**DOCUMENTATION PASS-ALM**

**BRIQUE CALCULATEUR**

**DOCUMENTATION DU CODE DU CALCULATEUR PASS-ALM POUR LES ECOULEMENTS DES PRODUITS ECHEANCES et CAP-FLOOR**

**Sommaire**

[1. Objet 2](#_Toc149902256)

[2. Synthèse des modèles généraux 2](#_Toc149902257)

[2.1 Modèle général d’amortissement du capital 2](#_Toc149902258)

[2.2 Modèle général des scénarios de RA et RN 3](#_Toc149902259)

[3. Procédure générale pour le traitement des contrats échéancés 4](#_Toc149902260)

[4. Implémentation numérique 5](#_Toc149902261)

[4.1 Nomenclature des produits simulés 5](#_Toc149902262)

[4.2 Gestion des données avec le dossier data\_manager 6](#_Toc149902263)

[4.3 Gestion des paramètres de projection données avec le dossier projection 12](#_Toc149902264)

[4.4 Calcul des taux RA/RN avec les classes du dossier « rarn » 18](#_Toc149902265)

[4.5 Calcul de l’amortissement avec les classes du dossier « amortization » 19](#_Toc149902266)

[4.6 Calcul des indicateurs avec le dossier : indicators 34](#_Toc149902267)

[4.7 Génération des résultats avec le dossier : reports 36](#_Toc149902268)

# Objet

Ce fichier vise à documenter les différentes fonctions implémentées dans PASS-ALM pour décrire le processus d’écoulements des produits échéancés en tenant compte des remboursements anticipés et de renégociations (RA/RN). Chaque classe est décrite de manière fonctionnelle avec une description des méthodes principales.

# Synthèse des modèles généraux

Cette partie synthétise les modèles utilisés pour les écoulements des produits échéancés. Pour une meilleure compréhension de la démarche, la lecture de l’ensemble des documents « DOCU\_MODELE\_PASS\_ALM\_v.4.0\_MODELE\_RARN\_STOCK.docx » et « DOCU\_MODELE\_PASS\_ALM\_v.5.1\_PRODUIT\_ECHANCES.docx » est recommandée.

## 2.1 Modèle général d’amortissement du capital

La formule générale appliquée dans PASS-ALM pour l’amortissement du capital d’un contrat immobilier à taux fixe est la suivante :

,

Cette formule est adaptée en fonction du profil du contrat : profil et fréquence d’amortissement, capitalisation des intérêts sur la première période, taux d’intérêt variable selon le palier etc.

La formule ainsi écrite permet une implémentation dans python maximisant l’usage des librairies *numpy* et *numexpr* et en conséquence, elle permet d’accélérer les temps d’exécution.

## 2.2 Modèle général des scénarios de RA et RN

**a. Calcul des taux de RA et de RN**

**Il existe deux types de modèles RA : les taux RA financiers et les taux RA statiques. Dans le cas des crédits immobiliers, les taux RA financiers s’appliquent.**

**i) Taux RA/RN financiers dans le cas des crédits immobiliers**

Le modèle implémenté est similaire à RCO et est tel que défini par l‘équipe Modèles :

+ 0,5%)

*Les paramètres**, , , , et dépendent de du ratio entre la* ***DRAC*** *(durée restante à courir résiduelle) et la durée restante initiale.* ***M(t)*** *est la maturité résiduelle du contrat à la période t.*

**ii) Taux RA statiques pour les autres produits**

Dans ce cas de figure, les taux de RA sont constants et n’ont pas à être évalués de manière dynamique, ils sont directement donnés par type de contrat, établissement et/ou par marché. Il faut cependant les mensualiser.

**b. Calcul du capital après effet RA**

Le calcul du capital après effet RA est conforme à RCO et s’écrit :

**c. Calcul du capital des contrats renégociés**

**c.1**

**c.2**

**c.3**

**c.4**

La formule en **c.3** est issue d’un développement limité à l’ordre de 2 dont les hypothèses et la preuve sont données dans le document. Elle permet d’accélérer le temps de traitement en gardant une précision suffisante.

Par ailleurs, PASS-ALM applique de manière séquentielle l’amortissement du capital et les effets de RA/RN, contrairement à RCO qui les applique de manière simultanée. Il s’agit d’une approximation dont les effets sont négligeables et qui concerne seulement les contrats à amortissement non mensuel.

# 3. Démarche générale du traitement des contrats échéancés

1. **Mise en forme des données et initialisation des paramètres de projection**
2. **Génération des amortissements du capital**
3. **Obtention des taux de RA, RN**
4. **Mise à jour du capital en fonction de l’effet RA/RN**
5. **Calcul du capital moyen de la jambe statique**
6. **Calcul de la MNI de la jambe statique**
7. **Calcul du capital et de la MNI de la jambe renégociée**
8. **Calcul des indicateurs LEF** (Gap de liquidité fin de mois), **LEM** (Gap de liquidité moyen mensuel), **TEF** (Gap de taux fin de mois), **TEM** (Gap de taux moyen mensuel), la **MNI** (Marge Nette d’Intérêt aux taux client), la **MNI FTP** (Marge Nette d’Intérêt au TCI), la **MNI GAP de TAUX** (Marge Nette d’Intérêt aux taux client calculé sur le TEM), la **MNI FTP GAP de TAUX** (Marge Nette d’Intérêt au TCI calculé sur le TEM).

# 4. Implémentation numérique

Le code ici va se dissocier en plusieurs en classes, ces classes sont réparties dans plusieurs dossiers en fonction de leur utilité. Voici les différentes catégories :

* **Data\_manager** : contient les différents fichiers en lien avec le chargement des paramètres utilisateurs, du modèle ainsi que pour le formatage et le filtrage des données.
* **Projection**: contient les différents fichiers pour générer les bons paramètres afin de réaliser le calcul des amortissements. En particulier, ces classes traitent les paramètres de calendrier, de palier ainsi que les taux.
* **Amortization**: contient les différents fichiers pour réaliser le calcul d’amortissement. Cela revient à calculer l’amortissement par défaut, par palier et des contrats douteux.
* **Rarn**: contient le fichier pour appliquer les taux ra/rn en fonction du produit.
* **Indicators** : contient les fichiers pour calculer des indicateurs comme la MNI ou le LEM.
* **Reports**: contient les fichiers pour sauvegarder les résultats dans un fichier excel.
* **Utils** : contient exclusivement des fonctions qui s’appliqueront aux différents fichiers (par exemple pour effectuer des transformations afin d’écrire le problème sous forme matricielle).

## 4.1 Nomenclature des produits simulés

Avant d’expliciter comment le code s’articule, il est nécessaire de lister les différents produits à simuler ainsi que la nomenclature utilisée. Pour commencer, chaque produit commence par la lettre « a » ou « p » pour désigner si c’est un actif ou un passif (hormis le cas particulier des cap&floor). À la suite de cette lettre, le nom du produit est donné. Voici la liste complète:

* **"a-creq"** : crédit équipement à taux fixe
* **"a-creqtv"** : crédit équipement à taux variable
* **"a-crctz"** : crédit à taux zéro
* **"a-crctf"** : crédit à la consommation à taux fixe
* **"a-crctv"** : crédit à la consommation à taux variable
* **"a-crif"** : crédit immobilier à taux fixe
* **"a-criv"** : crédit immobilier à taux variable
* **"a-intbq-tf"** : actif des crédits interbancaires à taux fixe
* **"a-intbq-tv"**: actif des crédits interbancaires à taux variable
* **"p-intbq-tf"** : passif des crédits interbancaires à taux fixe
* **"p-intbq-tv"** : passif des crédits interbancaires à taux variable
* **"p-cat-tf"** : passif des comptes à terme à taux fixe
* **"p-cat-tv"**: passif des crédits interbancaires à taux variable
* **"cap\_floor"**: cap&floor
* **"a-autres-tf"** : actif des autres à taux fixe
* **"a-autres-tv"**: actif des autres à taux variable
* **"p-autres-tf"** : passif des autres à taux fixe
* **"p-autres-tv"**: passif des autres à taux variables

Le code va donc permettre la simulation de l’écoulement de chacun de ces produits, nous allons donc maintenant voir de manière détaillée les différentes classes utilisées.

