Recherche de l'allocation optimale du nombre de lignes à allouer par classe d'agrégation – Partie 1

Cheick Oumar Kouyaté
Chargé d'études actuarielles chez Allianz France

Résumé

Dans cette étude, notre objectif est de déterminer l'allocation optimale, c'est-à-dire celle qui est associée à la plus petite erreur de compression. Notre approche consiste à : 1) Formuler cette allocation optimale comme la solution d'un programme d'optimisation ; 2) Nous effectuons ensuite une série de transformations, comprenant un changement de variables, un changement d'échelle et une relaxation continue, pour convertir le programme initial, de nature discrète, en un programme continu. Cette première transformation introduit un premier biais d'estimation ; 3) Étant donné que les fonctions composant la fonction objectif du programme continu ne sont pas complètement connues en pratique, nous les interpolons linéairement à partir d'un nombre défini de valeurs intermédiaires, noté p. Cette interpolation entraîne une erreur d'interpolation dépendant de p; 4) Enfin, nous approximons les interpolations linéaires par des polynômes de Bernstein de degré n, transformant ainsi le programme continu en un programme continu et différentiable. Cette dernière étape induit une erreur d'approximation, dépendant de n.

En analysant les effets des hyperparamètres p et n sur les programmes d'optimisation, nous observons une convergence de la solution du programme continu et différentiable vers l'allocation optimale, l'augmentation de la valeur de p permettant une réduction de l'erreur d'interpolation et l'augmentation de la valeur de n permettant une réduction de l'erreur d'approximation. Ces résultats soulignent l'efficacité de notre approche et fournissent une méthode pratique pour la conception d'allocations optimales dans des contextes similaires.

Mots clés : Compression de données ; Allocation ; Optimisation différentiable ; Interpolation et Approximation
Abréviations : BEL : Best Estimate of Liabilities, ALM : Asset Liability Management, PM : Provision Mathématique

1. Introduction

Un des défis des modèles Actif-Passif en approche Full ALM[1] est d'arriver à traiter un volume important de données en préservant des temps d'exécution des projections compatibles avec les contraintes de clôture. En effet, pour des portefeuilles de grande taille, le volume de calculs peut vite devenir considérable : le temps et l'espace disque alloués sont alors des problématiques opérationnelles clés. Une solution est la compression des données, technique couramment utilisée pour regrouper les contrats dont les risques sous-jacents sont similaires. Le portefeuille est d'abord partitionné en sous-portefeuilles de contrats partageant des caractéristiques similaires. Ces sous-portefeuilles définissent des classes d'agrégation ou strates, au sens où seuls les contrats appartenant à un même sous-portefeuille peuvent être agrégés. Un algorithme est ensuite utilisé indépendamment au sein de chaque classe d'agrégation ou strate pour en réduire la taille (le nombre de lignes) en altérant le moins possible le volume de BEL[2]. Le facteur de réduction est appelé **taux de compression** et l'écart de BEL entre les données compressées et les données initiales est appelé **erreur de compression**.

[1] Le Full ALM est une approche utilisée dans le domaine de la gestion des actifs et passifs (ALM) dans le secteur de l'assurance. Elle modélise directement les interactions entre l'actif et le passif et prend en compte des éléments tels que le rachat conjoncturel, l'inflation, le taux de rendement des actifs et le taux de participation aux bénéfices. C'est une méthode qui permet d'obtenir des projections de cash-flows plus précises et détaillées.

[2] Le BEL ou Best Estimate of Liabilities est la meilleure estimation en vision économique des engagements futurs de l'assureur vis-à-vis de son portefeuille.

Préalablement à la compression, le nombre total de contrats souhaité en sortie est fixé pour satisfaire aux contraintes d'exécution (temps de calcul et espace de stockage). Par exemple, pour un portefeuille de 45 000 contrats, on pourra souhaiter après compression ne disposer que de 6 000 lignes. Mais l'allocation optimale de ces 6 000 lignes par strate, celle qui minimise l'erreur de compression globale, reste un enjeu important de qualité des projections de cash-flows.

Traditionnellement, cette allocation optimale est obtenue par proratisation par rapport à la Provision Mathématique (PM) de chaque strate. Cependant, cette approche, bien que largement utilisée, a montré ses limites en termes de précision et de pertinence. En effet, la proratisation par Provision Mathématique ne prend pas en compte les spécificités en termes d'hétérogénéité des contrats de chaque strate, ce qui peut conduire à une allocation non optimale des contrats. Par exemple, certaines strates du portefeuille peuvent présenter des structures plus hétérogènes que d'autres, nécessitant une allocation plus importante pour garantir une représentation adéquate. C'est dans ce contexte que cet article propose une solution opérationnelle novatrice.

