri(tirage(proba_union(t)*(sexe==2)),age)
  liste_h <- tri(tirage(proba_union(t)*(sexe==1),cible=length(liste_f)),age)
  conjoint[liste_f,t+1] <- liste_h
  conjoint[liste_h,t+1] <- liste_f
```

La simulation des mises en couple est tout aussi rapide, au moins sous la forme très allégée qu'on lui donne ici. On crée une première liste de femmes tirées sans contrainte sur la base des probabilités retournées par

la fonction `proba_union`, restreinte à la population féminine, ce qu'on obtient directement par le produit `proba_union(t)*(sexe==2)` et on trie immédiatement cette liste selon l'âge qui, ici, sera notre seul critère d'appariement. Puis on fait la même opération pour les hommes mais en contraignant la liste tirée à être de même taille que la liste féminine. On réalise ensuite l'appariement en mettant à jour les variables `conjoint` de la date `t+1` pour chacune de ces deux listes. Il s'agit donc d'une modalité très rudimentaire d'appariement, mais l'écriture serait à peine plus longue en appariant selon un critère plus complexe que l'âge. *A minima*, on peut flexibiliser marginalement cet appariement en triant non pas selon l'âge brut, mais selon l'âge augmenté d'un aléa plus ou moins important, comme on l'avait fait pour le lien employabilité/ancienneté dans le chômage de l'exemple de la fin de la section 2.

3.7 Simulation des naissances

Une fois mises à jour les unions, on procède à la simulation des naissances. Contrairement à la simulation des formations/dissolutions d'unions pour lesquelles aucun élément de calage n'est fourni par les projections officielles, on va ici se caler directement sur les naissances totales données par les projections standard.

Puisque le résultat de cette étape va être de créer de nouveaux individus, on commence par stocker dans le vecteur `liste` l'ensemble des positions vacantes : c'est dans cette liste qu'on va puiser progressivement pour positionner les nouveaux-nés. On crée ensuite la liste de femmes qui vont avoir une naissance dans l'année, selon la probabilité de tirage `proba_naissance(t)`, contrainte à la somme des naissances par âge des projections standard. On parcourt ensuite la liste des mères ainsi constituée. Pour chaque naissance, on se positionne sur la première des positions vacantes restant dans `liste`, à savoir `e <- liste[1]` puis on attribue à cet individu `e` les caractéristiques requises. En particulier, mais ceci n'est qu'indicatif, ce que sera l'âge de fin d'études de ce nouvel individu est généré comme moyenne simple de ceux de ses deux parents, cette spécification pouvant être facilement enrichie. Symétriquement, on affecte cet individu `e` comme nouvel enfant à chacun de ses deux parents biologiques. On note aussi que, comme annoncé plus haut en section 3.2.2, l'âge est temporairement mis à zéro : il ne passera à la valeur 1 qu'en fin de boucle temporelle, lorsqu'on mettra à jour les âges de l'ensemble de la population. Une fois finie la création du nouvel individu, on retire cette position `e` de la liste des positions vacantes par `liste <- extr(liste,-1)` avant de passer à la simulation de la naissance suivante.

```
liste   <- which(statut[,t+1]==0)
liste_f <- tirage(proba_naissance(t),cible=reduc(sum(naiss[16:51,t])))
for (m in liste_f)
{
  e                  <- liste[1]
  p                  <- conjoint[m,t]
  sexe[e]            <- 1+rbinom(1,1,sex_ratio)
  mere[e]            <- m
  pere[e]            <- p
  findet[e]          <- 0.5*(findet[p]+findet[m])
  n_enf[m]           <- n_enf[m]+1
  n_enf[p]           <- n_enf[p]+1
  enf[m,n_enf[m]]    <- e
  enf[p,n_enf[p]]    <- e
  anaiss[e]          <- t
  age[e]             <- 0 # Age = 0 à la naissance, sera incrémenté en fin de boucle
```

```
  statut[e,t]         <- 1
  conjoint[e,t]       <- celib
  enf[e,]             <- 0
  n_enf[e]            <- 0
  liste               <- extr(liste,-1)
}
```

On notera au passage qu'il est très facile de désactiver le calage des naissances simulées sur celles de la projection de référence, si on souhaite s'assurer que le comportement du modèle non calé est spontanément assez satisfaisant pour que le calage ne joue bien qu'un rôle résiduel, suivant la recommandation faite plus haut en section 2.5.2. Il suffit de supprimer la mention de la cible dans l'appel de la fonction `tirage`.

3.8 Simulation des migrations

Pour la simulation des migrations, il n'y a plus besoin de recourir à des calculs de probabilités *ex ante*. On génère directement des nombres de migrants pour chaque croisement sexe × âge en ligne avec les flux migratoires fournis par les projections de référence. Si le solde migratoire à simuler est négatif, on constitue la liste des personnes susceptibles de migrer âgées de `a-1` au 1er janvier et on en retient les `-mig[s,a,t]` premières pour qui on met la variable statut à `hors_terr`. Symétriquement, si le solde migratoire est positif, on établit la liste des positions vacantes, on en extrait les `mig[s,a,t]` premières et on y crée les nouveaux individus en nombre requis, de sexe `s`. Leur âge est temporairement mis à `a-1` et sera incrémenté en fin de boucle en même temps que les âges du reste de la population, l'année de naissance étant choisie de manière cohérente avec cet âge. On notera que, comme annoncé plus haut, cette gestion du compteur d'âge conduit à traiter des individus d'âge zéro, même s'ils ne seront pas comptabilisés comme tels au 1er janvier suivant : il pourra s'agir soit de sorties du territoire de nouveaux-nés simulés à l'étape précédente, ou au contraire d'entrées faisant immédiatement suite à une naissance. Ces flux sont calés sur les valeurs `mig[s,1,t]` compte tenu du mode de rangement des données par âge dans le tableau `mig`, contraint par la borne d'indiçage à un du premier élément.

```
# Simulation des migrations
for (s in 1:2)
{
  for (a in 1:age_max)
  {

    # Flux sortant: on tire les sortants parmi les présents d'âge a-1
    if (mig[s,a,t]<0)
    {
      liste             <- which(statut[,t]>0 & sexe==s & age==(a-1))
      liste             <- permut(liste)
      liste             <- extr(liste,reduc(-mig[s,a,t]))
      statut[liste,t+1] <- hors_terr
    }
    # Flux entrant: on tire dans la liste des positions vacantes
    else if (mig[s,a,t]>0)
    {
      liste             <- which(statut[,t+1] == 0)
      liste             <- extr(liste,reduc(mig[s,a,t]))
      statut[liste,t+1] <- 1
```

```
      age[liste]        <- a-1
      sexe[liste]       <- s
      anaiss[liste]     <- t-a+1
    }
  }
}
```

Cette façon de simuler les migrations reste évidemment très partielle. D'une part, elle simule la migration de manière individuelle. Simuler une migration familiale supposerait une programmation plus complexe, car faire migrer en bloc un individu et sa famille conduit à intervenir sur les soldes migratoires simultanés de plusieurs tranches d'âge. Il devient plus difficile de respecter l'ajustement global.

Par ailleurs, on suppose que l'immigration correspond forcément à l'ajoût de nouveaux individus, alors qu'il pourrait en partie s'agir d'individus ayant antérieurement quitté le territoire et qui y reviennent. Dans ce cas, il faut prévoir que l'alimentation de la population se fasse à la fois en ajoutant des individus sur des positions vacantes, et en allant puiser dans le stock d'individus connus du modèle mais pour lesquels la variable `statut` est à la valeur `hors_terr`. Le problème de cette démarche est que ce stock n'est en général pas connu initialement si la base du modèle est le fichier d'une enquête collectée sur le territoire national. Si on voulait introduire ce type de complication, il faudrait initialiser le fichier en y rajoutant par imputation des individus appartenant à cette population ayant quitté le territoire avant la date `t_deb` et susceptibles d'y revenir.

3.9 Simulation des décès et fin de la boucle temporelle

Pour finir, le modèle simule la mortalité en se calant sur le tableau `deces`. La liste de commandes est encore plus simple que dans les cas précédents. Par sexe puis par âge, il s'agit de réaliser un nombre de décès égal à `deces[s,a,t]`. Toujours compte tenu des conditions d'indiçage, les individus concernés sont des individus dont l'âge courant est `a-1`, incluant notamment des nouveaux-nés dont l'âge, à ce stade, est encore égal à zéro. On en tire la liste dont on extrait `reduc(deces[s,a,t])` individus, sauf dans le cas où on a atteint l'âge maximum auquel cas l'ensemble de la liste est conservée. On met ensuite à `decede` le statut de l'ensemble des membres de cette sous-liste. Pour le sous-ensemble de ces décédés qui étaient en couple, on met également à `veuf` la variable `conjoint[,t+1]` de leurs partenaires survivants. Enfin, puisqu'on est arrivé en fin de boucle temporelle, on incrémente les âges de l'ensemble des individus qui seront présents dans la population au 1er janvier de `t+1`, de sorte à avoir des données immédiatement exploitables à l'étape suivante de la boucle temporelle.