## 4.2 Gestion des données avec le dossier data\_manager

Ce dossier contient les classes suivantes :

* Class\_data\_manager
* Class\_data\_formater
* Class\_data\_quality\_manager

Il contient également les dossiers suivants :

* data\_model\_manager
* data\_products\_manager

Nous reviendrons sur les classes à l’intérieur de ces dossiers par la suite.

### 4.2.1 Classe : class\_data\_manager

Cette classe a pour objectif de charger les fichiers des contrats à écouler (i.e. les fichiers LDP, PAL) et contient les paramètres en lien avec ces différents fichiers. Plus précisément, elle a pour attributs les colonnes de ces différents fichiers et leur attribuent une catégorie.

Par exemple, les colonnes en lien avec les paramètres nécessaires à la projection sont stockées dans la variable « *proj\_vars*». Cette variable va alors contenir les colonnes comme le nom du contrat, la valeur de son échéance, la fréquence d’amortissement, etc.

**Instanciation :**

* **def** \_\_init\_\_(self, name\_product=**"a-crctz"**): prend en argument le nom du produit à écouler.

**Fonctions clés :**

* **def** generate\_general\_params(self) : va récupérer les différents paramètres à partir des colonnes des fichiers LDP et PAL. Ces paramètres vont être scindés en différentes catégories selon leur utilité et être stocké dans un attribut. Par exemple : les variables des fichiers LDP sont stockées dans : self.ldp\_vars, les variables des fichiers PAL dans : self.pal\_vars, les variables nécessaires à la projection dans self.proj\_vars.
* **def** load\_usr\_params(self, sources\_params, dar): charge les fichiers LDP, PAL, CashFlow et les stock dans les attributs respectifs : self.input\_rarn\_ldp, self.input\_rarn\_pal, self.input\_rarn\_cf
* **def** set\_reports\_options(self, gen\_agreg\_report, gen\_detailed\_report, gen\_model\_report): défini le type de résultats que la simulation va générer (des résultats agrégés par établissement ou détaillés pour chaque contrat par exemple).

### 4.2.2 Classe : class\_ data\_formater

Cette classe a pour objectif de trier et de formater les différents contrats conformément à RCO. Cette classe va par exemple analyser les dates des contrats pour supprimer les contrats finalisés. Un autre exemple consiste à formater la fréquence d’amortissement en transformant des chaînes de caractères en nombre i.e. : « ANNUEL » => 12.

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_data): prend en argument la classe data\_manager.

**Fonctions clés :**

* **def** parse\_date\_vars(self, data\_ldp\_hab) : transforme les dates en nombre de mois 15/12/2024 => 2024\*12 + 12 = 24300, avec gestion d’exception. Il faut par exemple s’assurer que la date de maturité ne soit pas inférieure à la date du premier amortissement. Cette fonction permet également de calculer les variables first\_amor\_date (date du premier amortissement), value\_date (date à laquelle le contrat commence), matur\_date (date de fin de contrat) et trade\_date (date à laquelle on signe le contrat).
* **def** generate\_freq\_amor(self, data, col\_freq\_amor, col\_freq\_int): Fonction pour décrire à quelle fréquence le crédit est remboursé (annuel, mensuel, trimestriel, semestriel). Elle formate également les données de la colonne « col\_freq\_amor » (qui est appelée pour transformées les mots « monthly, quaterly, etc.. » en français i.e. "TRIMESTRIEL", "SEMESTRIEL". Également, une nouvelle colonne est créée pour transformer l’information « MENSUEL » en un nombre i.e. 1.
* **def** get\_supsension\_or\_interets\_capitalization\_var(self, data, col\_freq\_int, capi\_rate\_col=[], capi\_freq\_col=[], type\_data=**"ldp"**): fonction qui crée une variable pour savoir s’il faut capitaliser les intérêts ou non dans un premier temps (i.e. le temps d’un palier). Cela permet de savoir si l’amortissement est suspendu et dans le cas où l’amortissement est suspendu, faut-il faut capitaliser les intérêts ou non.

### 4.2.3 Classe : class\_data\_quality\_manager

Cette classe a pour objectif de mettre à jour la date de départ du contrat (forward ou non) pour les contrats ayant un déblocage avec un outstanding. Elle met aussi à jour les dates de maturité. Ces fonctions servent à traiter les problèmes de déblocage.

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_debloc\_params): prend en argument une instance de la classe data\_deblocage\_model\_params qui sera détaillée par la suite)

**Fonctions clés :**

* **def** update\_mat\_and\_val\_date\_for\_already\_realeased\_cap(self, data\_hab) et **def** update\_mat\_and\_val\_date\_for\_deblocage(self, data) :

mise à jour des dates de maturité et de valorisation pour les déblocages. Elles servent notamment à traiter les problèmes de déblocage.

### 4.2.4 Dossier : data\_model\_manager

#### 4.2.4.1 Classe : classe\_deblocage\_model\_params

Récupère les matrices de déblocage pour le produit considéré et par type de contrat à partir du fichier modèle. Également, récupère la matrice de déblocage associée à une règle de déblocage.

**Instanciation :**

* **def** \_\_init\_\_(self, model\_wb, cls\_data): nécessite le fichier modèle pour pouvoir récupérer les différentes informations sur les matrices de déblocage ainsi que la classe data\_manager.

**Fonctions clés :**

* **def** get\_deblocage\_mtx\_per\_prod(self, name\_prod): récupère les matrices de déblocage pour un produit considéré et pour les types de contrat associés à cet établissement.
* **def** get\_deblocage\_mtx(self, data\_num\_rule) : récupère la matrice de déblocage selon une règle de déblocage.

#### 4.2.4.2 Classe : classe\_douteux\_model\_params

Cette classe initialise les paramètres des contrats douteux. Elle définit diverses constantes, telles que "NR\_EVOl\_DOUTEUX" et "NC\_MOD\_CTRT", et une taille pour les données douteuses (size\_ecoul\_douteux).

Elle effectue également des opérations spécifiques en fonction du produit (name\_product) associé aux données de classe. En fonction du produit, la classe charge des profils de contrats douteux depuis le classeur de modèle (model\_wb), filtre et stocke ces profils en fonction de certaines conditions, telles que le type de contrat ("NC\_MOD\_CTRT").

**Instanciation :**

* **def** \_\_init\_\_(self, model\_wb, cls\_data): nécessite le fichier modèle pour pouvoir récupérer les différentes informations sur les contrats douteux ainsi que la classe data\_manager.

**Fonctions clés :**

* **def** load\_douteux\_model\_params(self) : charge les paramètres i.e. les contrats douteux à partir du produit et des contrats considérés.

#### 4.2.4.3 Classe : classe\_proj\_horizon\_params

Cette classe initialise les données de paramètres liés à la projection d'horizons, en prenant en compte des paramètres tels que le modèle, la durée de projection (horizon), et les données de classe (cls\_data). Elle définit également des paramètres maximums pour diverses valeurs, comme le nombre maximum de périodes de projection, la durée maximale de projection écoulée, etc. La classe ajuste la durée de projection si elle dépasse les valeurs maximales définies.

**Instanciation :**

* **def** \_\_init\_\_(self, model\_wb, horizon, cls\_data) : a besoin de l’horizon pour calculer différents attributs comme le nombre de projection au total.

**Fonctions clés :**

* **def** load\_horizon\_params(self): charge les paramètres de durée de projection (comme le nombre de mois de projection).

#### 4.2.4.4 Classe : classe\_ra\_rn\_model\_params

Récupère les paramètres ra/rn pour le produit considéré. Il est important de souligner qu’il y a le cas particulier où des paramètres RA/RN dynamiques s’appliquent là où dans le cas des autres produits, il n’y a que des taux RA constants.