En combinant des concepts de la théorie de l'approximation de fonctions et de l'optimisation différentiable, notre approche vise à optimiser l'allocation des contrats par strate afin de minimiser l'erreur de compression globale. L'approximation de fonctions permet d'estimer les courbes d'erreurs de compression en fonction du nombre de contrats alloués à chaque strate, tandis que l'optimisation différentiable permet de déterminer l'allocation optimale pour atteindre cet objectif.

2. Méthode traditionnelle

2.1 Formalisme mathématique

Les lignes du portefeuille initial avant compression sont des contrats et les lignes après compression sont appelées Model Points ou MP.

On note :

- G : le nombre total de classes d'agrégation ou de strates
- N : le nombre total de contrats ou de lignes
- K : le nombre total de MP souhaité en sortie après compression

Pour une strate g : N_g et k_g désignent respectivement le nombre de lignes avant et après compression. Ainsi, le taux de compression de la strate se définit par :

$$t_g = \frac{k_g - N_g}{1 - N_g} \quad \text{avec} \quad 1 \leq k_g \leq N_g \qquad (1)$$

On suppose que le nombre total de MP en sortie de compression K est fixé, et que les strates ont été définies. On doit alors choisir les tailles $k_1, k_2, \cdots, k_G$ à sélectionner dans chaque strate.

Dans le cas d'une strate exhaustive g, la taille choisie k_g est égale à la taille de la strate N_g et le taux de compression vaut 0, tous les MP sont d'office sélectionnés et aucune compression n'a lieu dans la strate.

Définition 2.1 – On appelle allocation tout vecteur à G composantes entières non-nulles et dont la g-ème composante n'excède pas la taille de la strate g et ce pour tous les g.

Notons $\mathbb{A}$ l'espace des allocations,

$$\mathbb{A} = \left\{k = (k_1, k_2, \cdots, k_G) \in \mathbb{N}^{*G} : \forall\, g \in 1, \ldots, G \quad 1 \leq k_g \leq N_g \right\}$$

Définition 2.2 **-** Une allocation k est dite *contrainte* ou *admissible* si la somme de ses composantes vaut K.

On note $\mathbb{A}_K$ le sous-espace de $\mathbb{A}$ des allocations admissibles,

$$\mathbb{A}_K = \left\{k = (k_1, k_2, \cdots, k_G) \in \mathbb{A} : \sum_{g=1}^{G} k_g = K \right\}$$

2.2 Les allocations proportionnelles

L'allocation par proratisation par rapport à la Provision Mathématique fait partie d'une catégorie plus vaste d'allocations appelée Allocations proportionnelles [1]. Soit $\mathcal{T}$ une variable ou une grandeur actuarielle pouvant servir à définir une notion d'importance relative entre les strates. Une allocation $k = (k_1, k_2, \cdots, k_G)$ est dite **au prorata de $\mathcal{T}$** si :

$$\forall g = 1, \ldots, G \quad k_g = \frac{\mathcal{T}_g}{\sum_{i=1}^{G} \mathcal{T}_i} * K \qquad s.c. \quad k_g \in \mathbb{N}\ et\ 1 \leq k_g \leq N_g$$

Autrement dit, plus la valeur de $\mathcal{T}$ pour la strate est grande, plus celle-ci se verra allouer un grand nombre de lignes. Quelques exemples de variables $\mathcal{T}$ traditionnellement choisies par les experts du métier pour la proratisation sont :

- Le nombre de lignes (N)
- La Provision Mathématique (PM)
- Le Best Estimate of Liabilities (BEL)

Remarque : Les allocations proportionnelles sont toujours admissibles $(k \in \mathbb{A}_K)$.

L'approche proportionnelle est une approche naïve de détermination de l'allocation optimale. Elle suppose que la qualité de la compression au sein d'une strate évolue avec la valeur de la variable de proratisation de la strate, d'où le fait d'accorder plus de lignes aux strates avec des poids élevés. Les allocations proportionnelles ont l'avantage d'être intuitives et assez simples à mettre en place, cependant elles n'ont aucune propriété remarquable hormis le fait d'être toujours admissibles. Toutefois, on peut s'imaginer que l'allocation optimale, si elle existe, n'est pas significativement éloignée d'une allocation proportionnelle si la variable de proratisation est bien choisie. L'allocation proportionnelle peut donc être un bon point de départ pour trouver l'allocation optimale. En d'autres termes, elles peuvent être de bonnes candidates pour l'initialisation de nos algorithmes d'optimisation.