```
# Simulation des décès
for (s in 1:2)
{
  for (a in 1:age_max)
  {
    liste                           <- which(statut[,t]>0 & sexe==s & age==(a-1))
    if (a<age_max)
    {
      liste                         <- extr(liste,reduc(deces[s,a,t]))
    }
    statut[liste,t+1]               <- decede
    liste                           <- extr(liste,keep=(conjoint[,t]>0))
```

```
      conjoint[conjoint[liste,t],t+1] <- veuf
    }
  }

  # Fin de boucle temporelle : on incrémente les ages des individus
  # qui seront présent l'année prochaine
  age[which(statut[,t+1]>0)] <- age[which(statut[,t+1]>0)]+1

}
```

Comme pour les naissances et la migration, il serait assez facile de complexifier ce segment du modèle pour introduire des déterminants supplémentaires de la mortalité, en sus du sexe et de l'âge. Prenons l'exemple de la mortalité différentielle selon le milieu social. Si on dispose ou si on peut reconstituer des probabilités de décès différenciées par âge et milieu social, une démarche naturelle serait de tirer les décès individuels selon ces probabilités, sans autre contrainte. Mais, ce faisant, on doit formuler des hypothèses prospectives sur l'évolution des probablités de décès de chaque catégorie sociale et il n'y a aucun raison *a priori* que ces hypothèses fassent exactement retomber sur les scénarios de mortalité générale des projections calculées par la méthode des composantes. Pour éviter ce problème, l'aternative est de paramétrer cette mortalité différentielle dans un vecteur de probablités relatives `proba`, dans lequel il n'est pas indispensable d'introduire le facteur âge ni le genre. Ensuite, pour chaque combinaison `a`, `s`, on tire la liste des décès par un tirage selon le vecteur de risques relatifs `proba`, calé sur la cible de décès globaux. Le principe de la séquence peut ainsi être le suivant, dans le cas où la mortalité différentielle se ramènerait à un effet logistique de l'âge de fin d'études `findet`, avec un coefficient `coeff` :