**Instanciation :**

* **def** \_\_init\_\_(self, model\_wb, cls\_data, nb\_mois\_proj) : nécessite le fichier modèle pour pouvoir récupérer les différentes informations sur les taux, la classe data\_manager et le nombre de mois de projection.

**Fonctions clés :**

* **def** load\_rarn\_model\_params(self): récupère les taux RA constants en fonction des différents produits et des différents contrats.
* **def** load\_crif\_ra\_rn\_model(self): récupère les paramètres permettant de calculer les taux RA et RN de manière dynamique ainsi que les taux RA constants pour les crédits immobiliers à taux fixe.

### 4.2.5 Dossier : data\_products\_manager

#### 4.2.5.1 Classe : classe\_data\_cash\_flow

Cette classe a pour objectif de récupérer les données contenues dans le fichier cash\_flow. Cette classe va également mettre à jour le nominal et l’outstanding ainsi que supprimer les contrats du fichier cash\_flow sans cash\_flow.

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_hz\_params, data\_cf\_cpfl): s’instancie à partir de la classe horizon\_model\_params et des données du fichier cash\_flow

**Fonctions clés :**

* def read\_cash\_flow\_file(self): lit le fichier de flux de trésorerie et effectue diverses opérations pour le formater. Les données sont nettoyées, converties en valeurs numériques, et organisées en fonction de la date de fin du mois. Si le produit n'est pas "p-pel" ou "pn-ech", les données de flux de trésorerie sont également filtrées et préparées pour le calcul. Si le produit est "p-pel" ou "pn-ech", des listes de données vides sont créées pour ces produits.
* **def** update\_nominal\_and\_outstanding(self, data\_hab) : met à jour le nominal et l’oustanding. Pour cela, cette méthode calcule le nouvel « oustanding » en additionnant des flux au mois M1 pour les contrats en cours, tandis que le « nominal » est mis à jour pour les contrats sans règle de déblocage

#### 4.2.5.2 Classe : classe\_data\_palier\_manager

La classe Data\_Palier\_Manager gère le formatage des données relatives aux contrats du fichier PAL. Cette classe va permettre de construire un DataFrame contenant l’ensemble des informations contenues dans le fichier PAL

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_format, cls\_data, data\_palier): s’instancie à partir des classes data\_formater et data\_manager ainsi du dataFrame obtenu en ouvrant le fichier PAL.\_

**Fonctions clés :**

* **def** format\_pal\_file (self) : effectue plusieurs opérations de nettoyage et de formatage sur les données du palier. Ces opérations incluent la gestion des valeurs manquantes, le renommage de certaines colonnes, le traitement des dates, la conversion des données numériques, la génération de certaines colonnes, et la création d'une nouvelle structure de données : un dataFrame (self.mapping\_paliers) contenant l’ensemble de ces informations post-traitement.

#### 4.2.5.3 Classe : classe\_data\_ldp\_manager

La classe Data\_LDP\_Manager est responsable de la gestion des données liées au fichier LDP et de leur formatage pour le traitement ultérieur.

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_format, cls\_data): s’instancie à partir des classes data\_formater et data\_manager.

**Fonctions clés :**

* **def** set\_data\_ldp(self) : formate les données LDP en effectuant plusieurs opérations de nettoyage, de filtrage et de reformatage. Elle gère également les contrats LDP en fonction du produit. En effet, si le produit est un passif ou un actif, les opérations à réaliser seront différentes.
* **def** get\_ldp\_data\_in\_chunk (self) : obtient les données LDP en chunks, c'est-à-dire en morceaux de données plus petits (de taille 10000), ce qui est utile pour le traitement de gros volumes de données.

## 4.3 Gestion des paramètres de projection données avec le dossier projection

Ce dossier va contenir 5 classes :

* La classe « Calendar\_Manager »  pour la gestion des paramètres de calendrier : cela est nécessaire pour pouvoir calculer les dates d’amortissement, de capitalisation des intérêts ainsi que pour l’application des taux de RA/RN.
* La classe « Rate\_Manager » pour la gestion des taux : il est nécessaire de faire un mapping entre les taux donnés dans le fichier LDP et ceux contenus dans le fichier de taux.
* La classe « Palier\_Manager » pour la gestion des paliers : il faut pouvoir stocker l’ensemble des informations du fichier PAL dans une structure de données afin de pouvoir effectuer la projection sur ces contrats.
* La classe « Data\_Versements\_PEL » : une classe propre au PEL car la projection de ce produit nécessite d’être traitée de manière à part entière en comparaison des différents crédits.
* La classe « Projection\_Manager » : cette classe commence à projeter les données et centralise les variables calculées dans les différentes classes ci-dessus.

### 4.3.1 Classe : class\_calendar\_manager

Cette classe a pour objectif de calculer les différents paramètres de calendrier en fonction de la DAR. Elle va par exemple calculer la date de départ et de fin d’amortissement, le début des périodes d’application des taux RA/RN ainsi que les périodes où les intérêts sont capitalisés.

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_mod\_params): s’instancie à partir de la classe data\_model\_params

**Fonctions clés :**

* **def** get\_calendar\_coeff(self): cette méthode calcule les paramètres du calendrier en fonction de la DAR et de la durée de projection. Elle remplit les attributs date\_deb, date\_fin, etc., qui stockent les mois de début et de fin de chaque contrat.
* **def** generate\_proj\_begin\_dates(data): cette méthode va modifier son input « data » de sorte à stocker dans « data » les dates de début et de fin d’amortissement et effectuer un décalage par rapport à la DAR si nécessaire.
* **def** calculate\_month\_shift\_for\_non\_monthly\_amor(self, data, col\_freq\_amor, maturity\_month, mois\_depart\_amor): réalise un décalage par rapport à la DAR car certains contrats ne sont pas amortis mensuellement, il est alors nécessaire de recalculer la date d’amortissement. Pour cela, il faut partir de la date de maturité = dernier amortissement, si le profil d’amortissement est de 6, il faut alors remonter de 6 en 6 pour trouver la dernière date d’amortissement. Cela permet de savoir de combien il faut décaler de mois pour pouvoir trouver la bonne date d’amortissement.
* **def** load\_rarn\_calendar\_parameters(self, data\_hab, mois\_depart, current\_month, dar\_mois, n): cette méthode calcule les paramètres liés aux produits financiers tels que la durée résiduelle dynamique (DRAC) afin d’appliquer les taux RA/RN.
* **def** get\_nb\_days\_per\_period(self, data\_hab, current\_month, mois\_depart\_int, mois\_fin\_int, dar\_mois, n, t): cette méthode va calculer en nombre de jours entre deux plots d’amortissement les durées de plusieurs variables comme la durée d’amortissement.
* **def** calculate\_begin\_and\_end\_months2(self, value\_date\_month, maturity\_date\_month, val\_date, mat\_date, first\_amor\_date, broken\_period, dar\_mois, typo=**"amor"**): fonction qui calcule par rapport à la DAR les mois de départ et de fin de l'amortissement, des ra/rn et du calcul des intérêts. Pour rappel, un contrat à amortissement mensuel commence à s'amortir en fin de période. Exemple: contrat de date de valeur 15/10/2024. La première période a lieu entre le 15/10/2024 et le 15/11/2024. L'amortissement tombe le 15/11/2024 soit en fin de première période donc le mois de la value date + 1. Quelques remarques peuvent être ajoutées :
* Si le contrat n'est pas forward, alors la première période est DAR-DAR+1 et l'amortissement tombe en DAR +1.
* Cette règle souffre des exceptions: lorsque la date d'amortissement est explicitement donnée dans les données, auquel cas, l'amortissement a lieu le mois de la date d'amortissement + 1 (fin de période). Cependant, il faut adapter cette date en cas de s’il y a une broken period.  
  + S’il y a une broken period en "SS" et si le jour de la date de valeur est inférieur au jour de la date de maturité, alors la première période est écourtée et tombe le mois même de la date de valeur. Exemple: pour une date de valeur du 14/10/2023 et une date de maturité au 16/02/2040, la première période est du 14/10/2023 au 16/10/2023, l'amortissement tombe le 16/10/2023 et donc le mois de la date de valeur.
  + S’il y a une broken period en « SL », et si le jour de la date de valeur est supérieur au jour de la date de maturité, alors la première période est allongée et le premier amortissement tombe au mois de la date de valeur +2. Exemple: 14/10/2023 et 10/02/2023, alors la première période est 14/10/2023-10/12/2023 et l'amortissement tombe au mois de la date de valeur + 2
  + Si la broken period est en end long et la le jour de la date de valeur supérieur à la date de maturité alors la dernière période est (mois\_mat - 2 : mois\_mat). Donc notre cas, on réduit artificiellement le mois maturité de 1 avant de l'allonger plus tard

* La fin de l'amortissement a lieu le mois de la date de maturité et le capital tombe à zéro ce mois là. La dernière période  
  donc (mois\_mat - 1 : mois\_mat).

