RAPPORT DE PSC INF14

Optimisation de la compensation entre positions sur des devises étrangères corrélées

2022-2023

Tess Breton, Barthélémy Bulteau, Madeleine Hueber,
Hector Rays, Gabriel Zeitoun

TABLE DES MATIÈRES

INTRODUCTION

Contexte et motivation du sujet

Suite à la crise financière de 2007-2008 et la faillite de plusieurs institutions bancaires dans le monde, les régulateurs internationaux ont réagi par le durcissement des réglementations en matière de provision de fonds propres. En effet, ces fonds servent à épancher les éventuelles pertes dues au dégonflement d'une bulle de prix. Toutefois, cet argent immobilisé sur un compte ne peut pas être investi et il est nécessaire d'optimiser, dans le cadre des règles en vigueur, le montant de cette provision.

Dans le cadre de ce PSC, nous nous intéressons à un type d'actif bien précis : celui des devises étrangères. Pour chacune d'entre elles, la Banque est dans l'une des situations suivantes : elle peut être créditrice et on parle alors de position "longue", ou être débitrice et on parle alors de position "courte". Dans tous les cas, les taux auxquels s'échangent ces monnaies sont variables et ces positions comportent un risque pour la banque et le régulateur lui impose de s'en prémunir.

Pour couvrir le risque de change associé aux positions de la Banque, le régulateur impose donc une provision de capital que la banque doit immobiliser. Le montant de cette provision est un pourcentage du montant d'exposition de la banque sur les devises, appelé Risk-Weighted Asset (RWA), qui correspond à la valeur maximale de deux sommes : celle sur les positions longues et celle sur les positions courtes, à laquelle on ajoute la position sur l'or. Ainsi, plus la banque souhaite prendre de positions, plus sa provision doit être élevée. On comprend alors l'importance d'un algorithme de compensation efficace entre devises afin de minimiser les montants immobilisés en provision.

En effet, le régulateur autorise la banque à compenser des positions opposées (créditrices-débitrices) entre ce qu'il considère comme des "devises corrélées". Deux devises sont considérées corrélées lorsque leurs cours sont corrélés au sens statistique : ils ont tendance à évoluer dans le même sens au cours du temps. Partant du principe que les variations de change pour des positions opposées sur de telles devises s'annulent, le risque de perte est minime, d'où l'autorisation de compensation du régulateur.

L'objectif de ce PSC est donc de déterminer les compensations optimales permettant de minimiser la provision de capital, à partir de positions initiales et de corrélations entre devises connues. L'approche naïve consistant à explorer l'ensemble des combinaisons envisageables n'étant pas suffisamment efficace, il s'agira donc d'identifier un ou plusieurs algorithmes fournissant des solutions exactes ou approchées, le tout avec une complexité raisonnable.

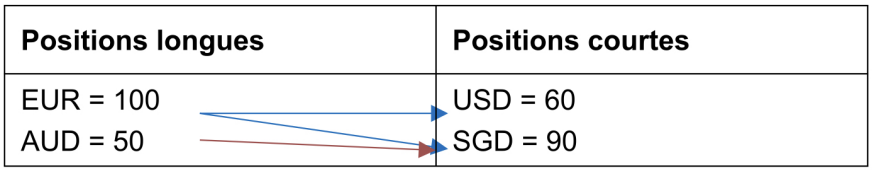


Figure 1: Exemple simple avec 4 monnaies

Illustration du problème par un exemple simple

Pour illustrer plus concrètement la situation, intéressons nous à un cas simple où seul quatre monnaies sont considérées et où la relation binaire de corrélation est représentée par des flèches. (Figure 1)

Premièrement, on peut remarquer que le bilan est bien à l'équilibre (ce qui sera toujours le cas lors de nos tests des algorithmes). En effet, les sommes des valeurs absolues des positions dans chaque colonne donne bien le même résultat : 150. C'est la valeur du RWA sans compensation.

Pour faire diminuer ce RWA, procédons par exemple aux compensations suivantes :

- On compense EUR et SGD à hauteur de 90 (on a alors EUR = 10 et SGD = 0)
- On compense EUR et USD à hauteur de 10 (on a alors EUR = 0 et USD = 50)

Plus aucune compensation supplémentaire n'est alors possible (AUD et USD ne sont pas corrélées) et le RWA (c'est-à-dire la somme de chaque colonne) vaut 50.

Toutefois, ces choix de compensation ne sont pas optimaux et il aurait fallu procéder comme suit :

- On compense EUR et USD à hauteur de 60 (on a alors EUR = 40 et USD = 0)
- On compense EUR et SGD à hauteur de 40 (on a alors EUR = 0 et SGD = 50)
- On compense AUD et SGD à hauteur de 50 (on a alors AUD = 0 et SGD = 0)

Toutes les monnaies étant désormais à 0, nous avons trouvé une solution optimale et le RWA est nul. C'est là que réside l'enjeu du problème, toutes les compensations ne sont pas équivalentes.

1

PREMIERS ALGORITHMES

1.1 Algorithme naïf

Après nous être renseignés sur l'aspect financier du sujet et avoir compris les enjeux de compensations entre devises, nous nous sommes concentrés sur un premier algorithme qui nous paraissait intuitif. Nous l'avons dénommé algorithme "naïf" car notre approche visait à tester un maximum de schémas de compensation différents.

La première question à laquelle nous nous sommes confrontés était donc celle du choix de l'ensemble des possibilités à explorer. En effet, toute compensation entre deux monnaies pouvant être partielle (à valeurs dans $\mathbb{R}$), nous ne pouvions pas recenser toutes les combinaisons possibles. Nous avons donc décidé de restreindre l'ensemble des schémas de compensation explorés de la manière suivante : nous avons introduit un ordre artificiel dans nos compensations, partant du principe que l'on compense le montant maximal disponible entre deux monnaies corrélées à un certain instant.

1.1.1 • Première version de l'algorithme naïf

Ainsi, nous avons implémenté une première version de notre algorithme naïf selon le principe suivant. Pour chaque ordre d'énumération des monnaies en position longue possible, on procède comme suit. Pour chaque monnaie en position longue, puis pour chaque ordre d'énumération des monnaies en position courte qui lui sont corrélées, on parcourt l'ensemble de ces monnaies, en compensant systématiquement le maximum possible. Une fois la devise en position longue entièrement compensée ou les position courtes corrélées épuisées, on passe à la position longue suivante. On répète ce procédé jusqu'à avoir traité toutes les monnaies en position longue. Au cours de l'exécution de l'algorithme, on garde en mémoire le RWA minimal et le meilleur schéma de compensation associé, que l'on renvoie à la fin.

Exemple :

En reprenant l'exemple de la partie 1.2, on peut illustrer ce processus pour faciliter la compréhension, avec un exemple d'ordres :

- On choisit tout d'abord un ordre sur les positions longues : EUR puis AUD.
- Puis, pour chaque devise en position longue, on choisit un ordre des positions courtes qui lui sont corrélées : Pour EUR, on choisit SGD puis USD. Pour AUD, l'ordre est imposé car il n'y a qu'une monnaie corrélée.

Avec ce choix d'ordres, on obtient un RWA de 50. L'algorithme va tester tous les ordres possibles de cette manière et garder en mémoire l'ordre optimal (qui aboutit au RWA minimal parmi les ordres testés).

1.1.2 • Seconde version de l'algorithme naïf

Quelques semaines après la conception de cette première version, nous avons eu une idée d'amélioration permettant d'éviter l'une des limitations majeures de cet algorithme. Par construction, il était impossible de commencer à compenser une monnaie en position longue puis d'y revenir après en avoir compensé une autre : chaque position longue était traitée une seule fois par l'algorithme. Nous avons donc décidé de modifier légèrement le principe de notre algorithme naïf de la manière suivante. Au lieu d'énumérer l'ordre des positions longues puis l'ordre des positions courtes corrélées, nous avons choisi d'énumérer l'ensemble des couples de devises corrélées (de la forme [position longue, position courte]). L'algorithme teste alors tous les ordres d'énumération possibles de ces couples, en compensant systématiquement le montant maximal autorisé. Pour l'implémentation, voir l'annexe 5.2.

1.2 Algorithme Glouton

1.2.1 • Principe de l'algorithme

Après avoir constaté qu'une approche aussi exhaustive n'était pas viable pour des raisons de complexité, nous avons décidé d'implément er un algorithme glouton. Bien que l'exactitude de la solution ne soit plus assurée, un tel algorithme a l'avantage de réduire considérablement la complexité temporelle en évitant un parcours exhaustif. En effet, le principe d'un algorithme glouton est de construire une solution étape par étape en s'efforçant d'obtenir la meilleure optimisation locale à chaque fois, dans l'espoir d'aboutir à un optimum global. [2]

Si la notion d'optimum global est claire dans notre problème, la définition de la meilleure optimisation locale l'est beaucoup moins. Par exemple, une stratégie envisageable serait de réaliser la compensation la plus importante à chaque étape dans le but de maximiser la valeur totale des compensations effectuées. Pourtant, cela n'est empiriquement pas la meilleure stratégie.