3. Méthode proposée

Dans cette section, nous présentons notre approche pour déterminer l'allocation optimale. Nous allons écrire l'allocation optimale recherchée comme la solution d'un programme d'optimisation sous contrainte [2]. Ce programme sera ensuite transformé successivement en des programmes de plus en plus simplifiés afin d'en obtenir les solutions plus facilement.

3.1 L'allocation optimale comme solution d'un programme d'optimisation

Nous introduisons tout d'abord la fonction h_g, qui associe à un nombre de lignes alloué k_g l'erreur de compression pour la strate g. L'erreur globale de compression, notée H, est définie comme la somme des erreurs de compression individuelles pour chaque strate. Ainsi, H est déterminée par l'expression suivante :

$$H(k) = \sum_{g=1}^{G} h_g(k_g)$$

La Figure 1 illustre l'erreur de compression d'une strate en fonction du nombre de lignes alloué. Il peut être observé que l'erreur de compression diminue à mesure que le nombre de lignes alloué augmente, avec une erreur maximale atteinte lorsqu'une seule ligne est allouée à la strate, ce qui correspond à un taux de compression de 100%.

Définition 3.1 – L'allocation optimale k^*, si elle existe, est solution du programme d'optimisation suivant :

$$\min_{k \in \mathbb{A}_K} H(k) \qquad \text{Ou de façon équivalente} \qquad \min_{k \in \mathbb{A}} H(k) \quad s.c. \quad k_1 + k_2 + \cdots + k_G = K$$

Le programme d'optimisation dans sa forme actuelle est un problème discret ou combinatoire. Les problèmes de cette nature sont en effet très difficiles à traiter en raison de l'explosion combinatoire. Pour pallier cette difficulté, nous allons entreprendre une succession de transformations visant à reformuler le problème initial en un nouveau problème dont les solutions sont plus aisées à déterminer et convergent vers la solution du problème initial.

3.2 L'allocation optimale comme solution d'un programme d'optimisation continu

3.2.1 Changement de variables

Le Changement de variables est une transformation qui consiste à passer de l'espace des allocations à l'espace des taux de compression. Les vecteurs de $\mathbb{A}$ sont transformés composante par composante en leur équivalent en termes de taux de compression au moyen de la transformation bijective (1). Le nouvel espace sera nommé « projeté de $\mathbb{A}$ » et sera noté $Proj(\mathbb{A})$.

$$Proj(\mathbb{A}) = \left\{ t = (t_1, t_2, \dots, t_G) \in [0, 1]^G : \exists\, k \in \mathbb{A}, \forall\, g \quad t_g = \frac{k_g - N_g}{1 - N_g} \right\}$$

Le projeté de $\mathbb{A}$ étant maintenant défini, on peut définir une fonction F sur cet ensemble telle que $F(t) = H(k)$ où la fonction F prend en argument les taux de compression tandis que la fonction H quant à elle, prend en argument les allocations. Ainsi donc nous avons l'équivalence suivante :

$$\min_{k \in \mathbb{A}_K} H(k) \quad \Leftrightarrow \quad \min_{t \in Proj(\mathbb{A}_K)} F(t)$$

Remarques :

1. Par ce procédé, il est également possible de définir le projeté de $\mathbb{A}_K$ qui sera nécessairement inclus dans le projeté de $\mathbb{A}$.
2. Si les composantes des vecteurs de $\mathbb{A}$ sont tous majorés par des entiers naturels différents correspondant aux tailles des strates, l'intérêt de la projection réside dans le fait que toutes les composantes sont désormais comprises entre 0 et 1, ce qui facilite grandement la recherche de solution.

La Figure 2 montre l'impact du changement de variable sur les fonctions des erreurs de compression. Les erreurs de compression ne sont plus représentées en fonction du nombre de lignes alloué, mais en fonction du taux de compression auquel ce nombre alloué correspond.