### 4.3.2 Classe : class\_palier\_manager

Cette classe traite la gestion des paliers, avec une attention particulière portée aux contrats du produit PEL. En particulier, cette classe contient un dictionnaire avec l’ensemble des informations associés aux paliers pour chaque contrat (la nouvelle échéance, le nouveau taux, etc..).

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_data, nb\_months\_proj): s’instancie à partir de la classe data\_manager et du nombre de mois de projection

**Fonctions clés :**

* **def** prepare\_palier\_data(self, clas\_proj, cls\_cal, cls\_data\_palier, cls\_ra\_rn\_params): Cette méthode prépare les données des paliers en les stockant dans un dictionnaire. Ce dictionnaire contient entre autres les contrats ayant des paliers, la valeur de la nouvelle échéance, le nouveau taux considéré, la nouvelle date d’amortissement. Aussi, cette fonction traite spécifiquement les contrats du produit "p-pel" en appelant la méthode create\_palier\_for\_pel\_contracts et obtient le calendrier des paliers en utilisant get\_palier\_schedule.
* **def** get\_palier\_schedule(self, cls\_cal, cls\_data\_palier, data\_hab, mois\_depart, period\_end\_date, contracts\_updated, cls\_proj, n, t): cette fonction renvoie un dictionnaire dic\_palier qui contient une variable is\_palier qui renvoie True or False si le contrat contient des paliers ou non. Également, dic\_palier contient un dataFrame dans la variable « palier\_schedule » contenant tous les contrats qui ont un palier, les dates des différents paliers, les différents taux et les différentes échéances.
* **def** format\_data\_before\_joining\_with\_palier\_mapping(self, data\_hab, mois\_depart, period\_begin\_date, n, t): Sélectionne toutes les informations dans le fichier LDP qui vont permettre de faire la jointure avec le fichier PALIER.
* **def** generate\_palier\_schedule(self, data\_hab, contracts\_updated, cls\_data\_palier): Fonction effectuant des sélections pour éviter de considérer des données « inutiles ». Par exemple, la date du palier ne peut pas être supérieure à la date de maturité. Cette fonction met aussi à jour les dates de palier en les décalant si nécessaire et effectue un formatage pour créer le dictionnaire.

### 4.3.3 Classe : class\_rate\_manager

La classe « Rate\_Manager » est responsable de la gestion des taux (récupération des bons taux pour chaque contrat), du formatage des données et du calcul des taux dans le cas des taux variables.

**Pour rappel, dans le cas des taux variables**, nous considérons un taux moyen unique pondéré par la durée restante. Il se formalise ainsi :

Avec:

A partir de ce taux, nous calculons une unique échéance **.**

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_mod\_params): s’instancie à partir de la classe data\_model\_params

**Fonctions clés :**

* **def** prepare\_curve\_rates(self, cls\_proj, tx\_params): Cette fonction va récupérer les taux associé à chaque contrat. Si c’est un taux fixe, cette méthode va récupérer le taux directement dans le fichier LDP. Dans le cas des taux variables, cette fonction va faire un mapping entre le fichier LDP et le fichier de taux pour pouvoir associé les bons taux à chaque contrat. Les taux sont ensuite stockés dans l'attribut data\_rate\_sc.
* **def** get\_scenario\_rates(self, data\_ldp, data\_rate\_sc, fixing\_calendar, current\_month, col\_fix\_next\_date, col\_curr\_rate, col\_fix\_per\_num, col\_mult\_fact, col\_mkt\_spread, col\_floor, col\_cap, col\_tenor\_num, col\_accrual\_curve, col\_accrual\_basis, col\_fix\_per, mat\_date, mat\_date\_month, dar\_mois, col\_fixing\_nb\_days, n, t) : cette fonction est la fonction principale pour obtenir dans le cas des taux variables, les taux pour chaque contrat. Cela s’effectue en trois temps :
  1. Obtenir la fixing\_date afin de connaître quels taux à appliquer.
  2. Une fois les taux connus, ces derniers sont ajustés en fonction de la base de calcul.
  3. Rendre les taux périodiques : i.e. pour chaque période, le taux obtenu avec la fixing\_date va s’appliquer. Donc si on a un taux de 1% sur une période de 3 mois, ce taux de 1% va alors s’appliquer sur ces trois mois.
* **def** mean\_sc\_rate(self,, rate\_sc, n, drac\_amor, end\_month, begin\_month, current\_month, t): Cette fonction s’applique dans le cas des taux variables afin de calculer un taux moyen pondéré par la durée restante (voir formule *(A)* ).

### 4.3.4 Classe : class\_projection\_manager

La classe Projection\_Manager est conçue pour finaliser la préparation les données afin de faire les calculs d'amortissement. Elle va notamment les formater et calculer les paramètres pour commencer la projection.

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_mod\_params): s’instancie à partir de la classe data\_model\_params.

**s**

**Fonctions clés :**

* **def** split\_data\_for\_projection\_data(self, cls\_data\_ldp): cette fonction partitionne les données en deux catégories. En particulier, après l’ensemble de la phase de préparation des données, elle va créer un DataFrame « data\_ldp » contenant l’ensemble des colonnes nécessaires à la projection à partir des « proj\_vars ». Un autre DataFrame est aussi créé « data\_optional » à partir des colonnes « other\_vars ».
* **def** load\_dimensions(self, data\_ldp, dar\_mois): Cette fonction détermine les dimensions de la projection, notamment le nombre de contrats (n), le nombre de périodes de projection (t), et la période maximale de projection (t\_max). Elle prend également en compte la durée des contrats douteux.
* **def** build\_current\_month(self): cette fonction crée une matrice qui représente le mois courant par rapport à la DAR pour chaque contrat.
* **def** prepare\_proj\_params(self, cls\_data\_ldp, cls\_data, cls\_format, cls\_cal): Cette fonction prépare les différents paramètres pour le calcul de l’amortissement, elle va effectuer les opérations suivantes :
  1. Récupération des profils d’amortissement pour chaque contrat.
  2. Création d’une variable pour savoir s’il faut capitaliser les intérêts ou non.
  3. Récupération de la fréquence de remboursement du crédit (annuel, mensuel, trimestriel, semestriel).
  4. Récupération de la fréquence à laquelle les intérêts sont capitalisés, ex : les intérêts se cumulent un an et se rajoutent au nominal. Si c’est mensuel, il faut payer les intérêts mensuellement.
  5. Calcul de la fréquence de périodicité de fixation des taux (se calcule en fonction de la périodicité du paiement).
  6. Récupération des taux en fonction de la périodicité de fixation des taux.
  7. Calcul des différents paramètres de calendrier comme le début de l’amortissement, la date de début des ra/rn, des intérêts.

## 4.4 Calcul des taux RA/RN avec les classes du dossier « rarn »

### 4.4.1 Classe : class\_rarn\_liq\_manager

La classe RARN\_LIQ\_Manager calcule et ajuste des taux de RA/RN en fonction du produit.

**Instanciation :**

* **def** \_\_init\_\_(self, class\_static\_amortization): s’instancie à partir de la classe « class\_static\_amortization » qui sera détaillée par la suite.