Pour aboutir à cette conclusion, nous avons codé le squelette d'un algorithme glouton (voir annexe 5.3) dont le choix de la compensation à effectuer est paramétrée par une heuristique. Les heuristiques suivantes ont été implémentées :

- f1 : choisit le couple de monnaies corrélées dont la somme des montants est la plus élevée.
- f2 : choisit le couple de monnaies corrélées dont la somme des montants est la plus faible.
- f3 : choisit le couple de monnaies corrélées qui sont les moins corrélées aux autres monnaies (on minimise le nombre de corrélations cumulé sur les deux monnaies).

1.2.2 • Efficacité de l'algorithme glouton

Après avoir imaginé et conçu cet algorithme pouvant être effectué avec différentes heuristiques, deux questions se sont posées. La première était la preuve de l'efficacité de l'algorithme. Au vu des résultats très satisfaisants de l'algorithme glouton présentés dans la section suivante, . En effet, nous voulions trouver une ρ-approximation pour chaque heuristique afin d'évaluer la qualité de l'algorithme glouton. Cependant, après avoir cherché de notre côté sans aboutir à un résultat concret, nous avons demandé de l'aide à notre coordinateur. Ce-dernier nous a indiqué que la réponse n'était pas évidente, mais pas forcément nécessaire. En effet, comme nous allions avoir un algorithme qui nous fournit la solution exacte par la suite, avoir cette preuve d'efficacité n'était pas fondamentalement utile.

Un nouvel objectif s'est rapidement présenté à nous. Encore une fois lié à l'évaluation de la qualité de l'algorithme glouton, cet objectif était celui de la génération d'exemples. En effet, il était nécessaire de fabriquer différents exemples de taille variable afin de tester notre algorithme, de relever sa complexité, de comparer les heuristiques et de pouvoir plus tard comparer les nouveaux algorithmes.

2

GÉNÉRATION D'EXEMPLES

2.1 Notre démarche

2.1.1 • Intérêt de la génération d'exemples

La première approche de l'étude du problème présenté par BNP Paribas a été de réfléchir à un algorithme "naïf", qui résoudrait le problème simplement, à la manière d'un humain tâtonnant pour trouver la réponse. Ce premier algorithme décrit plus tôt avait évidemment une complexité bien trop élevée par rapport à d'autres algorithmes sur lesquels nous nous sommes penchés plus tard. Mais il restait toutefois intéressant de mesurer le temps réel d'exécution de cet algorithme.

Notre tuteur M. Mikaël Imberty nous avait fourni un jeu d'exemples de monnaies et de corrélations entre monnaies. Disposant alors seulement de cet exemple, il nous paraissait logique d'essayer d'y appliquer notre algorithme naïf. Mais du fait de sa complexité trop importante, l'exécution de l'algorithme n'a pas terminé sur cet exemple initial.

Face à cette difficulté, nous nous sommes fixé un nouvel objectif : il s'agissait pour nous de chercher à créer des exemples plus petits, d'abord pour vérifier que notre code fonctionnait correctement, et ensuite pour évaluer jusqu'à quelle taille il était fonctionnel.

2.1.2 • Algorithme de génération d'exemples

La première idée a donc été d'extraire, à partir du tableau de corrélations original fourni par BNP Paribas (voir annexe 5.8), une version plus petite afin de garder des exemples plutôt réalistes. Pour cela, on sélectionne d'abord n monnaies parmi les N disponibles (avec $n < N$), ce qui revient à sélectionner n lignes du tableau de taille N. Ensuite, pour chacune de ces lignes correspondant à une monnaie retenue, on regarde les autres monnaies qui lui sont corrélées, et on les retire de la ligne si elles ne font pas partie de l'ensemble retenu. En faisant la sélection des monnaies aléatoirement, on s'assure de générer au hasard un sous-tableau de corrélations parmi les $\binom{n}{N}$ possibles.

Une fois que nous savons avec quelles monnaies nous travaillerons, il reste à déterminer les montants des positions dans chacune de ces monnaies. Nous avons décidé de restreindre nos exemples à des situations pour lesquelles le RWA minimal après compensation est nul. Ce choix était motivé par le fait qu'avec de tels exemples, on sait si l'algorithme renvoie le résultat optimal ou non. En effet, un résultat donnant un RWA nul est nécessairement le meilleur résultat possible, puisqu'il est impossible d'obtenir un RWA négatif en respectant les règles de compensation imposées. Ainsi, on possède une

Table 1: Exemple d'un tableau de corrélations de monnaies

Monnaie de référence	Monnaies qui lui sont corrélées			
EUR	USD	SGD	...	AUD
AUD	SGD	CAD	TST	
...	...			
CAD	AUD	TST		

mesure exacte de l'efficacité de notre algorithme. Si, appliqué à un exemple, il trouve une compensation de RWA nul, alors il a déterminé une solution exacte du problème.

Pour générer ces exemples à RWA compensé nul, nous avons choisi de procéder dans le sens inverse de résolution du problème. Tout d'abord, on initialise les montants des positions de toutes les monnaies à 0. On sélectionne ensuite aléatoirement une paire de monnaies corrélées, puis on génère aléatoirement une valeur que l'on ajoute à l'une et que l'on retranche à l'autre (voir Figure 2). Cette méthode de génération d'exemples devait[1] garantir l'existence d'une solution donnant un RWA nul en sortie de l'algorithme : puisque l'état initial avait bien un résultat nul, en effectuant les opérations dans le sens inverse, on doit retrouver le cas d'un RWA nul.

Par exemple, en reprenant le tableau de monnaies corrélées ci-dessus, on sélectionne aléatoirement la paire $(AUD; CAD)$. On génère ensuite un nombre x que l'on va ajouter au compte AUD et soustraire au compte CAD.

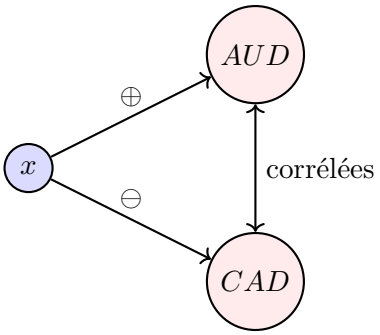


Figure 2: Génération d'un exemple

En répétant cette opération un nombre suffisant de fois, on obtient à la fin un exemple conforme à la situation réelle, puisque l'on a simplement opéré dans le sens inverse du sens de résolution. On peut donc comparer le résultat de l'algorithme avec une solution que l'on sait exacte.

[1] Nous nous sommes rendu compte seulement bien plus tard d'une erreur dans notre algorithme, voir 2.4.3.

2.2 Performance de l'algorithme naïf

Comme attendu, les performances de l'algorithme naïf sont extrêmement décevantes, et n'aboutissent que très rarement à une solution minimale globale. Cependant, au-delà des performances décevantes, le vrai souci est le temps d'exécution de cet algorithme. En effet, dès que la taille des exemples dépasse la dizaine de monnaies, l'algorithme a besoin de plusieurs minutes pour aboutir.

Pour visualiser la rapidité de l'algorithme naïf, nous avons moyenné son temps d'exécution sur plusieurs exemples pour chaque taille (jusqu'à ce que le temps d'exécution ne soit plus acceptable). Les résultats obtenus sont présentés en Figure 3 :

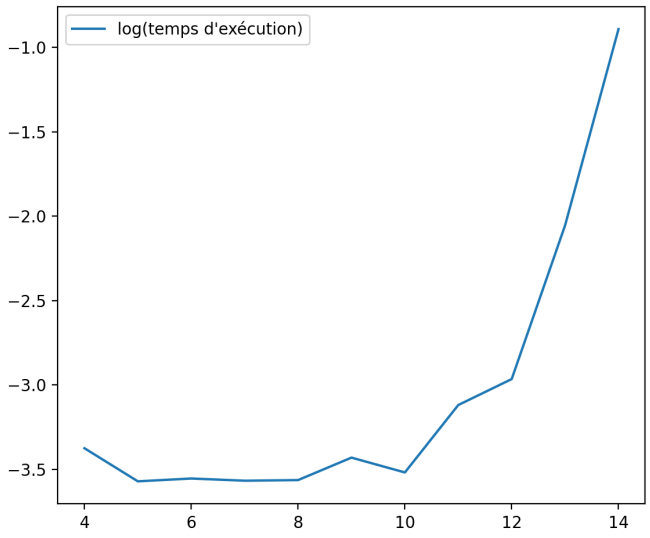


Figure 3: Logarithme du temps d'exécution en fonction de la taille des exemples

On voit sur ce graphique que le temps d'exécution n'est absolument pas raisonnable par rapport à la taille des exemples testés, surtout lorsqu'on connaît la taille d'un exemple réaliste (une trentaine de monnaies). De plus, par nature, cet algorithme naïf se restreint à un sous-ensemble de solutions dans lesquelles les compensations entre monnaies corrélées sont maximales, et seul l'ordre de sélection des monnaies varie. Ainsi, non seulement ce premier algorithme est particulièrement lent, mais en plus, le résultat obtenu n'est généralement pas optimal.