```
  # Simulation des décès avec mortalité différentielle
  proba <- 1/(1+exp(coeff*findet))
  for (s in 1:2)
  {
    for (a in 1:age_max)
    {
      liste             <- tirage(proba*(statut[,t]>0 & sexe==s & age==(a-1))
                                  cible=reduc(deces[s,a,t]))
      statut[liste,t+1] <- decede
    }
  }
}
```

et cette procédure reviendra, pour chaque date de projection et chaque combinaison `a`, `s` à avoir des décès respectant la hiérarchie des risques relatifs par niveau d'éducation donnés par `coeff`, mais ajustée du facteur requis pour avoir le nombre de décès cohérent avec la mortalité générale. Ceci suppose évidemment que l'hypothèse de constance des risques relatifs soit valide, mais rien n'empêche d'avoir une spécification plus compliquée où ces risques relatifs évoluent aussi avec l'âge ou toute autre caractéristique.

Le choix entre cette méthode de projection de la mortalité différentielle et la méthode libre, non contrainte par les projections standard, dépend de l'objectif poursuivi. Si le but est d'utiliser la microsimulation pour explorer les conséquences sur la mortalité générale d'hypothèses bien contrôlées sur les mortalités par sexe, âge et âge de fin d'études, c'est la méthode de projection non contrainte qu'il faut retenir. Mais si l'objectif est uniquement de construire une population prospective conforme à ces projections qui prenne en compte le fait qu'il y a des différentiels de mortalité, la méthode contrainte est totalement justifiée, et assez facile à

mettre en œuvre si on peut caractériser ces différentiels de mortalité par une forme fonctionnelle simple.

3.10 Résultats et prolongements

L'objectif de toute cette troisième partie était uniquement de présenter des principes de programmation et non pas de déboucher sur un module totalement opérationnel. On se bornera donc à quelques résultats illustratifs. Globalement, la population obtenue en 2060 reproduit bien la structure par âge prévue par les projections démographiques usuelles (figure 9B), à la fluctuation d'échantillonnage près, un peu plus forte que celle observée au début de la projection (figure 9A). Ces fluctuations sont essentiellement imputables à des problèmes d'arrondis, notamment sur la simulation des flux migratoires. Sur un croisement sexe/âge annuel, le flux migratoire annuel net est typiquement de l'ordre du millier de personnes. Sur un échantillon au 10 000ème, on tirera donc alternativement une ou zéro entrée avec des probabilités respectives d'un dixième ou de neuf dixième, ce qui génère une variance significative des effectifs. On est évidemment dans un cas extrême en matière de faible taille d'échantillon. Mais on peut constater que les effets de ces fluctuations d'échantillonnage disparaissent totalement sur un indicateur tel que le ratio des 60 ans et plus aux 20-59 ans (Figure 10a).

D'autres exemples de résultats sont les effectifs en couple à chaque âge (figures 9A et B, lignes en grisé) et les nombres d'enfants selon l'âge en 2060 (figure 10b), ce dernier graphique étant un exemple d'utilisation de la fonction `profil` : les données représentées sur ce graphique sont obtenues en fin de boucle par l'instruction `profil(n_enf,statut[,t_fin]>0)`, alors que les données des figures 9A et B étaient obtenues par l'instruction `pyram` utilisée en début de boucle temporelle.

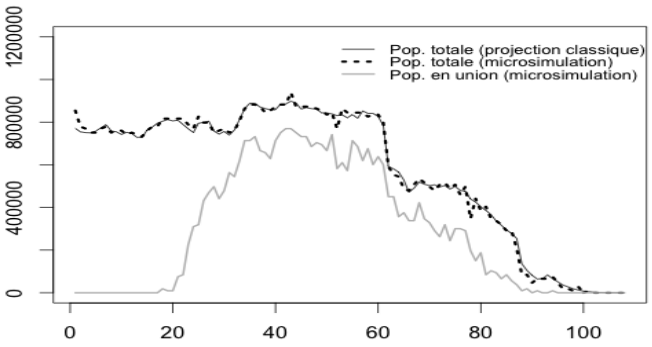
D'autres graphiques de résultats peuvent être construits sur ce modèle, soit au fur et mesure de la simulation, soit *a posteriori* tant que l'ensemble des résultats micro sont stockés dans l'espace de travail de R. Cet espace de travail peut ensuite être archivé et réutilisé pour la sortie d'autres résutats agrégés ou pour servir de base de travail pour les étapes suivantes de la microsimulation. Il y a en fait deux façons de procéder :

- Soit toutes les simulations sont faites en un seul bloc, ce qui supposerait d'intégrer au même programme la simulation des trajectoires d'emploi et les départs en retraite avec l'inconvénient que, d'une variante à l'autre de politique des retraites, c'est l'ensemble des trajectoires individuelles qui seront resimulées ce qui interdira des comparaisons fines de type avant/après réforme sur des individus à caractéristiques inchangées.

- Soit on simule par étapes. C'est ce que fait le modèle Destinie 2 avec un bloc générant des trajectoires démographiques et d'emploi, dont les résultats micro sont archivés, puis un bloc retraite qui relit ces trajectoires et leur applique le ou les scénarios d'évolution du système de retraite qu'on souhaite étudier. L'interêt de cette approche séquentielle est que les simulations des retraites sont plus rapides et portent sur une population de caractéristiques fixes autorisant des comparaisons de droits toutes choses égales par ailleurs. Et ceci n'interdit pas totalement de simuler une rétroaction des réformes des retraites sur certains comportements en amont de la retraite, puisqu'il est toujours possible de retravailler tout ou partie des trajectoires avant retraite en fonction des scénarios de retraite.

Si on procède de la deuxième manière, il suffit de prévoir l'archivage des bases issues des modules amont ensuite réutilisées par les modules aval. Ceci se fait par l'instruction `save`. Par exemple, la commande

A : Année 2007



B : Année 2060

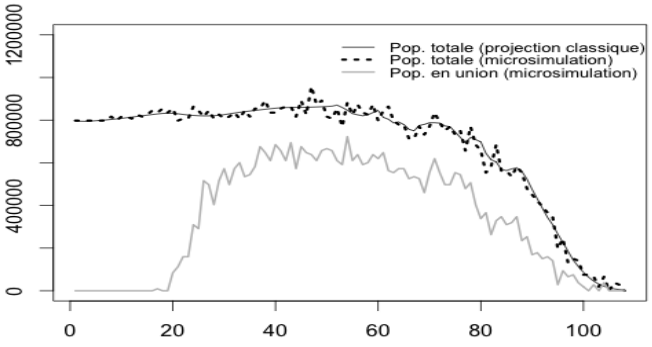


Figure 9: Pyramides initiales et terminales

a : Ratio 60+/20-59

b : Nombre d'enfants selon l'âge (2060)

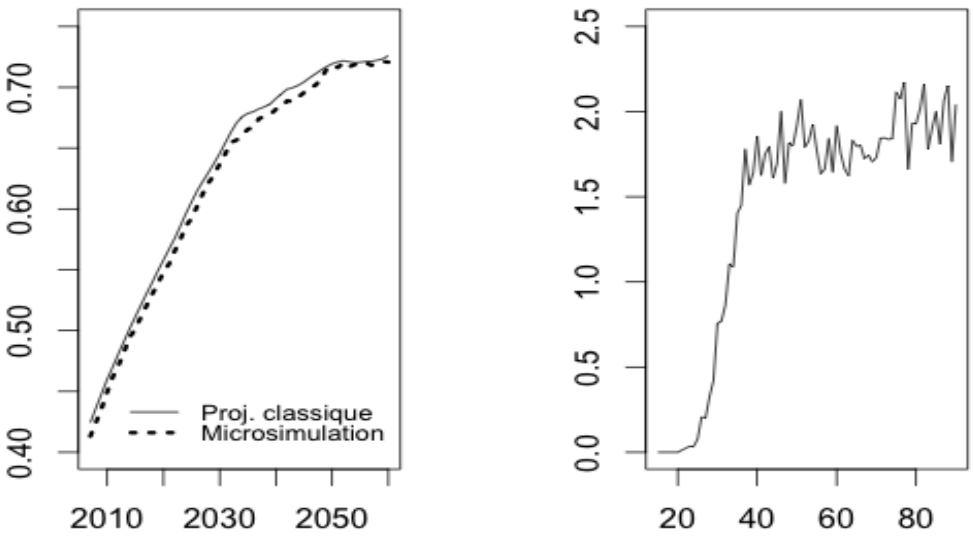


Figure 10: Résultats auxiliaires

`save("Bios.RData")` en fin du programme présenté ici sauvegarderait l'espace de travail courant et donc toute la mémoire des variables individuelles dans un fichier d'extension `RData`. Ensuite, un nouveau programme débutant par un `load("Bios.Rdata")` réaccède directement à ces informations et peut s'en servir comme base pour la simulation d'évènements ou le calcul de résultats additionnels. C'est de cette façon que travaille le modèle PENSIPP.

Conclusion

D'un usage longtemps réservé à quelques domaines très particuliers, tels que l'analyse de la fiscalité, les modèles de microsimulation voient leur application progressivement étendue à des champs de plus en plus variés. On peut y voir trois raisons : l'accroissement des moyens de calcul, l'accès plus facile aux bases de micro-données nécessaires à l'alimentation de ces modèles, et l'intérêt pour la modélisation des inégalités ou de phénomènes complexes qui ne se laissent facilement appréhender qu'en passant par le niveau micro-économique.

On a vu que cette technique a un autre avantage, celui d'être conceptuellement très simple : repartir du niveau individuel permet une modélisation très intuitive. Par exemple, microsimuler des transitions entre états selon des probabilités données *a priori* ne nécessite rien d'autre que la connaissance de ces probabilités et le recours à un langage permettant de générer des tirages pseudo-aléatoires selon ces probabilités, individu par individu. Pour simuler des comportements plus élaborés -par exemple le départ en retraite en fonction du niveau des droits à retraite offerts à un âge donné- il suffit de programmer la représentation qu'on se fait de ce comportement, ce qui est en général faisable dans n'importe quel type de langage. Dans ces deux cas de figure, les principales difficultés ne se situent pas du côté de la technique de microsimulation elle-même. Les coûts les plus importants sont du côté de la collecte des données ou la définition des paramètres qui sont nécessaires à alimenter la simulation, tels que les jeux de probabilités de transition entre états ou les barèmes fiscaux et sociaux en fonction desquels vont s'ajuster les comportements individuels. Ces éléments sont par nature peu mutualisables d'un exercice de microsimulation à l'autre ce qui explique que la majorité des modèles existants se présentent comme des exercices autonomes, réalisés dans des langages très divers, avec pour seul dénominateur commun l'esprit général de la méthode. Corrélativement, la littérature consacrée aux modèles de microsimulation est souvent une littérature grise et peu diffusée, et le plus souvent assez allusive sur le fonctionnement interne des modèles, se focalisant plutôt sur la présentation de leurs caractéristiques générales et de leurs résultats.

Pour autant, il existe un petit noyau de questions communes à tous ces modèles et qui méritent examen. Comment organiser au mieux les données microéconomiques qu'ils génèrent ? Quel style de programmation adopter pour faciliter leur développement et leur maintenance ? Y-a-t-il des moyens de réduire la variabilité stochastique des résultats qui constitue l'un des principaux désavantages de la méthode ? Comment calibrer les paramètres du modèle lorsque celui-ci fait intervenir des paramètres individuels inobservables ? On a essayé de proposer une introduction à ces diverses questions, ilustrée par des exemples concrets de mise en œuvre sous R. On a vu qu'il présente plusieurs avantages : il permet la rédaction de programmes assez compacts, avec un mode de manipulation de données qui est celui auquel sont habitués les utilisateurs de logiciels statistiques, et il s'agit d'un logiciel libre ce qui garantit la portabilité maximale aux programmes réalisés dans ce langage.