**Fonctions clés :**

* **def** get\_rarn\_params(self, cls\_stat\_amor, name\_product, data\_hab, mois\_depart\_rarn, drac\_init, drac\_rarn, current\_month, adj\_delta\_days, rarn\_periods, tx\_params, n, t): Fonction principale pour calculer les taux de RA/RN pour n’importe quel produit. Celle-ci fait appel à plusieurs fonctions détaillées ci-dessous.
* **def** apply\_ra\_rate\_model(self, data\_hab, current\_month, mois\_dep\_rarn, rarn\_periods, n, t): Cette méthode calcule des taux de RA mensuels pour les produits ayant des taux de RA statiques.
* **def** adjust\_to\_etab(self, data\_hab, tx\_ra, tx\_rn, n): Cette méthode calcule des taux de RA/RN mensuels pour les crédits immobiliers pour chaque établissement.
* **def** apply\_ra\_rn\_rate\_model(self, data\_hab, rates, tx\_cms, drac\_rarn, drac\_init, delta\_days, mois\_dep\_rarn, current\_month, n, t): Cette méthode est la méthode principale pour le calcul les taux de RA/RN dans le cas des crédits immobiliers à taux fixe. De manière plus générale, elle calcule des taux dynamiques en fonction de la DRAC et les mensualise pour chaque établissement. Cela s’effectue via les formules :

+ 0,5%)

* **def** get\_rate\_arbitrage\_from\_cms(self, data\_hab, dic\_tx\_swap, drac\_rarn, current\_month, mois\_depart\_rarn, n, t): Cette méthode calcule le taux d'arbitrage nécessaire au calcul des taux dynamiques de RA/RN comme l’indique la formule ci-dessus.
* **def** get\_rate\_renego\_from\_cms(self, data\_hab, dic\_tx\_swap, drac\_rarn, current\_month, mois\_depart\_rarn, n, t): Cette méthode calcule le taux de renégociation.

### 4.4.2 Classe : class\_rarn\_taux\_manager

La classe RARN\_TAUX\_Manager est conçue pour calculer les taux RA/RN pour les comptes à termes (CAT) => reprend les mêmes fonctions que taux\_liq sauf pour les P-CAT où elle va recalculer les RA en fonction d’un modèle simple qui consiste à regarder la différence entre le taux que tu aurais eu sur le marché si tu avais pris ton CAT aujourd’hui vs le taux courant du CAT.

C’est fait ici :

**def** apply\_RA\_CAT\_rate\_model(self, tx\_ra\_liq, rates, tx\_cms, mois\_dep\_rarn, current\_month):  
 delta\_tx = ne.evaluate(**'rates - tx\_cms'**)  
 tx\_ra = np.where(delta\_tx < 0, 1, tx\_ra\_liq)  
 tx\_ra = ne.evaluate(**"where(current\_month < mois\_dep\_rarn, 0, tx\_ra)"**)  
 tx\_ra = np.nan\_to\_num(tx\_ra)  
 **return** tx\_ra

Ex : j’ai un P-CAT existant de 10k avec 3% par an, sa date de maturité est dans 2 ans sachant que l’on ait le 30/06/2023. On va comparer le taux du CAT 3% vs le taux du CAT à un mois donné compris entre le 30/06/2023 et le 30/06/2025

On va par exemple regarder le taux EURIBOR 1Y le mois M12, on va le comparer à 3%, si c’est > 3% => RA = 100% => P-CAT tombe à 0, il retire l’argent car taux plus intéressant de l’Euribor1Y.

**TODO**

**Instanciation :**

**Fonctions clés :**

## 4.5 Calcul de l’amortissement avec les classes du dossier « amortization »

Ce dossier va contenir des classes permettant de réaliser les calculs d’écoulement. Pour cela, il se divise en deux sous-dossiers : « amortissement\_statique » et « amortissement\_scenario ». Le premier réalise les simulations d’écoulement sans prendre en compte les taux RN alors que dans le deuxième cas, le capital est réajusté en fonction de ces taux.

### 4.5.1 Dossier : amortissement\_statique

#### 4.5.1.1 Classe : class\_init\_ecoulement

Cette classe va réaliser le calcul de l’écoulement avant amortissement (i.e. le déblocage du nominal, la capitalisation des intérêts ainsi que le montant par mois à payer). Pour cela, il faut lister les contrats suivants et paramètres suivants:

- Ceux qui sont forward

- Ceux ayant une releasing\_date i.e. la date à laquelle le nominal va être libéré. Si la date est négative, alors elle est déjà passée et donc n’existe pas.

**4 calculs d’écoulements sont alors réalisés :**

* Ecoulements ayant une num\_rule et étant forward
* Ecoulements forward
* Ecoulements non forward mais décalés : cela correspond à la situation où la date d’amortissement (first\_amor\_date) est dans le futur par rapport à la DAR alors que la date pour laquelle le contrat commence (value\_date) est passée. Dans ce cas, la matrice d’écoulement est remplie par la valeur de l’outstanding jusqu’à la première date d’amortissement.
* Ecoulements par défaut : Dans ce cas, la première colonne de la matrice d’écoulement (qui représente le premier mois) est remplie par l’outstanding, les autres coefficients sont mis à 0.

**Instanciation :**

* **def** \_\_init\_\_(self, cls\_proj, cls\_rate, cls\_cal, cls\_vrsments) : s’instancie à partir de la classe « Projection\_manager », « Rate\_manager », « Calendar\_Manager » ainsi que « Data\_Versements\_PEL». Cette dernière classe est utile seulement dans le cas des PEL (PEL qui ne sont pas traités dans ce document).

**Fonctions clés :**

* **def** generate\_eclt\_before\_amortization(self, data\_ldp, outstd\_crd, nominal, rate, current\_month, dar\_mois, point\_depart, n, t): Fonction principale pour réaliser les calculs d’écoulement. Cette dernière va appeler des sous-fonctions pour réaliser les écoulements par défaut, forward etc.
* **def** ecoulements\_num\_rule(self, data\_ldp, is\_forward, is\_release\_rule, outstd\_crd, nominal, num\_rule, current\_month, t, n, rate, is\_capitalized, nb\_days\_an, value\_date, accrued\_interests, nb\_day\_m0, base\_calc): Cette fonction effectue un calcul d’écoulement pour les contrats forward avec une num\_rule. Il y a alors 3 grandes étapes : 1) obtention de la règle selon laquelle le nominal est débloqué 2) ajout des intérêts capitalisés 3) prolongement du nominal connu jusqu’à la date de maturité.
* **def** ecoulements\_forward\_def(self, is\_forward, is\_release\_rule, current\_month, nominal, point\_depart, value\_date, releasing\_date, releasing\_date\_r, outstd\_crd, is\_capitalized, rate, nb\_days\_an, accrued\_interests, nb\_day\_m0, base\_calc, n, t) : Il y a deux cas à distinguer ici, ceux possédant une date à laquelle le nominal va être libéré (releasing\_date) et ceux dont cette date est inconnue.
  1. Pour le premier cas, la matrice d’écoulement est remplie de 0 (ou le outstanding s’il y en a) jusqu’à la releasing\_date. Après cette date, la matrice est remplie par la valeur du nominal.
  2. Pour le deuxième cas, la matrice d’écoulement est remplie par la valeur de l’outstanding jusqu’à la date à laquelle le contrat commence (value\_date) pour être conforme à RCO. Après cette date, la matrice est remplie par la valeur du nominal.
* **def** add\_capitalized\_interests(self, ec\_num\_rule, is\_capitalized, rate, \_current\_month, \_value\_date, \_nb\_days\_an, \_accrued\_interests, \_nb\_day\_m0, \_base\_calc): Fonction pour ajouter les intérêts capitalisés au nominal.

#### 4.5.1.2 Classe : class\_static\_amortization

Classe principale pour calculer l’amortissement. Cette dernière va se diviser en plusieurs sous classes pour effectuer les amortissements par défaut, par palier, ceux des contrats douteux et celui des cash\_flows. Etant donné que ces classes nécessitent des calculs similaires, un fichier à part entière « amortization\_commons.py » est créé.