3.2.2 Changement d'échelle

Afin de s'affranchir des ordres de grandeur des écarts de BEL par strate, on définit la fonction f_g comme suit :

$$f_g(t_g) = \frac{h_g(k_g)}{err_g}$$

Avec $err_g = h_g(1)$ égale à l'erreur de compression commise dans la strate g lorsque $k_g = 1$, c'est-à-dire l'erreur maximale. Ainsi notre fonction H peut être réécrite comme suit :

$$H(k) = \sum_{g=1}^{G} h_g(k_g) = F(t) = err_1 * f_1(t_1) + err_2 * f_2(t_2) + \cdots + err_G * f_G(t_G)$$

La Figure 3 montre l'impact du changement d'échelle sur les fonctions des erreurs de compression. Les erreurs de compression ne sont plus représentées sur l'axe des ordonnées en valeur monétaire, mais elles sont rapportées à l'erreur de compression correspondant à la compression maximale err_g pour une strate g.

Le programme d'optimisation, même après ces deux précédentes transformations reste toujours combinatoire. Cependant, dans le contexte de l'optimisation, une transformation du problème en un problème continu est possible par le biais d'une relaxation continue.

3.2.3 Relaxation continue

La relaxation continue est une méthode qui consiste à interpréter de façon continue un problème combinatoire ou discret. Elle permet de remplacer les contraintes discrètes par des contraintes continues, simplifiant ainsi la résolution numérique du problème. Cette méthode est utilisée afin d'obtenir des informations sur le problème discret initial et parfois même pour obtenir sa solution. Bien que les solutions obtenues ne soient pas nécessairement exactes dans le domaine discret, la relaxation continue tend à produire des solutions proches de l'optimum global.

Cette hypothèse fait passer le domaine de recherche de $Proj(\mathbb{A}_K)$, qui est un espace discret, au plus petit convexe contenant $Proj(\mathbb{A}_K)$ que nous notons $\overline{Proj(\mathbb{A}_K)}$, qui lui est un espace continu. Pour caractériser $\overline{Proj(\mathbb{A}_K)}$, on considère une allocation $k \in \mathbb{A}_K$

$$k_1 + k_2 + \cdots + k_G = K$$

$$N_1 + t_1(1 - N_1) + N_2 + t_2(1 - N_2) + \cdots + N_G + t_G(1 - N_G) = K$$

$$t_1(1 - N_1) + t_2(1 - N_2) + \cdots + t_G(1 - N_G) = K - N$$

D'où :

$$\overline{Proj(\mathbb{A}_K)} = \{t \in [0,\ 1]^G\ : (1 - N_1)t_1 + \cdots + (1 - N_G)t_G = K - N\}$$

Remarques :

1. Les fonctions f_g initialement discrètes sont étendues par leurs prolongements continus sur $\overline{Proj(\mathbb{A}_K)}$ mais garderont la même notation.

2. Dans toute la suite, la notation f_g sera utilisée pour désigner uniquement ces prolongements continus.
3. Il existe plusieurs manières de prolonger les fonctions f_g, mais notre choix se porte sur le prolongement par interpolation linéaire pour des raisons de parcimonie.

La Figure 4 montre l'impact de la relaxation continue sur les fonctions des erreurs de compression. Les taux de compression ne sont plus considérés dans un espace discret, mais sont pris dans l'ensemble continue $[0,1]$ tout entier, et les paires de points adjacents sont interpolées linéairement. La technique d'approximation par interpolation linéaire sera discutée plus en détail plus bas dans cet article.

Les trois transformations, à savoir : le changement de variables, le changement d'échelle et la relaxation continue nous permettent de passer du programme initial (*)

$$\min_{k\in\mathbb{A}} H(k) = \sum h_g(k_g) \quad s.c. \quad k_1 + k_2 + \cdots + k_G = K$$

Au programme (**)

$$\min_{t\in[0,\,1]^G} F(t) = \sum err_g * f_g(t_g) \quad s.c. \quad (1-N_1)t_1 + (1-N_2)t_2 + \cdots + (1-N_G)t_G = K - N$$

Le passage du problème discret au problème continu introduit un premier biais dû au fait qu'il nous est impossible de connaître les véritables formes continues des fonctions discrètes que nous avons prolongées. Ce biais est donc un biais d'estimation et existera quel que soit le prolongement que nous déciderons de réaliser. Par exemple si nous décidions de prolonger les fonctions de façon quadratique au lieu d'une interpolation linéaire, ce biais prendrait une autre forme plus au moins élevée ou faible mais pas nulle. C'est donc une fonction du mode d'interpolation choisi. Toutefois, puisque nous avons pris le parti, dans cet article, d'interpoler toutes les fonctions linéairement, ce biais peut être considéré ici comme un invariant. Notons-le μ. Ce biais est impossible à évaluer et constitue une limite structurelle de notre approche. L'annuler reviendrait à résoudre le problème dans sa forme discrète.