2.3 Exemples de taille n quelconque

En avançant dans l'élaboration de notre algorithme glouton, nous sommes vite arrivés à des vitesses de résolution bien plus élevées, qui parvenaient à résoudre l'exemple initial quasi-instantanément. Il était alors nécessaire d'élaborer des exemples de taille n quelconque, en se basant sur la même méthode de construction qu'auparavant.

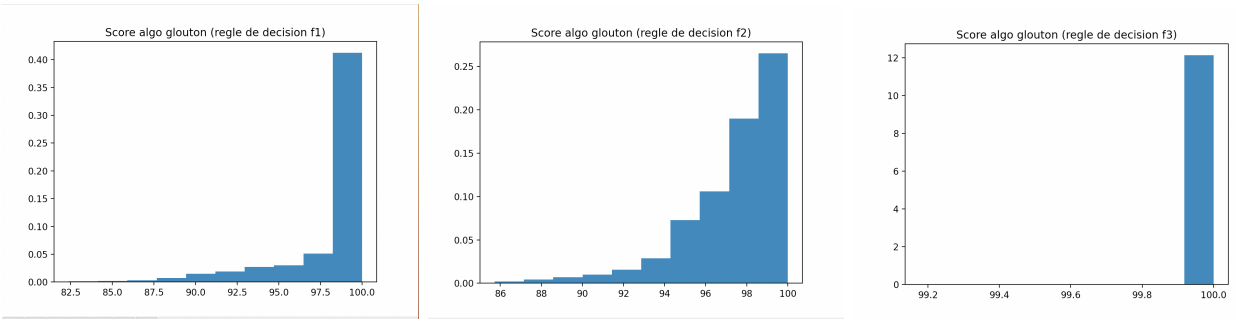
Pour cela, on génère simplement un tableau de corrélation *tab_corr* de dimension $len(tab) = n \times n$, qui renseigne dans la case (i, j) si la monnaie d'indice i est corrélée avec celle d'indice j. On génère ensuite la valeur des positions de la même manière qu'expliqué plus tôt.

2.4 L'algorithme glouton

2.4.1 • Efficacité de l'algorithme glouton

Pour mesurer l'efficacité de compensation de l'algorithme glouton sur $N = 2000$ problèmes à résoudre, nous avons tracé le score de chaque heuristique dans un graphe où l'abscisse représente le pourcentage du bilan qui a été compensé (l'objectif étant de le réduire à zéro et atteindre ainsi un score de 100%)

On obtient alors les résultats suivants :

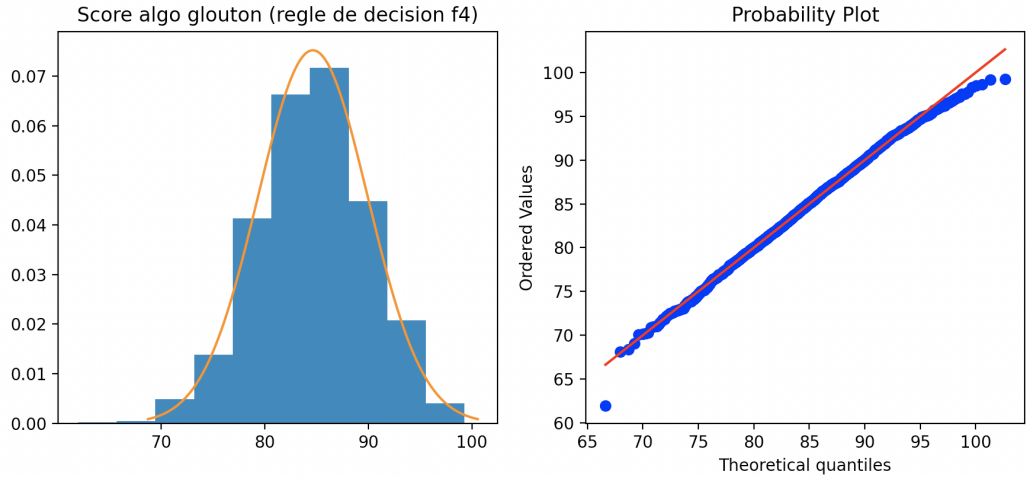


On voit ainsi que c'est la troisième règle de décision, celle qui choisit le couple composé des monnaies ayant le plus de corrélations, qui performe le mieux et qui ne trouve pas un RWA compensé nul dans seulement 2 des 2000 problèmes qu'elle traite[2].

[2]NB : cette analyse a été conduite avec la première version de notre générateur d'exemples.

2.4.2 • Utilisation de l'algorithme glouton pour quantifier la qualité des exemples

En plus de présenter une approche à la résolution de notre problème, l'algorithme glouton a trouvé une seconde utilité dans notre projet car il a permis de quantifier la qualité des problèmes produits par notre générateur d'exemples. En effet, en remplaçant l'heuristique habituelle par un générateur de nombres aléatoires, l'algorithme glouton produit alors une solution où les compensations sont effectuées au hasard (tout en s'assurant qu'elles soient légales). La performance d'une telle approche permet alors de vérifier la non-trivialité des exemples que nous produisons :



Comme on peut s'y attendre, les résultats sont distribués suivant une loi gaussienne (de paramètres $\mu \approx 85\%$ et $\sigma \approx 5$). Si la moyenne peut paraître anormalement haute, cela s'explique toutefois par le haut taux de corrélation entre les monnaies qui implique qu'un choix quelconque a peu de risque d'être vraiment mauvais.

2.4.3 • Correction de la méthode de création d'exemples

Plusieurs mois après cette première étude, après avoir implémenté les algorithmes de résolution exacte présentés dans la section suivante, nous nous sommes rendus compte d'une erreur dans notre génération d'exemples. En effet, alors que nous étions toujours censés trouver un RWA compensé nul (la résolution étant exacte et nos exemples construits dans cet objectif), nous avions quelques fois des résultats non nuls. Nous avons fini par comprendre qu'il s'agissait d'un problème dans la création des exemples, au cours de laquelle nous avions oublié un détail.

Pour bien comprendre le problème, prenons le cas précis de trois monnaies EUR, USD et CAD, et supposons que nous disposons des couples de corrélation suivants : (EUR, USD) et (USD, CAD) (on rappelle que la relation de corrélation n'est pas transitive, EUR et CAS ne sont donc pas corrélées).

- Initialement, on a $EUR = 0$, $USD = 0$ et $CAD = 0$.

- En suivant la méthode de génération d'exemple précédente, on se donne un nombre $x \geq 0$, que l'on distribue au couple (EUR, USD). On a donc maintenant $EUR = x$, $USD = -x$ et $CAD = 0$.
- De même, on se donne un nombre $y \geq 0$ que l'on distribue au couple (USD, CAD). On a alors $EUR = x$, $USD = y - x$ et $CAD = -y$ (voir Figure 4).

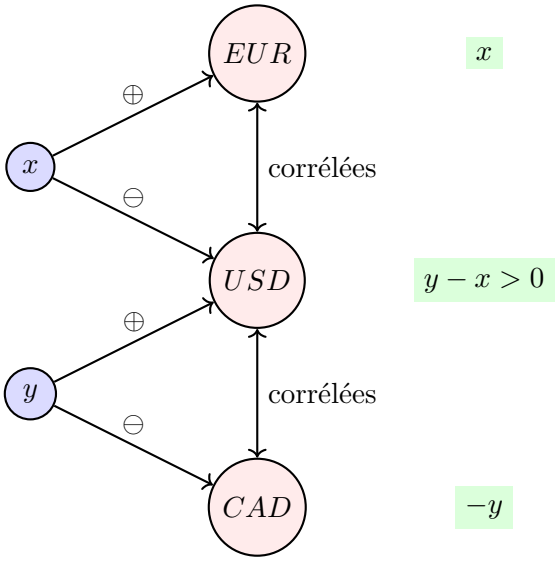


Figure 4: Création d'un exemple à 3 monnaies

Maintenant, si l'on essaie de résoudre cet exemple, on doit se poser la question du signe du montant en $USD = y - x$. En effet, les règles de compensation interdisent les compensations entre deux positions longues (positions positives sur une monnaie) ou entre deux positions courtes (positions négatives sur une monnaie). Ainsi, si $y > x$ on a $y - x > 0$, ce qui fait de USD une position longue qui ne pourra être compensée qu'avec $CAD = -y$ (position courte car négative). Si on compense, on ne pourra alors compenser que $y - x$, ce qui donnera la situation finale $EUR = x$, $USD = 0$ et $CAD = -x$ (solution exacte car aucune autre compensation ne peut être faite). Dans ce cas le RWA final est $RWA = |x| \neq 0$. On raisonne de manière analogue si $y < x$.