Le fait qu'il s'agisse d'un logiciel statistique présente un autre intérêt. Le recours à la microsimulation requiert également beaucoup d'opérations purement statistiques aussi bien en amont qu'en aval du modèle. En amont du modèle figure en général un travail assez important de mise en forme des données de base issues d'enquêtes ou de sources administratives. Ces données doivent être mises au format désiré pour la microsimulation, en les complétant au besoin par des variables non directement mesurées mais nécessaires au fonctionnement du modèle. Egalement en amont de la simulation se trouve tout un travail d'évaluation de paramètres : matrices de transition entre états, modélisations du lien entre ces probabilités de transition et

diverses variables explicatives, ou encore estimation des équations de comportement qui seront incorporées à l'outil de simulation. Tous ces travaux nécessitent de passer par un logiciel statistique. Il y a des économies d'échelle à utiliser le même outil tout au long de la chaine de construction et d'utilisation du modèle.

Il est également pratique de conserver le même outil en aval des simulations, pour l'exploitation des résultats. Les résultats de la microsimulation se présentent en effet comme des fichiers de données micro, à exploiter comme on le fait pour n'importe quel panel ou fichier d'enquête, donc en s'appuyant à nouveau sur un logiciel statistique. On a vu enfin en section 2.10 l'intérêt du recours à un logiciel statistique lorsqu'on a affaire à des paramètres qui ne peuvent être estimés hors modèle : cette question du calibrage de paramètres inobservés est un problème encore peu traité à ce jour, mais il est certain qu'il est plus facile de l'aborder avec un logiciel déjà adapté au calibrage de modèles plus simples.

D'autres critères peuvent néanmoins faire opter pour des langages plus spécifiquement dédiés à la microsimulation. On a mentionné les deux exemples de logiciels ModGen et Liam 2, et l'existence de langages plus spécialement dédiés aux modèles dits d'agents. Ceci étant, même pour des utilisateurs préférant se tourner vers ces instruments, le détour par R aura pu aider à mieux comprendre l'esprit de la méthode et les précautions qu'elle appelle, qui sont les mêmes quel que soit le logiciel de programmation adopté.

Bibliographie

Albouy V., Bouton F., Courtioux P., Lapinte A., Le Minez S., Pucci M. (2003) « Les effets redistributifs et les avantages familiaux du système socio-fiscal en 2002 », *France, portrait social*, éd. 2003-2004.

Allègre, G., Mélonio, T. et Timbeau, X. (2012) « Dépenses publiques d'éducation et inégalités : une perspective de cycle de vie, *Revue Economique*, 63 (6) : 1055-1079

Ardilly, P. (2006) *Les techniques de sondage*, Technip, Paris.

Aubert P., Duc C. et Ducoudré B. (2010) « Le modèle PROMESS : Projection « méso » des âges de cessation d'emploi et de départ à la retraite », Document de travail, Drees, série Études et Recherches, n° 102.

Bachelet, M., Beffy, M. et Blanchet, D. (2011) « Projeter l'impact des réformes des retraites sur l'activité des 55 ans et plus : une comparaison de trois modèles », *Économie et Statistique*, 441-442 : 123-143.

Bachelet, M., Leduc, A. et Marino, A. (2014) « Les biographies du modèle Destinie II : rebasage et projection », *Document de travail, Insee/Dese* n° G2014/01.

Bacon, B. et Pennec, S. (2009) « Microsimulation, macrosimulation: model validation, linkage and alignment », *International Microsimulation Conference*, Ottawa.

Ballot G. (2002) « Modelling the Labour Market as an Evolving Institution : Model ARTEMIS », *Journal of Economic Behavior and Organization*, vol. 49, n° 1, pp. 51-77.

Ballot, G., Kant, J.-D. et Goudet, O. (2013) « Un modèle multi-agents du marché du travail français, outil d'évaluation des politiques de l'emploi. L'exemple du contrat de génération », texte présenté à la Journée d'Etude Microsimulation Acoss-Cnaf-Erudite-Insee, 23 mai.

Bardaji J., Sédillot B. et Walraet E. (2003) « Un outil de prospective des retraites : le modèle de microsimulation Destinie », *Économie et Prévision*, n° 160-161, pp. 193-213.

Barlet, M., Blanchet, D. et Le Barbanchon, T. (2009) « Microsimulation et modèles d'agents : une approche alternative pour l'évaluation des politiques d'emploi », *Économie et Statistique*, 429-430 : 51-77

Benoteau, I. (2011) « Développpement d'un modèle de microsimulation du fonctionnement du marché du travail », Mémoire ENSAE 3eme année.

Billari, F. et Fürnkranz-Prskawetz, A. (2003) *Agent-Based Computational Demography*, Physica Verlag.

Blanchet D. et Crenner E. (2010) « Le bloc retraites du modèle Destinie 2 : guide de l'utilisateur », *Document de travail, Insee/Dese*, n° G2010/14.

Blanchet, D., Buffeteau, S., Crenner, E. et Le Minez, S. (2011) « Le modèle de microsimulation Destinie 2 : principales caractéristiques et premiers résultats », *Économie et Statistique*, 441-442 : 101-121

Blanchet, D., Bozio, A. et Rabaté, S. (à paraître) *Le modèle de microsimulation PENSIPP - version 0.0*, Guide Méthodologique, Institut des Politiques Publiques.

Bonnet, C. et Mahieu, R. (2000) « Public pensions in a dynamic microanalytic framework: the case of France », in Mitton, L., Sutherland, H. and Weeks, M. (eds) *Microsimulation modelling for policy analysis: challenges and innovations*, Cambridge: Cambridge University Press.

Bourguignon F., Chiappori P.A. et Sastre- Descals J. (1988) « Sysiff : a Microsimulation Program of the French Tax-Benefit System » in A. Atkinson et H. Sutherland, *Tax benefit models*, STICERD-London.

Bourguignon, F., Robilliard, A.S. et Robinson, S. (2008) « Examining the social impact of the Indonesian financial crisis using a macro-micro model », Open access publications, Université Paris-Dauphine.

Bozio, A., Dauvergne, R., Fabre, B., Goupille, J. et Meslin, O. (2012) *Fiscalité et redistribution en France 1997-2102*, Rapport, Institut des Politiques Publiques.

Bozio, A., Fabre, B., Goupille, J. et Laffeter, Q. (2012) *Le modèle de microsimulation TAXIPP - version 2.0*, Guide Méthodologique, Institut des Politiques Publiques.

Breuil-Genier, P. (1998) « Les enseignements théoriques et pratiques des microsimulations en économie de la santé », *Économie et statistique*, 315 : 73-94

Cai L., Creedy, J. et Kalb, G. (2006) « Accounting for Population Ageing in Tax Microsimulation Modelling by Survey Reweighting », *Australian Economic Papers*, 45 (1) : 18 - 37.

Chardon, O. et Blanpain, N. (2010) *Projections de population 2007-2060 pour la France métropolitaine*, Insee Résultats, série Société, n° 107.

Cirillo, P., Bianchi, C.L., Gallegati, M. et Vagliasindi, P. (2006) « Validating and calibrating agent-based models: a case study », *Computational Economics*, 30(3): 245-264.

Cogneau D. et Robilliard A.-S. (2007) « Growth, distribution and poverty in Madagascar: Learning from a microsimulation model in a general equilibrium framework », in A. Spadaro (ed) *Microsimulation as a tool for the evaluation of public policies* : *methods and applications*, chapitre 3, Fundacion BBVA, Bilbao.

Colander D., Howitt P., Kirman A., Leijonhufvud A. et Mehrling P. (2008) « Beyond DSGE Models: Toward an Empirically Based Macroeconomics », *American Economic Review*, Papers and Proceedings, vol. 98, n° 2, pp. 236-240.

Courtioux, P., Grégoir, S. et Houeto, D. (2011) « Enseignement supérieur et durées de subvention individuelle implicite : une analyse par microsimulation dynamique », *Revue Economique*, 62(5) : 835-866.

Courtioux, P. (2012) « How income contingent loans could affect the returns to higher education: a microsimulation of the French case », *Education Economics*, 20 (4) : 402-429.

Cumpston, J.R. (2010) « Alignment and matching in multi-purpose household microsimulations » , *International Journal of Microsimulation*, 3(2): 34-45

Davies, J.B. (2009) « Combining Microsimulation with CGE and Macro Modelling for Distributional Analysis in Developing and Transition Countries », *International Journal of Microsimulation* , 2(1): 49–65.

Dawid, H et Fagiolo, G. (2008) « Agent-based models for economic policy design: introduction to the special issue », *Journal of economic behavior and organization*, 67 : 351-354.

Debrand, T. et Sorasith, C. (2010) « Apports du modèle de microsimulation Arammis : une analyse des effets redistributifs du plafonnement des restes à charge en ambulatoire », *Questions d'Économie de la Santé*, Irdes, n° 159.

Deissenberg C., van der Hoog S. et Dawid, H. (2007) « EURACE: a Massively Parallel Agent- Based Model of the European Economy », draft.

Division RPS (1999) « Le modèle de microsimulation dynamique DESTINIE », *Document de travail Insee/Dese,* n° G99/13.

Dormont, B., Grignon, M. et Huber, H. (2006) « Health expenditure growth: reassessing the threat of ageing », *Health Economics*, 15(9) : 947-963.

Duc, C., Lequien, L., Housset, F. et Plouhinec, C. (2013) « Le modèle de microsimulation Trajectoire : trajectoire de carrières tous régimes », *Documents de travail de la Drees, série Sources et méthodes,* n° 40.

Duée M. (2005) « La modélisation des comportements démographiques dans le modèle de microsimulation Destinie », *Document de travail, Insee/ Dese*, n° G2005/15.

Duée, M. et Rebillard, C. (2004) « La dépendance des personnes âgées : une projection à long terme », *Document de travail, Insee/ Dese*, n° G2004/02.

Dupont, G., Hagnéré, C. et Touzé, V. (2004) « Les modèles de microsimulation dynamique dans l'analyse des réformes des systèmes de retraites : une tentative de bilan », *Économie et Prévision*, 160-161 (4-5): 167-192.

Fagiolo G., Dosi, G. et Gabriele, R. (2004) « Matching, Bargaining and Wage-Setting in an Evolutionary Model of Labor Market and Output Dynamics », *Advances in complex systems*, n° 14, pp. 237-273.