À ce stade, il est théoriquement envisageable de tenter de résoudre directement le programme (**). Cependant, la fonction objectif de ce programme est une somme de fonctions continues mais non différentiables, car issues d'une interpolation linéaire, ce qui la rend également continue mais non différentiable.

3.3 L'allocation optimale comme solution d'un programme continu et différentiable

Les fonctions objectifs non différentiables posent un défi majeur dans les programmes d'optimisation car elles ne permettent pas l'utilisation des méthodes traditionnelles de calcul du gradient. L'absence de différentiabilité signifie que la fonction ne présente pas de pente ou de direction unique en chaque point, rendant impossible l'application des algorithmes classiques de descente de gradient. Cela complique la recherche des points optimaux où la fonction atteint ses valeurs minimales ou maximales. En outre, l'absence de dérivées conduit souvent à des solutions non uniques ou à une convergence plus lente vers l'optimum, ce qui rend l'optimisation plus complexe et demande une exploration plus approfondie de l'espace des solutions.

Pour surmonter ce problème de non différentiabilité de la fonction objectif, nous proposons d'utiliser des approximations par les polynômes de Bernstein.

3.3.1 Approximation par les polynômes de Bernstein

Les polynômes de Bernstein [3] sont des outils mathématiques polyvalents qui permettent de représenter des fonctions sous forme d'une combinaison linéaire de fonctions de base appelées monômes de Bernstein. Chaque monôme de Bernstein est associé à un degré et est construit à partir de combinaisons de termes binomiaux. Les polynômes de Bernstein sont définis sur un intervalle $[0, 1]$ et sont utilisés principalement en analyse numérique et en informatique graphique pour approcher des courbes et des surfaces. Ils ont des propriétés utiles, notamment la positivité et la convexité, qui les rendent adaptés à diverses applications, telles que l'interpolation et l'approximation de fonctions.

Les polynômes de Bernstein peuvent être utilisés pour représenter les fonctions f_g de manière efficace tout en maintenant des propriétés de continuité et de différentiabilité. En effet, en approximant une fonction non différentiable par une combinaison de polynômes de Bernstein, on peut obtenir une fonction globalement différentiable qui conserve les caractéristiques essentielles de la fonction d'origine. Cette approximation permet ensuite l'application de méthodes d'optimisation classiques basées sur le calcul de gradients et de matrices Hessiennes, facilitant ainsi la recherche des optima.

La fonction f pour une strate g quelconque est approximée par la fonction polynômiale suivante :

$$B_n(f)(x) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) * C_n^k \, x^k (1-x)^{n-k} \qquad pour\ tout \quad x \in [0, 1]$$

Le programme final (***) à résoudre est le suivant :

$$\min_{t \in [0,\,1]^G} F_{bern}(t) = \sum_g err_g * B_n(f_g)(t_g) \quad s.c. \quad (1 - N_1)t_1 + \cdots + (1 - N_G)t_G = K - N$$

Définition 3.2 – On appellera allocation optimale réalisable, l'allocation solution du programme d'optimisation (***).

L'allocation optimale réalisable est la meilleure allocation qu'il est possible d'obtenir avec notre méthode. Elle peut être différente de l'allocation optimale présentée en définition 3.1. Cette différence vient de la relaxation continue et de l'approximation par les polynômes de Bernstein.

L'approximation par les polynômes de Bernstein introduit un deuxième biais dans les estimations dû au fait qu'il existe une différence de nature entre les fonctions sous-jacentes et les polynômes issus de l'approximation. Les premières sont non dérivables tandis que les seconds le sont. Ce biais est une fonction décroissante du paramètre n dans les polynômes. Ce paramètre correspond au degré du polynôme. Ce biais sera noté $bern\_approx\_error(n)$. Augmenter n permet d'obtenir une approximation plus précise de la fonction sous-jacente et donc une baisse de $bern\_approx\_error(n)$. En d'autres termes, plus n est grand, plus les polynômes de Bernstein peuvent représenter des fonctions complexes de manière précise. Cela peut être particulièrement utile lorsque l'on travaille avec des fonctions qui ont des variations rapides ou des caractéristiques complexes, car un n plus élevé permet d'obtenir une approximation plus fidèle de ces variations. Cependant, cela nécessite également plus de calculs, ce qui peut être un compromis à considérer en fonction des besoins spécifiques.

L'erreur d'approximation $bern\_approx\_error(n)$ peut être évaluée de la façon suivante :

$$bern\_approx\_error(n) = \sum_g err_g * \int_0^1 \left| f_g(x) - B_n(f_g)(x) \right| dx$$

Il peut convenir dans certains cas de normaliser cette expression.