**Instanciation :**

* **def** \_\_init\_\_(self, class\_init\_ecoulement): s’instancie à partir de la classe « Init\_Ecoulement » vue précédemment.

**Fonctions clés :**

* **def** generate\_static\_amortization(self, tx\_params): fonction principale pour réaliser l’amortissement. Elle se divise en 4 parties. Une première pour l’amortissement par défaut, puis par palier, puis pour les cashflows et une dernière pour les contrats douteux. Chacun de ces amortissements est implémenté dans une classe spécifique.

#### 4.5.1.3 Classe : class\_default\_amortization

**Modélisation :**

Pour rappel, la formule décrivant l’évolution du capital restant est la suivante :

C’est cette formule qui est implémentée dans PASS-ALM.

Remarque : la formule suivante équivalente à celle-ci dessus pourrait aussi être utilisée  :

**Instanciation :**

* **def** \_\_init\_\_(self, class\_init\_ecoulement): s’intancie à partir de la classe « Init\_Ecoulement »

**Fonctions clés :**

* **def** calculate\_default\_amortization(self, data\_hab, cap\_before\_amor, ech\_val, sc\_rates, ech\_rate, begin\_capital, current\_month, amor\_freq, capital\_amor\_shift, begin\_month, amor\_begin\_month, amor\_end\_month, mat\_date, interests\_periods, ech\_const\_prof, linear\_prof, linear\_ech\_prof, duree, year\_nb\_days, capi\_freq, suspend\_or\_capitalize, capitalize, cap\_rate, accrued\_interests, nb\_days\_m0, orig\_ech\_value, interest\_cap\_shift, is\_fixed, t): Cette fonction va calculer l’amortissement à partir de la formule ci-dessus. A cette fin, il faut d’abord charger les paramètres de calendrier (variables : begin\_date, interests\_schedule, amortization\_schedule). À la suite de cela, il faut ajuster (ou non) les taux d’intérêts ainsi que les accruals à partir de ces paramètres de calendrier. Puis, un premier calcul permet d’obtenir le terme : « » et un deuxième « ». À la suite de cela, le capital restant peut alors être calculé dans un dernier temps.

#### 4.5.1.4 Classe : class\_palier\_amortization

**Modélisation :**

Pour rappel, la formule implémentée de l’écoulement du capital est la suivante :

**Instanciation :**

* **def** \_\_init\_\_(self, class\_init\_ecoulement): s’instancie à partir de la classe « Init\_Ecoulement »

**Fonctions clés :**

* **def** calculate\_palier\_amortization(self, data\_hab, dic\_palier, cap\_before\_amor, begin\_capital, current\_month, capi\_freq, begin\_month, amor\_begin\_month, amor\_end\_month, mat\_date, dar\_mois, interests\_periods, cle\_contrat, nb\_days\_an, duree, accrued\_interests, nb\_day\_m0, capi\_rate, sc\_rates, sc\_ftp\_rates, tx\_params, t, tombee\_fixing, period\_fixing): Fonction calculant le capital en implémentant la formule ci-dessus. La démarche est sensiblement équivalente à celle réalisée dans le cas des échéances constantes, une grande partie des fonctions utilisées sont communes. Les étapes principales sont :

**Pour chaque palier :**

* + - Récupération des différents paramètres de palier (paramètres de calendrier, les taux, le type de profil). Il est important de noter que le « amortization\_schedule » peut changer d’un palier à un autre.
    - Ajustement (ou non) des taux d’intérêts ainsi que des accruals à partir de ces différents paramètres (**toujours avec la fonction** adapt\_compounded\_rate\_to\_periodicity()).
    - Calcul du terme :  **(voir page suivante)**
    - Calcul du terme : et somme desdéjà calculés **(voir dans les paragraphes suivants)**
    - Calcul du capital restant

Nous allons détailler ici les termes évoqués ci-dessus.

I) Pour pouvoir réaliser le calcul du terme **, ce dernier va se diviser en deux temps car le taux peut changer d’un palier à un autre.**

Pour pouvoir faire ce calcul, nous introduisons la matrice suivante :

Avec:

Où:

: taux du palier pour le contrat ***i***

**fonction indicatrice = 1 pour et 0 sinon**

**En conséquence, est une matrice contenant exclusivement des 0 en dehors de la période du -ème palier.**

Pour pouvoir obtenir la matrice , nous devons calculer la matrice à chaque palier.

En notant nous pouvons en déduire la matrice , introduisons à nouveau le vecteur:

**La matrice s’obtient alors après avoir fini de parcourir chacun des paliers.**

**Après calcul de le terme est généré.**

Aussi, pour calculer le terme :

En introduisant **,** on peut écrire cette somme sous la forme:

**Nous allons alors calculer à chaque palier :**

**En sommant les pour chaque palier, nous pouvons en déduire .**

**À la suite de cela, le capital restant peut alors être calculé dans un dernier temps.**

**Fonctions clés :**

* **def** load\_palier\_parameters(self, data\_hab, palier\_schedule, current\_month, pal\_nb, amor\_end\_month, tx\_params, dar\_mois, sc\_rates, sc\_ftp\_rates, max\_palier, n, t, tombee\_fixing, period\_fixing): Charge les paramètres : freq\_amor, cap\_interests, ech\_const\_prof, linear\_prof, rate, interests\_schedule\_b1 (intérêts du palier b1), interests\_schedule\_b2 (intérêts du palier b1 jusqu’au palier b2) via la variable palier\_schedule.
* **def** get\_rate\_factor\_for\_nominal\_palier(self, rate\_mtx, interests\_schedule\_b1, interests\_schedule\_b2, interests\_calc\_periods, nb\_days\_an): Cette fonction calcule les éléments suivants :

**et donc évalue à chaque palier.**

Comme expliqué précédemment, les coefficients de la matrice vont être mis à 0 en dehors de la période du -ème palier. La différence principale en comparaison du cas de l’échéance constante est de considérer une liste non vide pour la variable « interests\_schedule\_b2 ». Cela permet de préciser le palier en cours ainsi que de placer les 0 aux bons endroits dans la matrice **.**

**- def** get\_rate\_adjusted\_ech\_palier(self, rate\_adjusted\_ech\_cum, compounded\_rate, palier\_schedule, rate, begin\_capital, prof\_ech\_const, prof\_linear, prof\_linear\_ech, pal\_nb, freq\_amor, calendar\_amor\_adj,  
interests\_schedule\_b1, suspend\_or\_capitalize, interests\_schedule\_b2, nb\_periods, current\_month, mois\_pal\_i, is\_fixed, rate\_adjusted\_nominal, amor\_end\_month, capital\_amor\_shift, n, t): Cette fonction calcule les éléments par palier :

Et effectue une : somme de ces termes pour obtenir à la fin :

Pour ce faire, on procède exactement comme dans le cas des amortissements par défaut.

1. Calcul de l’échéance constante via la Fonction 1.5.2.3.1.4 get\_ech\_const()
2. Calcul du terme via la Fonction 1.5.2.3.1.5 get\_compounded\_rate\_for\_ech(). La différence avec les amortissements par défaut réside en l’utilisation de la variable « interests\_schedule\_b2 » au lieu d’une liste vide pour pouvoir bien évaluer afin de calculer la somme.
3. Calcul de et somme de tous les déjà calculés :

rate\_adjusted\_ech = ne.evaluate(**"const\_ech \* adj\_rate\_for\_ech"**)  
rate\_adjusted\_ech = np.where(capitalized\_interests, 0, rate\_adjusted\_ech)  
rate\_adjusted\_ech\_cum = ne.evaluate(**"rate\_adjusted\_ech\_cum + rate\_adjusted\_ech"**)

* **def** get\_rate\_adj\_nominal\_palier(self, rate\_factor\_cum, begin\_capital): A partir du : calculé précédemment, cette fonction calcule :

Pour en déduire

#### 4.5.1.5 Classe : class\_cash\_flow\_amortization

**Instanciation :**

* **def** \_\_init\_\_(self, class\_init\_ecoulement): s’instancie à partir de la classe « Init\_Ecoulement »

**Fonctions clés :**

* **def** get\_amortized\_capital\_with\_cash\_flows(self, data\_hab, cap\_before\_amor, data\_cf, current\_month, dar\_mois, t): Cette fonction calcule le capital amorti en fonction des cash\_flow. Elle commence par ajuster les cash\_flow en fonction des dates de valeur, puis soustrait ces cash\_flow initiaux pour obtenir le capital amorti. Les valeurs négatives sont ensuite converties en zéro, et la colonne du capital initial est ajoutée au début du résultat. Si aucun capital avant amortissement n'est disponible, un tableau de zéros est renvoyé.