Fagiolo G., Windrum, P. et Moneta, A. (2006) « Empirical Validation of Agent-Based Models: a Critical Survey », *LEM working paper series*, Sant'Anna school of advanced studies, n° 2006/14.

Fagiolo, G., Birchenhall, C. et Windrum, P. (2007) « Empirical validation in agent-based models: introduction to the special issue », *Computational Economics*, 30(3)

Fagiolo, G., Moneta, A. et Windrum, P. (2007) « A critical guide to empirical validation of agent-based models in economics: methodologies, procedures, and open problems », *Computational Economics*, 30(3) 195-226.

Franke, R. (2009) « Applying the method of simulated moments to estimate a small agent-based asset pricing model », *Journal of Empirical Finance*, 16(5) : 804-815.

Gaffard, J.L. et Napoletano, M., Eds (2012) *Agent based models and economic policy*, Revue de l'OFCE/debates and Policies, n° 124.

Gargiulo, F., Ternes, S., Huet, S., and Deffuant, G. (2010) « An iterative approach for generating statistically realistic populations of households », *PLoS ONE*, 5(1).

Geoffard, P.Y. et de Lagasnerie, G (2012) « Réformer le système de remboursement pour les soins de ville, une analyse par microsimulation », *Économie et Statistique*, 455-456: 89-113

Gilbert, N. (2008) *Agent-based models*, Series: quantitative applications in the social sciences, Sage Publications

Givord, P. (2010) « Méthodes économétriques pour l'évaluation de politiques publiques », *Document de travail, Insee/Dese*, n° G2010/08.

Gourieroux, C., Monfort, A., Renault, E. et Trognon, A. (1985) « Résidus généralisés, résidus simulés et leur utilisation dans les modèles non linéaires », *Annales de l'Insee*, 59/60 : 71-96

Gourieroux, C., Monfort, A. et Renault, E. (1993) « Indirect inference », *Journal of Applied Econometrics*, 8, S85–S118.

Grazzini, J. (2011) « Consistent estimation of agent-based models », WP n°1 110, Department of Economics, Université de Turin.

Hartig, F., Calabrese, J., Reineking, B., Wigand, T. et Huth, A. (2011) « Statistical inference for stochastic simulation models – theory and application », *Ecology Letters*, 14 : 816–827

Harland, K., Heppenstall, A., Smith, D., and Birkin, M. (2012) « Creating realistic synthetic populations at varying spatial scales: A comparative critique of population synthesis techniques », *Journal of Artificial Societies and Social Simulation*, 15(1):1.

Heathcote, J., Storesletten, K. et Violante, G.L. (2009) « Quantitative macroeconomics with heterogenous agents », draft.

Huet, S. et Deffuant, G. (2011) « Common framework for the microsimulation model in Prisma project ». Technical report, Cemagref, Lisc.

Kirman, A. (1992) « What or whom does the representative individual represent? », *Journal of Economic Perspectives*, vol. 6(2) : 117-36.

Klevmarken, N. A. (1998) « Statistical Inference in Micro Simulation Models: Incorporating external information », Working Paper Series 1998:20, Uppsala University, Department of Economics.

Lachaud C., Largeron C. et Rochaix L. (1998). « Franchise sur les soins ambulatoires et équité sociale », *Économie et Statistique*, 315 : 51-72.

Lafaye de Micheaux, P., Drouillet, R. et Liquet, B. (2011) *Le logiciel R : maitriser le langage, effectuer des analyses statistiques*, Coll. Statistiques et Probabilités Appliquées, Springer.

Landais, C., Piketty, T. et Saez, E. (2011) *Pour une révolution fiscale: un impôt sur le revenu pour le XXIème siècle,* Coll. La république des idées, Le seuil.

LeBaron B. et Tesfatsion L. (2008) « Modelling Macroeconomics as Open-Ended Dynamic Systems of Interacting Agents », *American Economic Review*, Papers and Proceedings, vol. 98, n° 2, pp. 246-250.

Legendre F., Lorgnet J-P., Thibault F. (2001). « La redistribution au bénéfice des familles : l'apport de Myriade », *Recherches et prévisions*, n° 66.

Legendre F., Lorgnet J.P. et Thibault F. (2003) « Que peut-on retenir de l'expérience française en matière de microsimulation ? », *Économie et Prévision*, n° 160-161, I-XIV.

Lenormand, M., Jabot, F. et Deffuant, G. (2012) « Adaptive approximate bayesian computation for complex models », draft

Lenormand, M. et Deffuant, G. (2012) « Generating a synthetic population of individuals in households: sample-free vs sample-based methods », IRSTEA, draft

L'Horty, Y. et Petit, P. (2010) « Evaluation aléatoire et expérimentation sociale », Document de travail EPEE 2010-2.

Li, J. et O'Donoghue, C. (2011) « Evaluating Alignment Methods in Dynamic Microsimulation Models », *3rd International Microsimulation Conference*, Stockholm.

Li, J. et O'Donoghue, C. (2012) « A methodological survey of dynamic microsimulation models », UNU-MERIT Working Paper Serie, n° 2012-002.

Li, J. (2011) *Dynamic microsimulation for public policy analysis*, Ph D Dissertation, Maastricht University.

Liégeois, P. et Dekkers, J. (2011) « Combining Euromod and LIAM tools for the development of dynamic cross-sectional microsimulation models: a sneak preview », 3rd General Conference of the International Microsimulation Association, Stockholm, June 8th-10th.

Maindonald, J. et Braun, W.J. (2010) *Data analysis and graphics using R*: *an example-based approach*, Cambridge University Press.

Mason, C. (2011) « SOCSIM oversimplified », Draft, University of Berkeley.

Mannion, O., Lay-Yee, R., Wrapson, W., Davis, P. and Pearson, J. (2012) « JAMSIM: a microsimulation modlling policy tool », *Journal of Artificial Societies and Social Simulation*, 15 (1)

Marbot, C. et Roy, D. (2012) « Projections du coût de l'APA et des caractéristiques de ses bénéficiaires à l'horizon 2040 à l'aide du modèle Destinie », *Document de travail, Insee/Dese*, n° G2012/10.

Morand, E., Toulemon, L., Pennec S., Baggio R., and Billari F. (2010) « Demographic modelling: the state of the art », *SustainCity Working Paper*, 2.1a, Ined, Paris.

Morrison, R. (2006) « Make it so: event alignment in dynamic microsimulation », DYNACAN paper.

Nguyen, M.K., Phelps, S. et Ng, W.L. (2011) « Empirical likelihood estimation of agent-based models », draft

Neufeld, C., (2000) « Alignment and Variance Reduction in DYNACAN » in Anil Gupta and Vishnu Kapur (eds) *Microsimulation in Government Policy and Forecasting.* North- Holland.

Neugart M. (2008) « Labour Market Policy Evaluation with ACE », *Journal of Economic Behaviour and Organization*, n° 67, pp. 418-430.

O' Donoghue, C., Hynes, S. et Lennon, J. (2009) « The Life-cycle income analysis model (LIAM): a study of dynamic microsimulation modelling computing framework », *International Journal of Microsimulation*, 2(1) : 16-31.

Nikolai, C. and Madey, G. (2009) « Tools of the trade: a survey of various agent-based modelling platforms », *Journal of Artificial Societies and Social Simulation*, 12 (2).

Orcutt G.H. (1957) « A New Type of Socio- Economic System », *The Review of Economics and Statistics*, vol. 39, n° 2, pp. 116-123.

Post, W. et Van Imhoff, E. (1997) « Méthodes de microsimulation pour des projections de population », *Population*, 52 (4) : 889-932

Poubelle, V. *et al.* (2006) « Prisme, le modèle de la Cnav », *Retraite et société,* 48 : 202-215.

R Development Core Team (2011). *R : A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http ://www.R-project.org/.

Robilliard A.-S., Bourguignon F. et Robinson S. (à paraître) « Crisis and Income Distribution: a Micro-Macro Model for Indonesia », in F. Bourguignon et L. Pereira da Silva, (eds) *The Impact of Macroeconomic Policies on Poverty and Income Distribution*, World Bank et Oxford University Press, New York.

Rutter, C.M., Miglioretti, D. et Savarino, J.E. (2009) « Bayesian Calibration of Microsimulation Models », *Journal of the American Statistical Association*, 104 : 1338-1350

Schelling, T.J (1971) « Dynamic Models of Segregation », *Journal of Mathematical Sociology*, Vol. 1.

Spadaro, A. (Ed) (2007) *Microsimulation as a tool for the evaluation of public policies : methods and applications*, Bilbao: fundaçion BBVA.

Tillé, Y. (2001) *Théorie des sondages : échantillonnage et estimation en populations finies*, Dunod.

Willekens, F.J.C. (2009) « Continuous-time microsimulation in longitudinal analysis », in A. Zaidi, A. Harding et P. Williamson (dir.), *New frontiers in microsimulation modeling*, Farnham, Ashgate : 353-376.

Willekens, F.J.C. (2011) « La microsimulation dans les projections de population », *Cahiers Québecois de Démographie*, 40 (2) : 267-297.

Windrum, P., Fagiolo, G. et Moneta, A. (2007) « Empirical validation of agent-based models: alternatives and prospects », *Journal of artificial societies and social simulation*, 10 (2) : 8.

Winker, P. et Gilli, M. (2001) « Indirect estimation of the parameters of agent-based models of financial markets », *FAME Working Paper,* No. 38.

Wolf, D. (2001) « The role of microsimulation in longitudinal data analysis », *Canadian Studies in Population*, 28(2) : 313-339.

Zaidi, A. et Rake, K. (2001) « Dynamic microsimulation models: a review and some lessons from SAGE », Sage discussion paper n° 2

Zinn, S., Himmelspach, J., Gampe, J. et Uhrmacher, A.M. (2009) « MIC-CORE : a tool for microsimulation », in *Proceedings of the 2009 Winter Simulation Conference: December 13-16, 2009*, Austin, Texas,

Piscataway (NJ), IEEE.

Zuchelli, E., Jones, A.M. et Rice, N. (2012) « The evaluation of health policies through dynamic microsimulation methods », *International Journal of Microsimulation*, 5(1): 2-20.

Annexe 1: Manipulation des vecteurs, filtres et masques

On se limitera pour simplifier au cas des vecteurs monodimensionnels, qui sont des vecteurs colonnes déclarés par une instruction de type