Le problème vient du fait qu'au cours de la création de l'exemple, nous n'avions pas tenu compte du fait qu'une monnaie est en position longue ou en position courte. Nous avons alors réglé ce problème en choisissant aléatoirement à l'état initial quelles monnaies seraient des positions longues, et quelles monnaies seraient des positions courtes. On s'assure ainsi que notre création d'exemple est bien conforme aux règles de compensations imposées.

3

RÉSOLUTION EXACTE PAR PROGRAMMATION LINÉAIRE

3.1 Formalisation mathématique

Après avoir commencé l'étude de nos premiers algorithmes et suite à notre première réunion avec notre coordinateur de PSC, nous nous sommes concentrés sur l'objectif de formalisation mathématique du problème qu'il nous avait fixé. Jusque là, nous avions étudié le sujet seulement de manière pratique, en termes algorithmiques. Nous n'avions pas pris le recul nécessaire pour modéliser le problème en dehors de son cadre financier, afin d'isoler le problème mathématique sous-jacent.

Par cette formalisation, en plus de consolider notre compréhension du sujet, nous avions bon espoir de retomber sur un problème classique d'optimisation, pour lequel il existe des outils de résolution performants. La formalisation telle que nous l'avons formulée est présentée ci-dessous.

3.1.1 • Notations

Notons n le nombre de positions longues et m le nombre de positions courtes dans le bilan considéré.

Soient $\mathcal{L} = \{L_1, \ldots, L_n\}$ l'ensemble des monnaies en positions longues, et $\mathcal{C} = \{C_1, \ldots, C_m\}$ l'ensemble des monnaies en positions courtes. On définit ainsi deux classes $\mathcal{L}$ et $\mathcal{C}$ formant une partition de l'ensemble des monnaies sur lesquelles la banque a des positions.

Notons l_i (resp. c_i) la position attribuée à la monnaie L_i (resp. C_i). Le bilan b de la banque vaut alors :

$$b = \sum_{i=1}^{n} l_i = \sum_{i=1}^{m} c_i$$

On note $\sim$ la relation de corrélation : si les monnaies L_i et C_j sont corrélées, on note $L_i \sim C_j$.

3.1.2 • Représentation sous forme de graphe

Nous pouvons alors visualiser le problème sous la forme d'un graphe biparti orienté, dont les deux classes sont $\mathcal{L}$ et $\mathcal{C}$, et dans lequel une arête indique une corrélation. Formellement, l'ensemble des arêtes E est défini par : $E = \{(L_i, C_j) \ : L_i \sim C_j\}$.

Sur chaque arête présente dans le graphe, on place une inconnue représentant le montant de la compensation effectuée entre les deux sommets considérés.

Exemple :

Dans le cas où $\mathcal{L} = \{\text{EUR, AUD}\}$ et $\mathcal{C} = \{\text{USD, SGD, CAD}\}$, avec les corrélations données par le tableau en annexe (Figure 11), nous obtenons le graphe biparti suivant :

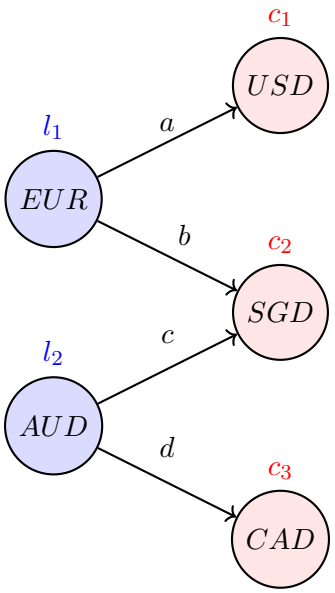


Figure 5: Exemple simple à cinq monnaies

Pour répondre au problème, il faut alors maximiser le montant compensé, à savoir $a + b + c + d$, en respectant la positivité des compensations et plusieurs contraintes logiques (une par monnaie) :

$$\mathcal{L} : \begin{cases} a + b \leq l_1 \\ c + d \leq l_2 \end{cases} \quad \text{et} \quad \mathcal{C} : \begin{cases} a \leq c_1 \\ b + c \leq c_2 \\ d \leq c_3 \end{cases}$$

Ces contraintes peuvent être rassemblées sous la forme matricielle $\begin{cases} Au \leq v \\ u \geq 0 \end{cases}$, avec :

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad u = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, \quad v = \begin{pmatrix} l_1 \\ l_2 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

Cas général :

Notons u le vecteur colonne dont les composantes sont les inconnues portées par les arêtes du graphe[3]. Si $d = |E| \leq nm$ désigne le nombre d'arêtes du graphe, alors $u \in \mathbb{R}^d$. L'objectif est alors de maximiser la compensation totale, à savoir $c^T u$, où $c = (1, \dots, 1)^T$, sous des contraintes à préciser.

Comme dans l'exemple précédent, on écrira les contraintes sous la forme $\begin{cases} Au \leq v \\ u \geq 0 \end{cases}$, avec :

$$u = \begin{pmatrix} u_1 \\ \dots \\ u_d \end{pmatrix} \in \mathbb{R}^d, \quad v = \begin{pmatrix} l_1 \\ \dots \\ l_n \\ c_1 \\ \dots \\ c_m \end{pmatrix} \in \mathbb{R}^{n+m}$$

Il s'agit alors de définir correctement la matrice $A \in \mathcal{M}_{n+m,d}(\mathbb{R})$. Pour cela, on la construit itérativement en s'inspirant de l'exemple. Les n premières lignes correspondent aux sommets de $\mathcal{L}$, et les m suivantes à ceux de $\mathcal{C}$. Sur chaque ligne, on place un 1 à chaque colonne correspondant (dans u) à une corrélation faisant intervenir le sommet considéré, et des 0 partout ailleurs.

3.1.3 • Formulation sous forme de programme linéaire

Étant donnée la formalisation mathématique précédente, notre problème s'exprime naturellement comme un programme linéaire sous forme canonique, en conservant les mêmes notations :

$$\begin{cases} \max_u c^T u \\ Au \leq v \\ u \geq 0 \end{cases}$$

Un tel problème peut être résolu exactement et efficacement,

Il suffit donc d'implémenter un programme en mesure d'identifier les variables du programme aux arrêtes et de construire la matrice des contraintes. La résolution se fait alors à l'aide d'algorithmes génériques tels que l'algorithme du simplexe que l'on peut trouver dans les bibliothèques python habituelles. Nous proposons donc l'implémentation présentée en annexe 5.4.

[3] À position longue fixée, on sélectionne les inconnues par ordre croissant de position courte ; puis par ordre croissant de position longue. On a bien $u = (a, b, c, d)^T$ dans l'exemple précédent.

3.2 Précision de la modélisation : réseau de flot

Suivant les conseils de notre tuteur, nous avons précisé notre modélisation du problème afin de faire apparaître une structure de réseau de flot. La maximisation du flot dans un réseau est un cas particulier de programmation linéaire, pour lequel il existe des algorithmes spécifiques plus performants que celui du simplexe. À terme, nous avions donc pour objectif d'obtenir une résolution toujours exacte, mais plus rapide qu'avec la modélisation précédente.

3.2.1 • Définition du réseau de flot

Selon la procédure habituelle lorsqu'on étudie un graphe biparti, nous avons ajouté une source s en amont de notre graphe (reliée aux éléments de $\mathcal{L}$), et un puits t en aval (relié aux éléments de $\mathcal{C}$).

On oriente alors les arêtes du graphe de la source vers $\mathcal{L}$, puis de $\mathcal{L}$ vers $\mathcal{C}$ (entre les monnaies corrélées) , et enfin de $\mathcal{C}$ vers le puits. Pour les arêtes partant de la source ou allant vers le puits, on définit la capacité comme la position (l_i ou c_i) de la monnaie sur l'autre sommet de l'arête. Pour les autres, on peut choisir comme capacité la position de l'une ou l'autre des deux monnaies (puisqu'il s'agit d'un réseau de flot)- nous avons choisi la position longue, soit l_i.