Amortissemetn cash\_flow : C(t) = C0 – cumsum(Cf\_d)d\_0 à t

#### 4.5.1.6 Classe : class\_douteux\_amortization

**Classe réalisant l’amortissement des contrats douteux.**

**Modélisation :**

Avec les échéances mensuelles du capital :

Et :

capital au tempscomme dans le cas d’un écoulement sain.

**Instanciation :**

* **def** \_\_init\_\_(self, class\_init\_ecoulement): s’instancie à partir de la classe « Init\_Ecoulement ».

**Fonctions clés :**

* **def** calculate\_non\_performing\_amortization(self, capital\_ec, current\_month, is\_hab\_bon, t): Calcul le capital restant lors d’un écoulement douteux., Pour cela, il faut considérer les écoulements comme s’ils avaient été sains, cela permet d’obtenir la variable . Ensuite, contrairement aux écoulements « sains », l’échéance n’est pas enlevée à 100% car le client ne peut pas complètement rembourser ses échéances. Un calendrier de remboursement se déroulant sur 73 mois est alors créé. Ce calendrier consiste à rembourser qu’une partie d’une échéance par mois. Exemple : soit un nominal de 100 euros, un premier remboursement de 70 euros est effectué le premier mois, de 15 euros le deuxième, etc.

On appelle 𝑬𝑪𝑯𝑪𝑨𝑷(𝒕,𝒊) cet écoulement de l’échéance en t de t à t + 72 au pas i. Il y a donc ***k*** -échéances, dont certaines qui peuvent être nulles. Pour calculer les échéances, on récupère le capital à t et on lui soustrait le capital à t-1, on regarde le delta capital entre deux plots.

Voici une synthèse de la procédure d’amortissement d’un profil douteux :

* Complétion avec des 0 dans la matrice d’écoulement jusqu’à la date de projection si durée > 73
* Le profil d’amortissement a besoin d’être « shifté » pour connaître les échéances, voici une explication ci-dessous :
  + Supposons qu’il paye 100 à chaque échéance :
    - * M1:100 – M2:100 – M3:100
  + Remboursement échéance M1 :
    - * 70%\*100 = 70 – 3%\*100 = 3 – 3%\*100 = 3
  + Remboursement échéance M2 :
    - * 0 – 70%\*100 = 70 – 3%\*100 = 3
  + Remboursement échéance M3 :
    - * 0 – 0 – 70%\*100 = 70
* L’amortissement s’obtient en faisant la somme des remboursements (donc de chaque colonne), de plus chaque ligne à son profil d’amortissement, il faut également les sommer. Cela revient à un problème à 3 dimensions, une dimension pour le nombre de contrats, t+1 le pas de projection, t+1 écoulement pour le profil
* Pour en déduire les différentes échéances mensuelles pour un contrat, une somme est réalisée sur l’axe 1 de cette matrice 3D.
* Pour obtenir le capital restant, il faut lui soustraire les échéances obtenues.

#### 4.5.1.7 Notes sur amortization\_commons

##### 4.5.1.7.1 Fonction adjust\_interest\_rate\_to\_profile\_and\_accruals ()

**Description fonctionnelle :**

Cette fonction va générer une matrice de taille nxt avec la valeur des taux à chaque pas de temps. Plus particulièrement, cette matrice va tenir compte des intérêts accumulés (accruals) avant la DAR durant le mois 0 (la première colonne). Illustrons cela à l’aide d’un exemple : imaginons la DAR le 01/01/2023 avec des intérêts accumulés du 15/12/2022 au 01/01/2023 pour un contrat donné. Dans ce cas, il va falloir rajouter aussi les intérêts du 01/01/2023 au 15/01/2023 pour réaliser l’amortissement.

##### 4.5.1.7.2 Fonction get\_rate\_factor\_for\_nominal()

**Description fonctionnelle :**

Cette fonction va générer la matrice suivante :

Pour calculer le terme , on effectue :

Si les intérêts sont capitalisés, il faut alors les rajouter au nominal, cela revient à changer en :

Soulignons que la matrice va contenir que des intérêts égaux à 1 avant l’amortissement pour éviter de les capitaliser. Cela revient à dire qu’une fois les intérêts capitalisés (si nécessaire), ces derniers sont rajoutés seulement lorsque la période d’amortissement commence.

**Remarque :** À la suite de cela, il est possible de calculer le terme à l’aide d’un cumprod sur la sortie du cette fonction i.e. :

compounded\_rate\_nom = rate\_factor\_nom.cumprod(axis=1)

Ainsi :

**En notant le vecteur suivant :**

**On peut alors en déduire la matrice suivante pour chaque contrat et chaque pas de temps :**

##### 4.5.1.7.3 Fonction adapt\_compounded\_rate\_to\_periodicity()

**Description fonctionnelle :**

Cette fonction consiste à « constantiser » les taux d'ajustement entre les plots d'amortissement lorsque l'amortissement est non mensuel.

Pour ce faire, il faut récupérer les contrats répondants à cette condition via la variable amortization\_schedule et le compounded rate calculé précédemment i.e. la matrice .

Pour « constantiser » les taux entre deux amortissements, les coefficients de la matrice sont changés en « nan » (entre deux plots d’amortissement). Cela se justifie car la partie des intérêts ne doit pas être rajoutée au nominal, elle doit rester constante sur cette période.

Ensuite, lorsque l’amortissement tombe, on remplit la nouvelle matrice via la fonction « ffill » de numpy qui prend la dernière valeur non nulle des colonnes précédentes et change les valeurs « nan » à partir de cette valeur.

**Remarque : le terme peut alors être calculé en effectuant :**

rate\_adjusted\_nominal = ne.evaluate(**"compounded\_rate\_nom \* begin\_capital"**)

##### 4.5.1.7.4 Fonction get\_ech\_const()

**Description fonctionnelle :**

Cette fonction permet de calculer .

* Dans le cas des échéances constantes, si l’échéance n’est pas fournie dans les données, il faut l’évaluer. Pour cela, il faut utiliser la formule suivante:
* Dans le cas des profils linéaires, la formule suivante est utilisée :

Soulignons que pour ce cas, il faut adapter la durée en fonction du profil d’amortissement, la « duree » étant le nombre de pas de temps avant de finaliser l’amortissement.

Par exemple, si la « duree » est trimestrielle => prêt sur 3 ans = 3 x 12 mois = 36 or si l’échéance est trimestrielle => 36/3 = 12).

##### 4.5.1.7.5. Fonction get\_compounded\_rate\_for\_ech()

**Description fonctionnelle :**

Fonction permettant de calculer le terme , une première partie consiste à calculer le rate\_factor comme effectué précèdemment, i.e.:

À la suite de ce calcul, pour chaque contrat, les coefficients de la matrice sont mis à 1 avant le début de l’amortissement. Il est alors possible de calculer via un cumprod (de la même manière que dans la fonction get\_rate\_factor\_for\_nominal()

Il faut alors la somme des inverses de i.e.

Enfin, dans un dernier temps est adapté à la périodicité avec la adapt\_compounded\_rate\_to\_periodicity().