```
x <- numeric(1000)
```

Le contenu d'un tel vecteur peut être renseigné par la fonction `c()` (pour « colonne ») dont les arguments seront la suite d'éléments du vecteur, par exemple

```
x <- c(1,2,3,2,3)
```

Après une telle affectation, la longeur effective du vecteur n'est plus que de 5, à laquelle on accède par l'instruction `length(x)`.

L'appel de l'élément particulier d'un vecteur se fait par `x[i]` : `x[2]` vaudra 2. Mais il est en général préférable de manipuler les vecteurs de manière globale. Les principaux opérateurs appliqués à des vecteurs ou des combinaisons de vecteur sont les opérateurs arithmétiques, qui s'appliquent terme à terme et les opérateurs logiques.

Par exemple :

```
y <- x==2
```

donne à `y` le contenu `(FALSE, TRUE, FALSE, TRUE, FALSE)`. On comprend donc pourquoi l'instruction `sum(x==2)` permet le comptage des observations vérifiant `x==2`, la commande `sum` traitant les valeurs `TRUE` et `FALSE` comme respectivement égales à 1 et 0.

On peut également extraire des groupes d'éléments de vecteurs, soit par indiçage direct, soit par « masque logique ». Les deux instructions :

```
x[c(3,5)] <- 7
x[x==3] <- 7
```

conduiront toutes deux à donner à `x` la séquence de valeurs `(1,2,7,2,7)` en remplaçant le contenu des éléments 3 et 5 mais sans changement du reste du vecteur. Cette extraction peut aussi s'appliquer du côté droit du symbole d'affectation, mais avec un impact un peu différent, puisque le résultat sera de prélever les éléments ayant les indices ou vérifiant la condition qui sont fournis pour en placer le résultat dans le membre de gauche. Par exemple :

```
y <- x[c(2,4)]
y <- x[x==2]
```

sont deux instructions à la suite desquelles y sera le vecteur `(2,2)`. Ce faisant, l'indiçage est modifié : le deuxième élément du vecteur d'origine `x` est devenu le premier élément du nouveau vecteur `y`.

Une alternative à l'utilisation du masque logique est le recours à la fonction `which` qui retourne les indices pour lesquels une condition sur une ou plusieurs variables est remplie. Ainsi, l'instruction :

```
y <- which(x==2)
```

affecte à y le couple de valeurs (2,4). De ce fait, il y a équivalence des instructions :

```
y <- x[x==2]
y <- x[which(x==2)]
```

et l'instruction `length(which(x==2))` sera équivalente à `sum(x==2)`.

Lorsqu'on veut procéder à des sélections sur la base de filtres complexes, il faut enfin faire attention à différencier les `&` (et) ou au `|` (ou) simples et les `&&` et `||` également disponibles en R. Ce sont les `&` et le `|` simples s'appliquent termes a termes. Si on a affecté

```
z <- c(0,1,0,0,1)
```

alors l'instruction

```
y <- x[x==1 | z==1]
```

affecte à `y` la série de données `(1,2,3)`.

Ces différentes modalités de filtrage seront particulièrement utiles en microsimulation dynamique puisque celle-ci, pour l'essentiel, consiste à appliquer des séries de transformations successives à des sous populations filtrées selon diverses caractéristiques. Pour ces opérations de sélection, on pourra aussi tirer parti d'opérateurs ensemblistes permettant de recombiner les résultats de ces filtres, tels que l'union, l'intersection, la différence ou la différence symétrique.

Annexe 2 : variables locales, variables globales et superaffectation

Pour illustrer les notions de variables globales et locales et la façon dont elles sont gérées par les fonctions, on peut partir de la séquence suivante éxécutée en mode interactif de l'environnement RStudio :