Pour l'exemple présenté en 3.1.2, nous obtenons alors le réseau de flot suivant :

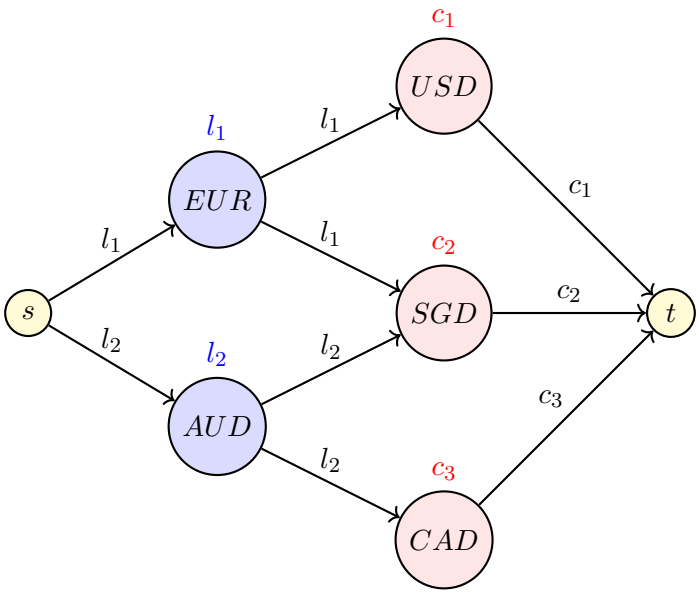


Figure 6: Définition sur réseau de flot sur l'exemple précédent

Cette modélisation permet ainsi de réduire notre problème à celui de la maximisation du flot dans le réseau défini ci-dessus. Pour l'implémentation, voir l'annexe 5.6.

3.2.2 • Pondération du RWA

En ce qui concerne la pondération du RWA selon les couples de monnaies corrélées, il nous semble qu'il n'y ait pas de manière simple d'en tenir compte dans une modélisation sous forme de réseau de flot.

En revanche, la première formulation sous forme de programme linéaire général permet de tenir compte de cette pondération. Il suffit de changer les coefficients de la matrice A de la manière suivante : on remplace le 1 correspondant au couple considéré par le coefficient de pondération associé à la corrélation.

3.3 Performances

Étant donné que les algorithmes de programmation linéaire et de maximisation du flot donnent des solutions exactes à notre problème, nous avons souhaité comparer leurs performances en temps de calcul. Suivant les méthodes de résolution employées par la bibliothèque *Scipy* et la configuration précise du graphe, les performances peuvent varier. Nous avons donc tracé leur complexité empirique. Celle-ci est polynomiale, comme on pouvait s'y attendre :

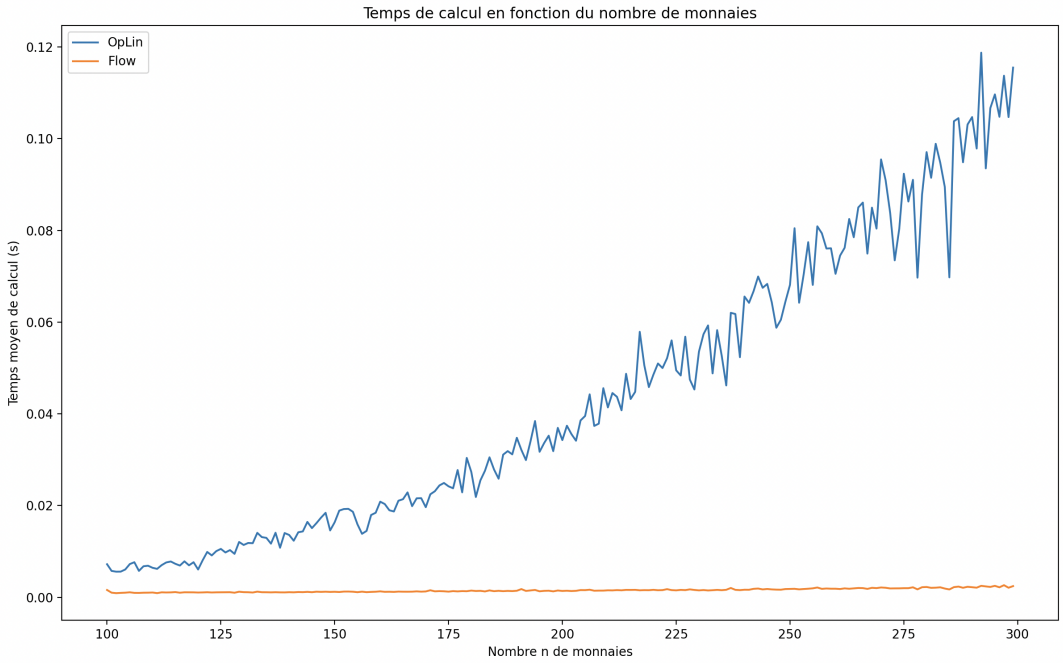


Figure 7: Représentation graphique de la complexité temporelle des algorithmes

Afin d'estimer l'ordre du terme dominant pour chacun des deux graphes de complexité, nous avons effectué une régression linéaire :

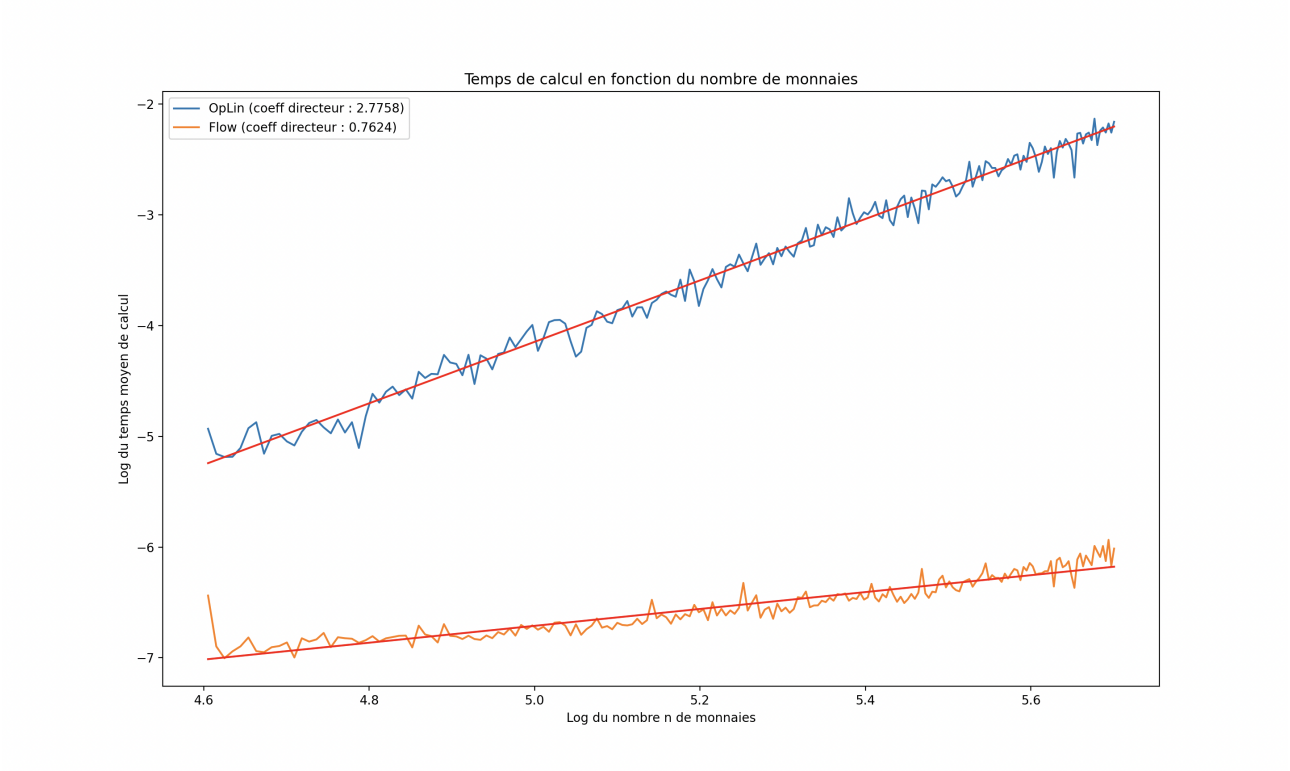


Figure 8: Régression linéaire sur les graphes de complexité temporelle

Cette analyse de la performance des deux algorithmes confirme que la méthode de maximisation du flot est une approche plus efficace.

3.4 Cas particulier : au plus une compensation par devise

L'algorithme de flot répondant exactement au problème posé, notre tuteur nous a proposé en guise de prolongement d'explorer des cas particuliers dans lesquels on restreint les compensations possibles.

Nous avons donc étudié le cas où l'on autorise au plus une seule compensation par devise. Tout comme le cas général, ce cas particulier peut être résolu par programmation linéaire.