##### 4.5.1.7.6. Fonction calculate\_remaining\_capital()

Fonction calculant le capital restant i.e. dans le cas des amortissements par défaut:

Pour cela, on considère lasortie de la fonction get\_rate\_factor\_for\_nominal() (après ajustement de la périodicité) = rate\_adj\_nominal = et le produit des sorties des fonctions fonction get\_compounded\_rate\_for\_ech() et get\_ech\_const() = rate\_adj\_ech =

On effectue alors la différence des deux pour obtenir le capital restant :

remaining\_capital[:, 1:] = ne.evaluate(**"rate\_adj\_nominal - rate\_adj\_ech"**)

⬄

### 4.5.2 Dossier : amortissement\_scenario

#### 4.5.2.1 Classe : class\_amortization\_renegociated

**Cette classe va réaliser l’amortissement des montants renégociés.**

L’amortissement des montants de capital renégocié est l’opération la plus gourmande en temps de l’ensemble du processus. En effet, à chaque pas de temps et pour chaque contrat, un nouveau contrat est généré et doit subir un amortissement spécifique, car le taux du nouveau contrat dépend des nouvelles conditions de marché et des caractéristiques spécifiques du contrat initial.

Ce nombre important de contrats ralentit considérablement les temps de simulation. Une linéarisation du processus d’amortissement de ces contrats peut se justifier, d’autant que le part du capital des contrats renégociés dans le capital total est faible, même si elle va de manière croissante.

Dès lors, PASS-ALM fait le choix de linéariser l’amortissement des contrats renégociés afin de gagner en temps de calcul.

En partant des hypothèses précédentes, on peut calculerl’échéance mensuelle du capital amorti d’un contrat de durée ***d***, de nominal ***1*** et de taux mensuel renégocié ***r*** selon la formule classique :

En utilisant la formule de Taylor d’ordre 2 pour car les taux d’intérêts sont relativement proches de zéro, on obtient pour ***PPMT(t)*** :

On peut en déduire une formule fermée du capital renégocié au mois taprès avoir posé A = :

Et en généralisant avec des nominaux non unitaires, on obtient :

Grâce à ces différentes approximations, on peut donc obtenir une formule fermée du capital restant au mois i, à même d’accélérer le calcul de l’amortissement des contrat renégociés.

Au final le capital renégocié total au temps t (LEG N RCO) pour un contrat est égal à la somme des capitaux renégociés avant t :

**Instanciation :**

* **def** \_\_init\_\_(self, class\_static\_amortization): s’instancie à partir de la classe « Static\_Amortization »

**Fonctions clés :**

* **def** calculate\_amortization\_dev2\_no\_infine(self, credit\_depart, rate, current\_month, mois\_depart\_amor, duree, prof\_ech\_const, mois\_reneg): Calcule le terme **pour les contrats ayant une durée de maturité limitée :**
* **def** calc\_const\_dev2(self, rate, duree): Calcule le terme :
* **def** calculate\_amortization\_infine(self, credit\_depart, current\_month, mois\_depart\_amor, mat\_date, dar\_mois): Calcule le terme **pour les contrats ayant un amortissement infine**

## 4.6 Calcul des indicateurs avec le dossier : indicators

Ce dossier va calculer les indicateurs suivants : LEM (Gap de liquidité moyen mensuel), la MNI (Marge Nette d’Intérêt aux taux client).

**Cependant, en fonction du produit considéré, PASS-ALM calcule ces indicateurs de manière différente.**

Avant d’expliciter les différences entre les produits, rappelons la formule du capital mis à jour en tenant compte des effets RA/RN :

**Cette dernière sera utilisée dans toute la suite.**

* **Pour le cas des produits ayant des taux de RA statiques :**

On pose d(t) et m(t) respectivement, le jour de tombée de l’amortissement dans le mois considéré et le nombre de jours du mois considéré.

Il faut également noter df(t) le jour de tombée de la date de fixing du mois considéré. Alors les indicateurs susmentionnés sont calculés de la façon suivante :

* La MNI statique quant à elle, est calculée sur des mois calendaires. Au mois i, elle est donnée par :

* **Pour le cas particulier produits ayant des taux de RA/RN dynamiques (dans notre cas, les crédits immobiliers à taux fixe) :**

On pose d(t) et m(t) respectivement, le nombre de jour de la période dans le mois considéré et le nombre de jours du mois considéré. Alors les indicateurs susmentionnés sont calculés de la façon suivante :

* +

**Les indicateurs LEM et MNI totaux correspondent alors à :**

Soulignons que les variables valent 0 pour les contrats sans renégociation ().

### 4.6.1 Classe : class\_indicators\_calculation

Cette classe va calculer les indicateurs ayant des taux de RA statiques.

**Instanciation :**

* **def** \_\_init\_\_(self, class\_static\_amortization): s’instancie à partir de la classe « Static\_Amortization »

**Fonctions clés :**

* **def** update\_former\_leg\_capital(self, data\_hab, remaining\_capital, tx\_rarn, mois\_fin\_amor, current\_month, dar\_mois, douteux, n, strategy=**"liq"**, type\_ind=**"liq"**, period\_fixing=[]): Cette fonction met à jour le capital en tenant compte des taux de RA/RN (si les taux RN n’existent pas, ils valent 0) :
* **def** generate\_average\_capital(self, data\_hab, static\_leg\_capital, ec\_avt\_amor, mois\_depart\_rarn, current\_month, dar\_mois, period\_end\_date, douteux, n, t, data\_cf\_tombee, strategy=**"liq"**,  
  type\_ind=**"liq"**, period\_fixing = [], day\_fixing = []): fonction calculant le terme suivant :

* **def** calculate\_static\_leg\_mni(self, data\_hab, static\_leg\_capital, static\_leg\_capital\_db, day\_val, sc\_rates, ftp\_sc\_rates, period\_end\_date, douteux, current\_month, mois\_fin, n, t, day\_fixing, dar\_mois,strategy=**"liq"**, type\_ind=**"taux"**, period\_fixing = []): Fonction calculant le terme suivant :

### 4.6.2 Classe : class\_indicators\_renegociated

Cette classe va calculer les indicateurs ayant des taux de RA/RN dynamiques.

Pour cela, elle va instancier la classe « Amortization\_Renegociated » (correspond à la classe dans le dossier amortissement\_scenario) de sorte à calculer les indicateurs renégociés.

**Instanciation :**

* **def** \_\_init\_\_(self, class\_static\_amortization): s’instancie à partir de la classe « Static\_Amortization »

**Fonctions clés :**

* **def** get\_reneg\_capital\_and\_mni(self, data\_hab, remaining\_capital, rarn\_effect, tx\_rn, rate\_renego, mois\_depart\_amor, mois\_fin\_amor, drac\_amor, current\_month, mois\_depart\_rarn, interests\_calc\_periods, dar\_mois, douteux, is\_palier, dic\_palier, n, t): Calcule le capital renégocié ainsi que la MNI. Cette méthode appelle un sous-ensemble de fonction à cette fin que nous allons détailler.
* **def** calculate\_renegociated\_amount(self, remaining\_capital, rarn\_effect, tx\_rn, current\_month, mois\_depart\_rarn): Calcule le terme :
* **def** get\_renegociated\_capital\_simplified(self, data\_hab, amount\_renegociated, rate\_renego, current\_month, mois\_depart\_amor, mois\_fin\_amor, drac\_amor, dar\_mois, n, t, is\_infine=**False**):

Calcule le terme **avec le capital renégocié au temps *u***

* **def** generate\_avg\_renego\_capital\_simplified(self, data\_hab, reneg\_leg\_capital, n, t): Calcule le capital renégocié moyen correspondant à LEM :
* **def** calculate\_renego\_mni\_simplified(self, data\_hab, avg\_renego\_leg\_capital, rate\_with\_spread, n, t, t\_l): Calcule la MNI renégocié à partir du capital renégocié moyen :
* **def** calculate\_simplified\_agregated\_indics(self, reneg\_leg\_capital, avg\_reneg\_leg\_capital, mni\_renego, mni\_ftp\_renego): Calcule les termes suivants :