```
> x <- 2
> x [1]
2
> f <- function(y) {x <- x**2; print (x); return (y +x)}
> z <- f(3)
[1] 4
> z
[1] 7
> x
[1] 2
```

Dans cet exemple, on commence par définir une variable globale `x` à laquelle on affecte la valeur 2. La fonction `f` utilise d'une part la variable interne `y` qui est son unique argument d'appel, et elle a aussi accès à la variable globale externe `x`. Elle peut même la modifier. L'affichage de `x` après mise au carré dans la fonction retourne bien la valeur 4 qui, additionnée à l'argument d'appel `y=3`, retourne bien `z=y+x**2=7`, mais une nouvelle demande d'affichage de `x` après sortie de la fonction redonne la valeur initiale de `x`. Sa modification dans la fonction `f` n'a été que temporaire, et elle a en fait porté sur une copie locale de `x`.

Avec ce mode de programmation des fonctions, la seule façon de modifier une variable du programme appelant est d'y affecter les valeurs de retour de ces fonctions. C'est ce mode de fonctionnement qui explique que les fonctions de la bibliothèque `OutilsMS.R` soient appelées dans des instructions du type :

```
liste <- permut(liste)
```

On aurait pu y préférer une formulation plus compacte du type `permut(liste)` qui aurait directement modifié l'ordre des éléments de `liste` sans qu'il y ait à rajouter une instruction de réaffectation modifiant le contenu de cette variable, mais ce format d'utilisation n'a pas que des désavantages puisqu'il permet d'écrire également :

```
liste_bis <- permut(liste)
```

au cas où on veut disposer d'une liste permutée tout en conservant la liste d'origine.
Néanmoins, dans le cas général, n'avoir que cette manière de modifier les variables globales serait très bloquant. On a besoin de fonctions modifiant directement des variables globales. Par exemple, dans le bloc retraites du modèle Destinie, la fonction la plus utilisée est la fonction de simulation des droits directs de retraite `SimDir(i)` qui, appliquée à un individu `i`, effectue directement un ensemble d'opérations complexes touchant à un grand nombre de caractéristiques de cet individu, sans que celles-ci ne soient mentionnées comme arguments d'appel ou de retour :

- Si l'individu n'est pas encore retraité, elle simule son départ en retraite éventuel en fonction du niveau de la retraite qui lui est offert à cet âge et, si liquidation il y a, elle modifie à la fois le statut de l'individu (il passe à la catégorie inactif), son salaire ou ses prestations chômage (mis à zéro) et elle renseigne les diverses composantes de sa retraite selon son profil de carrière antérieur (pension du régime général, pensions complémentaires, pension du régime de la fonction publique...)
- Si l'individu est déjà retraité, elle simule l'évolution de ces différentes composantes de la pension selon les règles d'indexation qui ont été retenues.

La commande R qui permet ce type de manipulation est la superaffectation qui permet aux fonctions de modifier les variables globales de manière permanente et pas seulement temporaire. Si on reprend la séquence de départ avec cette instruction de superaffectation notée <<−, on peut par exemple avoir :

```
> x <- 2
> z <- 1
> x
[1] 2
> z
[1] 1
> f <- function (y) {x <<- x**2; z <<- y +x}
> f(3)
> x
[1] 4
> z
[1] 7
```

Cette fois-ci, la fonction modifie bien les deux variables globales `x` et `z`. Dans cet exemple, cette fonction n'a d'ailleurs aucun argument de retour, son action se limitant à ces modifications de variables globales, mais on aurait pu évidemment combiner les deux, par exemple utiliser la super-affectation pour `x` et utiliser le renvoi par `return` pour modifier la valeur de `z`. Ces deux possibilités sont à panacher en fonction des besoins.

Annexe 3 : Procédures graphiques

Les codes sources pour les appels de graphiques n'ont pas été reproduits dans les exemples de programmes donnés au fil du texte, pour alléger leur présentation. On détaille ici ces appels, qui ne donnent qu'un aperçu des possibilités graphiques de R. L'instruction de base est l'instruction `plot` qui graphique un vecteur en fonction de son indice. En fin du programme `exemple_1.R`, l'instruction `plot(txcho)` donne ainsi le graphique du taux de chômage simulé en fonction de la date. Néanmoins, cette instruction ne permet de tracer qu'une seule série à la fois : lorsqu'un deuxième série lui est fournie en argument, c'est pour l'obtention d'un graphique traçant cette seconde série en fonction de la première. Pour un graphique superposant plusieurs séries, on doit recourrir à la fonction auxiliaire `points`, en faisant suivre l'appel de `plot` d'autant d'appels de `points` que de séries supplémentaires à ajouter. Ainsi, pour l'obtention du graphique 1 incluant également la trajectoire théorique du chômage dérivée de l'équation (1), on écrit :

```
plot (txcho,ylim=c(3,10),type="l",lwd=3,xlab="",ylab="Taux de chômage en %")
for (t in 1:25)
{
  txcho[t] <- 100*(proba_entree_chomage/
                   (proba_sortie_chomage+proba_entree_chomage)+
                (.1-proba_entree_chomage/
                   (proba_sortie_chomage+proba_entree_chomage))*
              exp(-(proba_entree_chomage+proba_sortie_chomage)*(t-1)))
}
points (txcho,type="l",lwd=1)
```

Dans cette série d'instructions, on notera que le premier appel de `plot` inclut aussi une définition des bornes du graphique en ordonnées, qui sont un des nombreux arguments optionnels de cette fonction. On précise également le type de ligne (`type= "l"` pour une ligne continue), son épaisseur (`lwd=3`) et des labels autres que les labels par défaut pour les axes des x et des y.

La combinaison des fonctions `plot` et `points` est utilisée de manière encore plus systématique pour le graphique 4 et plus particulièrement le graphique 4a qui superpose les résultats de 10 simulations successives et de leur moyenne traitée comme une onzième simulation. Le code à adjoindre au programme `exemple_4.R` est le suivant :

```
# Sortie premier graphique
par(mfrow=c(1,2))
plot (txcho[1,],type="l",ylim=c(3,12),type="l",lwd=1,xlab="",ylab="Taux de chômage en %")
for (sim in 2:10)
{
  points (txcho[sim,],type="l",lwd=1)
}
points (txcho[11,],type="l",lwd=4)

# Graphique direct de la simulation sur 10000 individus
plot (simul(10000),type="l",ylim=c(3,12),type="l",lwd=3,xlab="",ylab="")
```

Ce deuxième exemple illustre par ailleurs une option graphique supplémentaire activée dans la fonction `par`, qui est une fonction générale de définition des paramètres de graphiques qui suivent. Cette option est la possibilité d'afficher plusieurs graphiques sur une même page. C'est l'objet de l'affectation `mfrow =c(1,2)`

qui indique à R de placer les graphiques qui suivent, ligne par ligne, dans une grille de dimensions 1x2. De manière plus générale, l'affectation `mfrow=c(n,m)` (resp. `mfcol=c(n,m)`) indique à R de subdiviser la zone graphique en `n` lignes et `m` colonnes, et de remplir ces cellules ligne par ligne (resp. colonne après colonne) avec les résultats des `n×m` demandes de graphiques suivantes.

On notera par ailleurs que le graphique de la seconde simulation effectuée directement sur 10 000 individus, est établi directement en une seule instruction sans stockage des résultats de `simul` dans un tableau intermédiaire de taux de chômage par date, stockage qui est de fait superflu. Sous l'environnement de développement RStudio présenté en annexe 3, il est d'ailleurs tout à fait possible de relancer directement cette commande de manière interactive, dès lors que la fonction `simul` est stockée en mémoire. Ceci permet une exploration rapide de la sensibilité des résultats aux différents paramètres de cette fonction `simul`.

Les possibilités de graphiques 3D ont été ensuite illustrées par les figures 7 et 8, créés par la séquence suivante, à partir des vecteurs `grille_proba` et `grille_alpha` pour les coordonnées en x et y et des matrices `score`, `anc_cho_moy` et `anc_emp_moy` pour les variables $z(x, y)$ correspondantes :