3.3.2 Incomplétude des bases d'information et interpolation linéaire

En pratique, nous ne disposons jamais des erreurs de compression associées à chaque valeur de nombre de lignes alloué pour la strate. Certaines strates dépassent même les $1\,000$ lignes. Il devient donc inenvisageable de calculer les erreurs de compression pour les valeurs de 1 à $1\,000$ et ce pour toutes les strates du portefeuille, pour ensuite choisir la meilleure combinaison satisfaisant la contrainte d'admissibilité. Les fonctions présentées ci-dessus nous resteront donc toujours inconnues. Cette contrainte nous pousse à les approximer.

Définition 3.3 – On appelle une information sur une fonction f la connaissance d'un couple $(x, f(x))$. Par construction, la base d'information connue pour toute fonction f parmi les fonctions f_g définies plus haut est constituée des deux couples suivants :

$$(0,\ f(0)=0),\ \ (1,\ f(1)=1) \qquad (2)$$

Cependant, si f doit être correctement approximée, il est indispensable de disposer d'un minimum d'informations supplémentaires à son sujet. Disposer d'une information supplémentaire sur f consiste à choisir un taux de compression, par exemple $0{,}5$ et à calculer $f(0{,}5)$. On dit qu'on a placé la valeur intermédiaire $0{,}5$ sur le graphe de f. La qualité de l'approximation de f dépend du niveau d'information dont on dispose, c'est-à-dire du nombre de valeurs intermédiaires que l'on décide de placer sur le graphe. Plus l'on dispose d'informations, meilleure sera l'approximation.

L'élargissement de la base d'information a toutefois un coût computationnel non négligeable, car chaque nouvelle acquisition d'information requiert une nouvelle itération de l'algorithme de compression.

Remarque : il est nécessaire de faire tourner l'algorithme de compression au moins une fois pour chaque strate g avec $k_g = 1$, afin de calculer les erreurs de compression maximales err_g.

On suppose la base d'information suivante disponible pour la fonction f à la suite de plusieurs itérations de l'algorithme de compression :

$$(0,0), \big(x_1, f(x_1)\big), \big(x_2, f(x_2)\big), \dots, (x_p, f(x_p)), (1,1) \qquad (3)$$

Notre approche va consister à relier les points de la façon la plus intuitive possible. Pour cela, un choix naturel est l'interpolation linéaire. L'interpolation linéaire est une technique d'approximation qui consiste à relier chaque paire de points consécutifs par des segments de droite. Cela revient à supposer que la fonction f est linéaire entre chaque paire de points adjacents. Ainsi, pour tout x compris entre deux points donnés, l'interpolation linéaire prédit la valeur de $f(x)$ en utilisant une droite reliant les valeurs de f aux extrémités de l'intervalle.

La fonction f^{lin} désigne l'interpolation linéaire de f réalisée à partir des points disponibles dans la base d'information :

$$f^{lin}(x) = \sum_{k=0}^{p} f(x_k) + \ (f(x_{k+1}) - f(x_k)) * \frac{x - x_k}{x_{k+1} - x_k}\ \mathbb{1}_{[x_k,\, x_{k+1}[}(x)$$

Dans ce cas, le programme final (****) à résoudre est le suivant :

$$\min_{t \in [0,\,1]^G} F_{bern}(t) = \sum_g err_g * B_n\big(f_g^{lin}\big)(t_g) \quad s.c. \quad (1-N_1)t_1 + \cdots + (1-N_G)t_G = K - N$$

Le programme (****) est celui auquel nous aurons affaire dans la quasi-totalité des cas que nous rencontrerons dans la pratique. C'est lui qui nous fournira l'allocation qui sera effectivement utilisée pour réaliser la compression finale et produire les chiffres. L'allocation solution de ce programme sera nommée allocation optimale réalisée.

Définition 3.4 – On appellera allocation optimale réalisée, l'allocation solution du programme d'optimisation (****).

La base d'information d'une fonction f présentée en (3) peut être partielle ou complète. Dans le cas où elle est complète, il ne devient plus utile d'interpoler les fonctions f puisque les fonctions f^{lin} se confondraient avec celles-ci. On approxime directement les fonctions par les polynômes de Bernstein et l'on résout le programme (***). Dans le cas où elle est partielle, l'approximation par interpolation linéaire introduit un biais supplémentaire dû au fait que f^{lin} ne sera pas une représentation parfaite de f. Ce biais est donc un biais d'interpolation qui peut être évalué et que nous pouvons réduire en augmentant le nombre de valeurs intermédiaires choisi. C'est donc une fonction décroissante du nombre de valeurs intermédiaires.