Partons du graphe non orienté[4] suivant (Figure 9), dans lequel le montant indiqué au-dessus de chaque arête correspond à la compensation maximale autorisée :

[4]On note donc $(u, v) = (v, u)$ pour $u, v \in V$

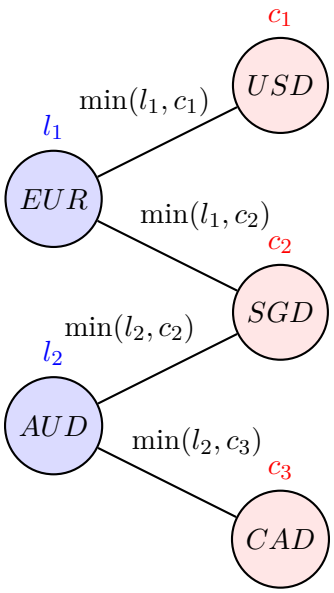


Figure 9: Exemple simple dans le cas d'au plus une compensation par monnaie

Résoudre le problème avec contrainte revient ainsi à choisir un ensemble d'arêtes (représentant les compensations effectuées) tel que :

- Chaque élément de $\mathcal{L}$ (position longue) soit relié au plus à un élément de $\mathcal{C}$ (positions courtes),
- Chaque élément de $\mathcal{C}$ (position courte) soit relié au plus à un élément de $\mathcal{L}$ (positions longues),
- La compensation soit maximale.

Mathématiquement, on obtient la formulation suivante : avec F l'ensemble d'arêtes que l'on sélectionne et $x_e = 1_{e \in F}$, $\forall e \in E$, on cherche à maximiser $\sum_{e \in E} c_e x_e$ sous la contrainte :

$$\forall\, v \in V, \quad \sum_{u \in N(v)} x_{(u,v)} \leq 1$$

avec $N(v) = \{u \in V,\ (u, v) \in E\}$ l'ensemble des voisins de v.

Il s'agit donc d'un problème de programmation linéaire en nombres entiers, dont nous n'avons pas eu le temps d'étudier la résolution.

4

OBJECTIFS ET MÉTHODOLOGIE

À travers ce Projet Scientifique Collectif, nous avons pu développer nos capacités à travailler efficacement en groupe, en analysant rapidement les objectifs à atteindre pour pouvoir avancer, en se répartissant les tâches de façon équilibrée, et en travaillant souvent à plusieurs afin de ne pas manquer des détails cruciaux.

Afin d'aborder plus en détails les différents objectifs et la méthodologie que nous avons adoptée, nous passerons en revue de manière chronologique les étapes clés de notre PSC.

4.1 Choix et découverte du sujet

4.1.1 • Juin : Recherche du sujet

Dans notre recherche de sujet du PSC, nous étions d'accord pour nous orienter vers un projet avec de l'informatique et des mathématiques appliquées. Nous nous étions intéressés à un thème de génération de musique grâce à des réseaux de neurones. N'ayant pas trouvé de tuteur pour ce sujet, nous avons regardé d'autres sujets de PSC.

4.1.2 • Septembre - Octobre : Découverte du sujet

Au début de l'année, nous sommes tombés sur une proposition de sujet sur X-Job Board. Ce thème alliait informatique et mathématiques appliquées, dans un contexte financier intéressant après les évènements de 2008. Nous avons donc opté pour ce sujet.

Pour les deux premiers mois de ce PSC, nous avons commencé par nous familiariser avec le projet. En effet, ce thème regroupe des notions de finance et d'informatique qu'il était important de maîtriser pour pouvoir aborder le problème de la meilleure façon. Nous n'avions comme base de travail qu'un premier document envoyé par BNP Paribas pour décrire l'offre de PSC. Nous nous sommes renseignés de notre côté avant de prévoir une première réunion avec notre tuteur Monsieur Mikael Imberty, qui nous a permis de lui poser nos questions sur le sujet.

Maintenant que le problème nous était plus familier, nous avons cherché un premier algorithme pour obtenir la meilleure compensation possible. L'algorithme dit "naïf" de la première partie était le choix logique.

La conception de cet algorithme fut assez complexe, notamment car il était difficile d'appréhender

l'ensemble des ordres possibles. Nous avons finalement réussi à concevoir une première version de l'algorithme, mais celle-ci ne répondait pas exactement à nos attentes car elle ne permettait pas de tester l'ensemble des ordres. Nous avons donc ensuite corrigé cet algorithme, jusqu'à aboutir à une version satisfaisante. Cependant nous avons rapidement réalisé que cet algorithme ne suffirait pas, tant en terme de puissance de calcul que de complexité temporelle. Notre objectif principal suivant était donc de trouver un type d'algorithme qui permettrait de résoudre efficacement ce problème, et qui supporterait un plus grand nombre de monnaies.

4.2 Traitement algorithmique et mathématique

4.2.1 • Novembre - Décembre : premiers algorithmes et génération d'exemples

Après une deuxième réunion avec M. Imberty, où nous lui avons présenté nos résultats sur l'algorithme naïf et exposé nos difficultés face aux problèmes que cet algorithme posait (il ne supportait que quelques monnaies), M. Imberty nous a conseillé de nous pencher sur le principe d'un algorithme glouton. Grâce à de la documentation qu'il nous avait fourni en début de projet sur les différents types d'algorithmes, nous avons bien opté pour ce principe glouton en deuxième algorithme.

Une des difficultés majeures de notre PSC s'est alors révélée à ce moment-là. Si nous voulions tester la validité, l'efficacité et la complexité de nos algorithmes, nous ne pouvions pas seulement avoir comme exemple de monnaies/corrélations le jeu que nous avait envoyé notre tuteur. Il était primordial d'avoir un générateur d'exemples nous permettant de connaître la valeur de compensation maximale des exemples, afin de pouvoir coder proprement nos algorithmes, et de les modifier si besoin.

Nous avons alors séparé le groupe en deux sous-groupes de travail. Un groupe s'occupait de coder l'algorithme glouton, pendant que l'autre se penchait sur la génération d'exemples.

Une fois l'algorithme glouton prêt (testé grâce au générateur), nous avons demandé la résolution obtenue par BNP Paribas de l'exemple qui nous avait été fourni. Nous avons donc pu comparer ce que l'algorithme de BNP Paribas réussissait à compenser par rapport à ce que notre algorithme glouton compensait. Le résultat était déjà plutôt satisfaisant !

4.2.2 • Janvier - Février : formalisation mathématique, programmation linéaire et réseau de flot

Début janvier a eu lieu notre première rencontre avec le coordinateur du PSC, Monsieur Gilles Schaeffer. Nous lui avons exposé nos algorithmes, nos difficultés rencontrées et nos objectifs futurs. L'aide de M. Schaeffer a permis une avancée importante dans notre projet. A la suite de cette réunion, nous avions comme objectif principal de formaliser mathématiquement le problème, réflexe que nous n'avions pas eu en première approche de ce sujet. Cette formalisation mathématique nous a permis de comprendre que le problème s'apparentait à celui d'un graphe biparti orienté.

Quelques jours après, nous montrions nos résultats de l'algorithme glouton à notre tuteur M. Imberty et nous lui expliquions les avancées faites sur la formalisation en problème linéaire. Ce-dernier nous a laissé assez libres dans la programmation de l'algorithme.

Notre dernière réunion avec M. Imberty nous a permis de lui présenter les résultats sur l'algorithme linéaire. Il nous a également indiqué de nous pencher sur l'algorithme de flot, qui était une autre bonne manière d'aborder le sujet, et d'étudier un cas particulier où seule une compensation par devise était autorisée (partie 4.4).

4.3 Mars : finalisation et mise en forme

Lors de ce dernier mois de travail, nous avons peaufiné les codes, pour leur donner plus de cohérence.

Nous avons également décidé de nous fixer un objectif supplémentaire pour la fin du PSC : celui de programmer une interface permettant d'afficher les résultats de notre travail sous Excel. En effet, ce projet a été initialement demandé par BNP Paribas auprès d'une startup pour diminuer leur RWA. Il semble alors naturel de fournir au "client" (ici BNP), un fichier en sortie qui soit exploitable par la banque.

Nous nous sommes également répartis l'écriture du rapport final du PSC, avec des relectures finales collectives.

CONCLUSION

L'objectif final de ce Projet Scientifique Collectif était de produire un outil qui permettrait d'optimiser les compensations entre monnaies corrélées et ainsi réduire au maximum les provisions immobilisées par une banque souhaitant se couvrir contre le risque de change.

Nous l'avons atteint en proposant plusieurs algorithmes, d'approches et d'efficacités différentes, et avons rendu l'outil final facilement utilisable pour une banque grâce à son format de sortie. Ce travail fut une progression continue, la meilleure réponse nous apparaissant au fur et à mesure que nous surmontions difficultés et échecs. Ce projet fut la situation idéale pour mener à bien une réflexion scientifique sans connaissances initiales sur le sujet.