```
contour(grille_proba,grille_alpha,log(score),levels=seq(-10,0,by=.5))
par(mfrow=c(1,2),cex.lab=0.8,adj=0)
persp(grille_proba,grille_alpha,anc_cho_moy,theta=45)
persp(grille_proba,grille_alpha,anc_emp_moy,theta=45)
```

Annexe 4 : Accès au langage et à un environnement de développement intégré

Faire tourner l'ensemble des programmes présentés dans cette note nécessite de disposer d'une version de R et d'un environnement de développement permettant la saisie et l'exécution de programmes. Le téléchargement de R peut se faire depuis le site `http ://cran.r-project.org/`. Le développement peut ensuite se faire depuis l'environnement fourni par défaut livré avec le langage mais il existe des environnements de développement plus conviviaux. L'exemple donné page suivante reproduit la copie d'écran d'une session de travail effectuée à l'aide d'un environnement de cette nature, l'environnement RStudio téléchargeable à l'adresse `http ://www.rstudio.org/`. Cet environnement propose quatre fenêtres :

- La fenêtre supérieure gauche permet d'éditer un ou plusieurs fichiers de code source R,
- La fenêtre située juste en dessous est la fenêtre « console » permettant d'exécuter des commandes R de manière interactive.
- La fenêtre supérieure droite récapitule les différents objets R créés lors de la session en cours (ou récupérés de la session précédente) : il s'agit soit de tableaux ou variables, soit de fonctions : selon le cas, les renseignements donnés sont la valeur de la variable, sa dimension ou, pour les fonctions, la liste de leurs arguments d'appel et leurs valeurs par défaut.
- La fenêtre inférieure droite est enfin celle ou s'affichent les graphiques (mais elle peut aussi donner d'autres informations, c'est notamment dans cette fenêtre que s'affiche l'aide en ligne lorsqu'une aide sur une commande R est demandée, par `help(commande)` depuis la fenêtre console.

L'exécution d'un programme nécessite que l'ensemble de ses instructions aient été rassemblées dans un fichier d'extension .R (ici `Exemple_1.R`). On lance ce programme depuis la fenêtre d'édition à l'aide de la séquence de menus `Edit/Run code/Run all`. L'ensemble des lignes du programmes sont alors déroulées automatiquement dans la fenêtre console, où viennent s'afficher par défaut les résultats dont on a demandé l'impression (il n'y en a pas dans l'exemple proposé) et/ou les messages d'erreurs.

Si l'exécution s'est finie sans erreur, les différentes variables générées par le programme restent ensuite présentes en mémoire dans l'état final où les laisse le programme. Si on le souhaite, il est toujours possible de continuer à les traiter interactivement. Par exemple, les deux instructions qui apparaissent sur l'exemple en bas de cette fenêtre sont des instructions saisies manuellement qui affichent la séquence des taux de chômage des 10 premières périodes ainsi que le statut atteint en fin de programme par les 100 premiers individus de l'échantillon simulé : on voit que ce type d'impression est obtenu en tapant simplement le nom de l'objet que l'on souhaite voir imprimer.

Cette interactivité facilite le travail de mise au point : il est possible de ne faire tourner qu'une partie du programme actif dans la fenêtre d'édition, en sélectionnant les lignes qu'on souhaite exécuter et en utilisant la séquence de menu `Edit/Run code/Run line(s)`, puis d'analyser à la main les résultats issus de cette exécution partielle, ou de tester pas à pas l'effet des lignes de programme successives sur l'état de la population.

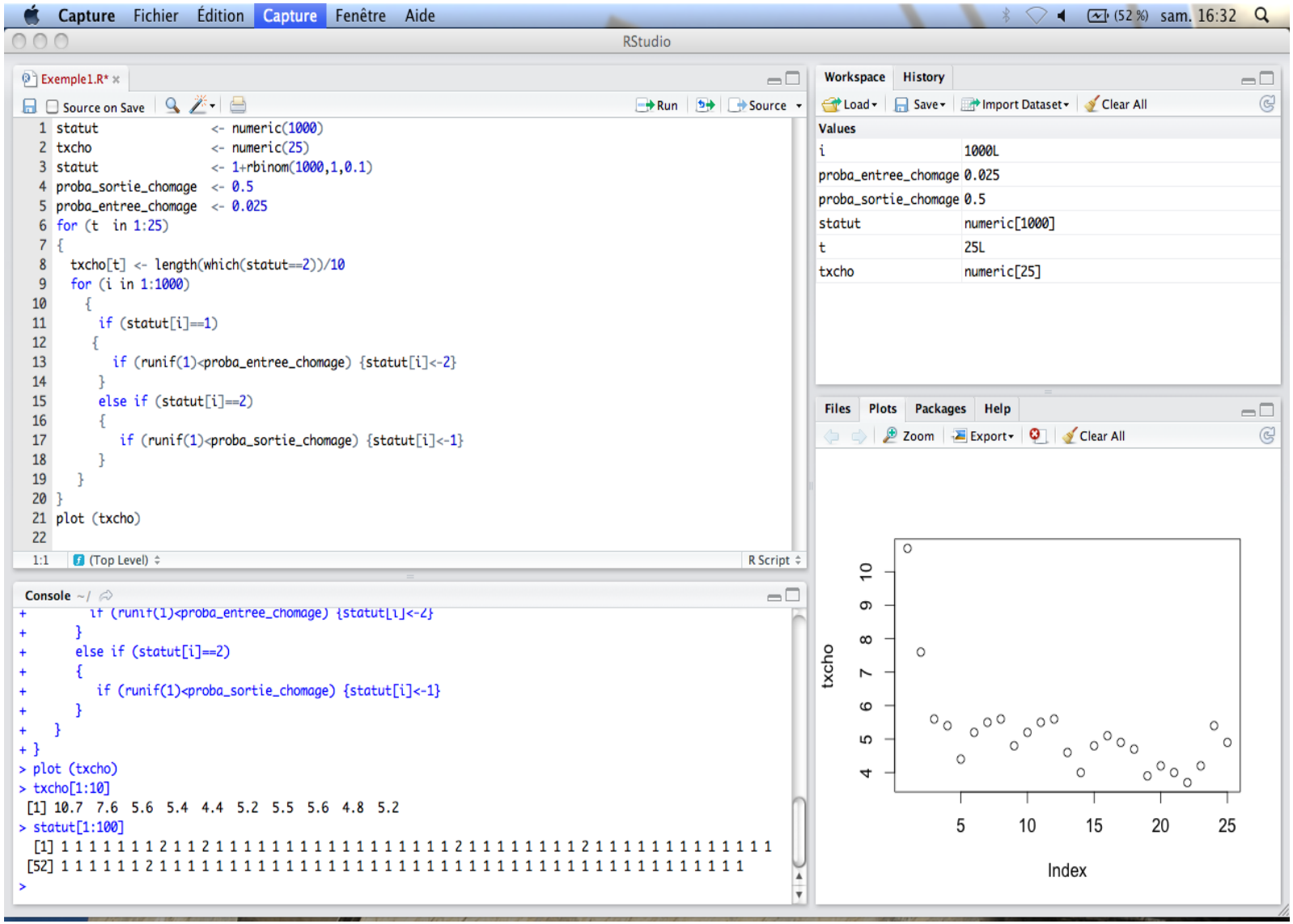


Figure 11: Exemple de sesssion RStudio : mise en œuvre du progamme `Exemple_1.R`

Série des Documents de Travail « Méthodologie Statistique »

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

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

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

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

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

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

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

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

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

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

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

C. LAGARENNE, C. THIESSET

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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