Nous allons le définir comme suit :

$$lin\_interp\_error(p) = \sum_{g} err_g * \epsilon_g(p) \quad \text{avec } p \in \{0, 1, 2, \dots, Nmax - 2\}$$

Et :

$$\epsilon_g(p) = \int_0^1 \left| f_g(x) - f_g^{lin}(x) \right| dx$$

Où :

- p désigne le nombre de valeurs intermédiaires choisi
- $Nmax$: le nombre de lignes de la strate qui en compte le plus, $Nmax = max(N_1, N_2, \dots, N_G)$
- $\epsilon_g(p)$ est la valeur de ce biais au sein de la strate g tel que $\epsilon_g(p) = 0 \quad \forall\, p \geq N_g - 2$

Enfin, il est important de remarquer que le cas $p = 0$ correspond à la situation où toutes les bases d'information correspondent à celle présentée en (2), $p = Nmax - 2$ correspond au cas où toutes les bases d'information sont complètes et donc $lin\_interp\_error(Nmax - 2) = 0$.

3.3.3 Hyperparamètres et critères de convergence

Pour un nombre de valeurs intermédiaires p et pour un degré des polynômes de Bernstein n, le biais global de notre méthode lors de la résolution du programme (****) peut être exprimé comme suit :

$$overall\_error(p, n) = \mu + lin\_interp\_error(p) + bern\_approx\_error(n)$$

Ces deux paramètres sont les hyperparamètres de notre méthode, étant donné qu'ils sont définis préalablement avant l'application de la méthodologie. Leur importance réside dans leur capacité à réguler la complexité de notre approche, ce qui influence directement son efficacité et ses performances. Une borne inférieure du biais peut être définie comme suit :

$$overall\_error(p, n) \geq \mu + bern\_approx\_error(n)$$

Cette inégalité indique que quelle que soit la valeur de n choisie, le biais introduit lors de la résolution du programme (****) est au moins aussi élevé que celui du programme (***), et que ces deux programmes tendent à se rapprocher à mesure que le nombre de valeurs intermédiaires augmente.

En d'autres termes, l'allocation optimale réalisée est toujours moins efficace que l'allocation optimale réalisable, mais elle converge vers cette dernière avec l'augmentation du nombre de valeurs intermédiaires. Notamment, l'allocation optimale réalisable est réalisée lorsque $p = Nmax - 2$.

Nous pouvons désigner cette borne inférieure comme suit :

$$borne\_inf(n) = \mu + bern\_approx\_error(n)$$

De plus, il convient de noter que en raison de la différence de nature évoquée plus haut entre les fonctions sous-jacentes et les polynômes de Bernstein, $bern\_approx\_error(n)$ ne peut jamais être réellement nul. Cependant, pour tout réel positif non nul α, aussi petit soit-il, il est possible de trouver un rang n_α à partir duquel $bern\_approx\_error(n)$ sera plus petit que α. On dit que $bern\_approx\_error(n)$ admet une limite en 0 lorsque n tend vers l'infini. Ainsi, augmenter n permet de repousser la borne inférieure de notre biais global, ce qui implique un rapprochement entre l'allocation optimale réalisable et l'allocation optimale. Cette convergence est mise en évidence par la limite suivante :

$$\lim_{n\to\infty} borne\_inf(n) = \mu + \lim_{n\to\infty} bern\_approx\_error(n) = \mu$$

Cette limite montre aussi que malgré tous les efforts entrepris pour repousser cette borne inférieure, nous serons confrontés à une limite structurelle : la présence inévitable d'un biais d'estimation résultant de la relaxation continue. Cette limite est infranchissable ; pour l'annuler, il est nécessaire de résoudre le programme discret (*). Par conséquent, même si l'allocation optimale réalisable tend à s'approcher de l'allocation optimale, nous ne sommes pas en mesure de savoir si celle-ci atteint l'allocation optimale, c'est à dire la plus efficace, ni à quelle distance de l'allocation optimale elle se situe.

3.4 Recommandations sur la résolution des programmes d'optimisation

3.4.1 Gradient et Matrice Hessienne

Les problèmes d'optimisation présentés dans cet article ont été résolus en utilisant le package ROI [4]. Le package ROI (R Optimization Infrastructure) est une bibliothèque R puissante et flexible conçue pour la résolution de problèmes d'optimisation. Il a été développé en 2020 par *Hornik K et al.* [5] et il offre une interface unifiée pour plusieurs solveurs d'optimisation, permettant aux utilisateurs de spécifier et de résoudre divers types de problèmes d'optimisation, y compris linéaires, quadratiques, non linéaires, convexes et non convexes.