Après un TIPE (Travail d'Initiative Personnelle Encadré) en classes préparatoires réalisé par groupe de deux, ce Projet Scientifique Collectif fut l'occasion de collaborer dans un groupe plus large, et ainsi de nous apporter un avant goût de nos futures méthodes de travail en entreprise. Une bonne communication, concision de nos idées, précision des résultats, et organisation dans la réflexion collective nous ont permis de ne pas perdre de temps.

Notre travail tout au long de cette année pour le Projet Scientifique Collectif aura été une expérience enrichissante. C'est une préparation à la vie en entreprise, où l'on devra allier travail personnel, avancement de projet en collectif, et présentation devant des supérieurs.

Enfin, nous remercions chaleureusement notre tuteur M. Mikaël Imberty, pour son aide précieuse pendant toute l'année, son accompagnement et sa finesse dans les conseils qu'il nous a données. Nous remercions également notre coordinateur, M. Gilles Schaeffer, pour son expertise et son aide sur la réalisation de ce projet.

BIBLIOGRAPHIE

[**1**] *European Banking Authority*, Implementing Technical Standards on closely correlated currencies

`https://www.eba.europa.eu/regulation-and-policy/market-risk/implementing-technical-standards-its-on-closely-correlated-currencies`

[**2**] *Grokking Algorithms*, Aditya Bhargava, *Manning Publications* :

- Chapitre 8 (Greedy Algorithms)

[**3**] *Algorithms in a Nutshell*, 2nd Edition, O'Reilly Media :

- Chapitre 6 (Graphs Algorithms)
- Chapitre 8 (Network Flow Algorithms)

[**4**] *Linear Programming*: The Simplex Method

`https://www.math.wsu.edu/students/odykhovychnyi/M201-04/Ch06_1-2_Simplex_Method.pdf`

[**5**] *Scipy Documentation* : Linear programming

`https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html`

[**6**] *Scipy Documentation* : Maximum flow algorithm

`https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.maximum_flow.html`

[**7**] *Xlsxwriter Documentation* : Writing to Excel files

`https://xlsxwriter.readthedocs.io/`

Annexes

5.1 La classe Algo

Cette classe dont hérite chaque algorithme prend en charge le chargement des données du problème et la mise en forme du résultat sous forme de tableau *Excel* grâce à la bibliothèque Python *xslxwriter* ([7]).

```
# manipulation de donnees
import numpy as np

# generateur d'exemples
from Exemples import creer_exemple, creer_exemple_simple

# gestion des fichiers excel
import xlsxwriter

"""
Classe de base servant de squelette aux algorithmes d'optimisation
    - Le constructeur gere la mise en forme des donnees commune aux algorithmes
    - La methode write_to_excel permet d'ecrire les resultats dans un fichier Excel
"""
class Algo:
    name = ""
    def __init__(self, tab, names):
        self.pos_long = {}
        self.pos_court = {}
        self.correlations = {}
        self.solution = {}
        self.values = np.zeros(len(names))
        self.RWA = -1
        self.initialsum = 0

        self.names = names

        for i in range(len(tab)):
            if tab[i][0] >= 0:
                self.pos_long[names[i]] = tab[i][0]
                self.values[i] = tab[i][0]
            else:
                self.pos_court[names[i]] = -tab[i][0]
                self.values[i] = -tab[i][0]

            self.correlations[names[i]] = []
            j = 1
            while j < len(tab[i]) and tab[i][j] != -1:
                self.correlations[names[i]].append(names[tab[i][j]])
                j += 1
```

```
        self.initialsum = np.sum(self.values)/2

    def write_to_excel(self, filename):
        """
        fonction qui presente la solution dans un fichier excel sous forme d'un tableau a
            double entree
        """
        workbook = xlsxwriter.Workbook(filename)
        worksheet = workbook.add_worksheet()

        # definitions des styles utilises pour la mise en forme du tableau
        # les cases vertes
        formatvert = workbook.add_format()
        formatvert.set_fg_color('green')
        formatvert.set_bold()
        formatvert.set_font_color('white')
        # les cases grises
        formatgris = workbook.add_format()
        formatgris.set_fg_color('gray')

        # mise en forme initiale (nom des lignes et colonnes)
        # pour les positions longues
        worksheet.write(1, 3, "AVANT")
        worksheet.write(2, 3, "APRES")
        # pour les positions courtes
        worksheet.write(3, 1, "AVANT")
        worksheet.write(3, 2, "APRES")

        worksheet.write(3, 3, "RWA")

        # ecrire les noms des monnaies, de leur montant initial, leur montant final et leur
            apport au RWA
        names_long = list(self.pos_long.keys())
        names_court = list(self.pos_court.keys())
        i = 0
        for name_long in names_long:
            worksheet.write(4+i, 0, name_long, formatvert)                    # NOM
            worksheet.write(4+i, 1, self.pos_long[name_long])                 # AVANT
            worksheet.write(4+i, 2, self.values[self.names.index(name_long)]) # APRES
            worksheet.write(4+i, 3, self.values[self.names.index(name_long)]/2) # RWA
            i += 1

        worksheet.write(4+i+1, 0, "TOTAL", formatvert)                        # TOTAUX
        worksheet.write(4+i+1, 1, np.sum(list(self.pos_long.values())))       # AVANT
        apres_long = np.sum([self.values[self.names.index(name_long)] for name_long in
            names_long])
        worksheet.write(4+i+1, 2, apres_long)                                 # APRES
        worksheet.write(4+i+1, 3, apres_long/2)                               # RWA

        i = 0
        for name_court in names_court:
            worksheet.write(0, 4+i, name_court, formatvert)                   # NOM
```

```
        worksheet.write(1, 4+i, self.pos_court[name_court])              # AVANT
        worksheet.write(2, 4+i, self.values[self.names.index(name_court)]) # APRES
        worksheet.write(3, 4+i, self.values[self.names.index(name_court)]/2) # RWA
        i += 1

    worksheet.write(0, 4+i+1, "TOTAL", formatvert)                       # TOTAUX
    worksheet.write(1, 4+i+1, np.sum(list(self.pos_court.values())))     # AVANT
    apres_court = np.sum([self.values[self.names.index(name_court)] for name_court in
        names_court])
    worksheet.write(2, 4+i+1, apres_court)                               # APRES
    worksheet.write(3, 4+i+1, apres_court/2)                             # RWA

    # ajout des montants des compensations enregistres dans self.solution
    for i in range(len(names_long)):
        for j in range(len(names_court)):
            if (names_long[i], names_court[j]) in self.solution.keys():
                worksheet.write(4+i, 4+j, self.solution[(names_long[i], names_court[j])])
            else:
                worksheet.write(4+i, 4+j, '', formatgris)

    # fermer le fichier
    workbook.close()
```

5.2 Algorithme naïf

```
class AlgoNaif(Algo):
    def __init__(self, tab, names):
        super().__init__(tab, names)
        self.name ="Naif"

# Renvoie une liste de couples correles a partir d'un dictionnaire de correlations
    def corrList(self,corres):
        L = []
        for c in corres :
            if str(c) in self.pos_long.keys():
                long = c
                for short in corres[c]:
                    L.append([long,short])
        return L

# Effectue les compensations a partir des donnees des positions (longues et courtes) et de
    la liste L donnant l'ordre des couples a compenser.
    def compensationList(self, poslong,poscourt,L):

        posLong = copy.deepcopy(poslong)
        posShort = copy.deepcopy(poscourt)

        for x in L :
            long,short = x
            diff = posLong[long] - posShort[short]

            if diff >= 0:
                posLong[long] = diff
                posShort[short] = 0

            else :
                posShort[short] = -diff
                posLong[long] = 0


        return posLong, posShort


# Compare les compensations obtenues a partir des differents ordres de couples a compenser
    def comparecompensation(self):

        L= self.corrList(self.correlations)

        permutations = list((itertools.permutations(L)))

        meilleureCompensation = sum([self.pos_long[c] for c in self.pos_long])
        meilleurePermutation = L
```

```
        for perm in permutations :
            comp = sum(self.compensationList(self.pos_long,self.pos_court,perm)[0].values())
# Si la compensation est nulle alors la solution est optimale
            if comp == 0:
                return perm, comp

            if comp < meilleureCompensation :
                meilleureCompensation = comp
                meilleurePermutation = perm


        return meilleurePermutation, meilleureCompensation

# Met a jour les valeurs une fois le meilleur ordre trouve.
    def optimize(self):
        Perm, Comp = self.comparecompensation()
        print(Perm)
        print(Comp)
        for x in Perm :
            long, short = x
            diff = self.values[self.names.index(long)] - self.values[self.names.index(short)]

            if diff >= 0:
                self.solution[(long,short)]= self.values[self.names.index(short)]
                self.values[self.names.index(long)] = diff
                self.values[self.names.index(short)] = 0

            else :
                self.solution[(long,short)]= self.values[self.names.index(long)]
                self.values[self.names.index(short)] = -diff
                self.values[self.names.index(long)] = 0
```