Une caractéristique remarquable de ROI est sa capacité à résoudre des problèmes d'optimisation sans nécessiter que lui soit fournis explicitement le gradient ni la matrice Hessienne de la fonction objectif. Pour ce faire, ROI utilise des techniques d'optimisation basées sur des heuristiques pour trouver des solutions. Cependant, bien que l'on soit tenté d'utiliser cette solution, il est crucial d'insister sur la nécessité de fournir, dans certains cas, à ROI à la fois le gradient et la matrice Hessienne de la fonction objectif lors de la résolution du programme (****). Pour illustrer cette nécessité, nous avons mené une étude portant sur 884 strates. Cette étude a consisté à configurer le programme d'optimisation selon trois approches distinctes. Dans la première configuration, seule la fonction objectif était fournie. Pour la seconde configuration, outre la fonction objectif, le gradient était également inclus. Enfin, pour la troisième configuration, la fonction objectif, le gradient et la matrice Hessienne étaient tous trois fournis à l'algorithme. Le tableau 1 synthétise les temps de calcul approximatifs requis par l'algorithme pour atteindre la convergence.

Table 1. Temps de calcul requis selon la configuration

Fonction objectif	Gradient	Hessienne	Temps
Oui	Non	Non	$\geq 3h$
Oui	Oui	Non	$\approx 1h$
Oui	Oui	Oui	$\approx 20\ mins$

3.4.2 Initialisation de l'algorithme

La plupart des algorithmes d'optimisation nécessitent qu'on leur fournisse un point de départ pour fonctionner : c'est l'étape d'initialisation. Dans notre cas, cette initialisation peut se faire avec toute allocation admissible. Parmi les différentes techniques d'initialisation, la plus simple et la plus repandue est l'initialisation aléatoire. Cependant, l'initialisation aléatoire présente plusieurs inconvénients. Tout d'abord, elle peut produire des résultats non reproductibles, ensuite elle ne tient pas compte de l'importance relative des strates et elle peut conduire à des situations de déséquilibre où certaines strates sont en situation de sous-allocation, tandis que d'autres sont en situation de sur-allocation sans raison valable, ce qui peut entraîner des temps de convergence plus longs ou même la non-convergence de l'algorithme. Enfin, il est plus intéressant d'utiliser comme point de départ l'allocation traditionnellement utilisée, afin de mettre en évidence la valeur ajoutée de notre méthode. Pour toutes ces raisons, nous allons préférer à l'initialisation aléatoire, une initialisation à partir d'allocations issues de l'approche proportionnelle, car elles sont toujours admissibles et à la fois reproductibles, faciles à calculer, reliées à des quantités actuarielles et se rapprochent intuitivement de la solution optimale.

La mise en œuvre de la démarche précédente sur un cas pratique est présentée dans une seconde partie, qui fera l'objet d'un prochain article à paraître dans Variances.

Références

[1] Ronan Le Gleut. *Stratification et calcul d'allocations dans les enquêtes auprès des entreprises.* Département des méthodes statistiques (INSEE), Version n°1. diffusée le 11 septembre 2017. url:https://www.insee.fr/fr/statistiques/fichier/2838097/3-stratification-et-calcul-d-allocations-dans-les-enquetes-entreprises.pdf

[2] Laurent Guillopé. *Optimisation sous contrainte*. Laboratoire de mathématiques Jean Leray Département de mathématiques, UFR Sciences et techniques (Université de Nantes), Version du 2 mars 2020. url: https://www.math.sciences.univ-nantes.fr/~guillope/l3-osc/.

[3] Øljen - Les maths en finesse. *Polynômes de Bernstein (Les origines)*. YouTube. 2020. url: https://youtu.be/_Z07z13Tol4?feature=shared.

[4] Kurt Hornik, David Meyer, Florian Schwendinger, and Stefan Theussl. ROI: *R Optimization Infrastructure*. R package version 1.0-1. 2023. url: https://CRAN.R-project.org/package=ROI.

[5] Stefan Theußl, Florian Schwendinger, and Kurt Hornik. "ROI: An Extensible R Optimization Infrastructure". In: Journal of Statistical Software 94.15 (2020), pp. 1–64. https://www.jstatsoft.org/article/view/v094i15