5.3 Algorithme Glouton

```
"""
Algrithme implementant l'optimisation par algorithme glouton
"""
class Greedy(Algo):
    def __init__(self, tab, names):
        super().__init__(tab, names)
        self.name = "Greedy"
        self.pos_long_copy = self.pos_long.copy()
        self.pos_court_copy = self.pos_court.copy()
        self.correlations_copy = self.correlations.copy()

    # realise la compensation et supprime les monnaies de valeur nulle
    def compAndUpdate(self, monnaies):
        val = self.pos_long_copy[monnaies[0]] - self.pos_court_copy[monnaies[1]]
        if val > 0:
            # necessaire pour l'output excel
            self.values[self.names.index(monnaies[0])] -= self.pos_court_copy[monnaies[1]]
            self.values[self.names.index(monnaies[1])] = 0
            self.solution[monnaies] = self.pos_court_copy[monnaies[1]]

            self.pos_long_copy[monnaies[0]] = val
            del self.pos_court_copy[monnaies[1]]
            del self.correlations_copy[monnaies[1]]
        elif val < 0:
            # necessaire pour l'output excel
            self.values[self.names.index(monnaies[0])] = 0
            self.values[self.names.index(monnaies[1])] -= self.pos_long_copy[monnaies[0]]
            self.solution[monnaies] = self.pos_long_copy[monnaies[0]]

            self.pos_court_copy[monnaies[1]] = -val
            del self.pos_long_copy[monnaies[0]]
            del self.correlations_copy[monnaies[0]]
        else:
            # necessaire pour l'output excel
            self.values[self.names.index(monnaies[0])] = 0
            self.values[self.names.index(monnaies[1])] = 0
            self.solution[monnaies] = self.pos_long_copy[monnaies[0]]

            del self.pos_long_copy[monnaies[0]]
            del self.correlations_copy[monnaies[0]]
            del self.pos_court_copy[monnaies[1]]
            del self.correlations_copy[monnaies[1]]

        return abs(val)

    # verifie s'il reste des monnaies compensables (de montants non nuls et correlees a des
        monnaies de montant non nul)
    def shouldContinue(self):
```

```
    for c in self.pos_court_copy:
        if len(set(self.correlations_copy[c]) and set(self.correlations_copy.keys()) and
            set(self.pos_long_copy.keys())) != 0:
            return True
    for l in self.pos_long_copy:
        if len(set(self.correlations_copy[l]) and set(self.correlations_copy.keys()) and
            set(self.pos_court_copy.keys())) != 0:
            return True
    return False

def f(self, couple):
    return -(len(self.correlations_copy[couple[0]]) +
        len(self.correlations_copy[couple[1]]))

# coeur de l'algo : trouve le couple qui maximise la regle de decision f, effectue la
    compensation et recommence jusqu'a ce qu'il ne reste plus de couples valides
def optimize(self):
    while(self.shouldContinue()):
        best=()
        for c in self.pos_court_copy:
            for l in self.pos_long_copy:
                if l in self.correlations_copy[c]:
                    best = (l, c)
                    break
        for c in self.pos_court_copy:
            for l in self.pos_long_copy:
                if c in self.correlations_copy[l]:
                    if self.f((l, c)) > self.f(best):
                        best = (l, c)
        if best == ():
            break
        else :
            self.compAndUpdate(best)
```

5.4 Programmation linéaire

Algorithme implémentant la résolution grâce à un programme linéaire. Nous utilisons l'algorithme de résolution des programmes linéaires de la bibliothèque *Scipy* ([5]).

```
from scipy.optimize import linprog
"""
Algorithme implementant l'optimisation par programmation lineaire
"""
class OpLin(Algo):
    def __init__(self, tab, names):
        super().__init__(tab, names)
        self.name = "OpLin"

    def optimize(self):
        # determination des bonnes dimensions pour les contraintes
        nb_contraintes = len(self.names)
        nb_variables = 0
        edges = []
        for name_long in self.pos_long.keys():
            for name_court in self.pos_court.keys():
                if name_court in self.correlations[name_long]:
                    edges.append((name_long, name_court))
                    nb_variables += 1

        # construction de la matrice intervenant les inegalites de contrainte Ax <= b avec
            b = valeurs de depart
        A = np.zeros((nb_contraintes, nb_variables))
        i = 0
        for e in edges:
            A[self.names.index(e[0])][i] = 1
            A[self.names.index(e[1])][i] = 1
            i += 1
        # construction du vecteur intervenant dans la fonction objectif (c)
        c = np.full(nb_variables, -1)

        # resolution du probleme
        self.result = linprog(c=c, A_ub=A, b_ub=self.values)

        # mise a jour des positions
        i = 0
        for e in edges:
            self.solution[e] = self.result.x[i]
            self.values[self.names.index(e[0])] -= self.result.x[i]
            self.values[self.names.index(e[1])] -= self.result.x[i]
            i += 1

        # calcul du RWA
        self.RWA = np.sum(self.values)/2
```

5.5 Algorithme de génération d'exemple

```
    import numpy as np

"""
Script permettant de creer des exemples de tableaux de donnees de taille arbitraire pour
    les tests des algorithmes d'optimisation.
"""

"""
Creation d'exemples dans le cas gnral
"""
def creer_exemple(n):
    # creation du tableau de noms (nous utilisons des noms fictifs dans ce genre de format
        : "0", "1", "2", ...)
    names = []
    for i in range(n):
        names.append(str(i))

    # creation du tableau de valeurs (format d'une ligne : [valeur, indice1, indice2, ...]
        ou indice1, indice2, ... sont les indices des noms des positions correlees a la
        position i)
    tab = np.zeros((n, n))

    # creation d'une matrice de correlation
    corr_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(i):
            if np.random.rand() < 0.54:
                corr_matrix[i, j] = 1
                corr_matrix[j, i] = 1

    # remplissage des correlations dans le tableau
    for i in range(n):
        corr_line = []
        for j in range(n):
            if corr_matrix[i, j] == 1:
                corr_line.append(j)
        corr_line = np.array(corr_line)
        tab[i, 1:] = np.concatenate((corr_line, np.full(n-1-len(corr_line), -1)))

    # incrmentation des valeurs des positions
    # on choisit alatoirement un couple de valeurs corrles auquel on ajoute/retranche une
        valeur alatoire
    for _ in range(1000):               # Le choix de 1000 est arbitraire et peut tre
        modifi suivant l'ordre de grandeur des valeurs des positions souhaites
        val = np.random.randint(1, 10000) # idem pour la borne suprieure de l'intervalle
            des valeurs
        i = np.random.randint(0, n)
        j = np.random.randint(0, n)
```

```
            if corr_matrix[i, j] == 1:
                tab[i, 0] += val
                tab[j, 0] -= val

        tab = tab.astype(int)
        return tab, names

"""
 Cration  d'exemples assurant une solution optimale nulle
"""
def creer_exemple_simple(n):
    #  cration  du tableau de noms (nous utilisons des noms fictifs dans ce genre de format
        : "0", "1", "2", ...)
    names = []
    for i in range(n):
        names.append(str(i))

    #  cration  du tableau de valeurs (format d'une ligne : [valeur, indice1, indice2, ...]
        o indice1, indice2, ... sont les indices des noms des positions corrles  la
        position i)
    tab = np.zeros((n, n))

    # creation d'une matrice de correlation
    corr_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(i):
            if np.random.rand() < 0.54:
                corr_matrix[i, j] = 1
                corr_matrix[j, i] = 1

    # remplissage des correlations dans le tableau
    for i in range(n):
        corr_line = []
        for j in range(n):
            if corr_matrix[i, j] == 1:
                corr_line.append(j)
        corr_line = np.array(corr_line)
        tab[i, 1:] = np.concatenate((corr_line, np.full(n-1-len(corr_line), -1)))

    # choix du nombre de positions longues
    nb_long = np.random.randint(1, n-1)
    pos_long = np.random.choice(n, nb_long, replace=False)       # on choisit alatoirement
        les positions longues (replace=False permet d'viter les doublons)
    pos_court = np.array([i for i in range(n) if i not in pos_long]) # les autres sont des
        positions courtes

    # incrmentation des valeurs des positions
    # on choisit alatoirement un couple de valeurs corrles auquel on ajoute/retranche une
        valeur alatoire suivant le type de position
    for _ in range(1000):               # Le choix de 1000 est arbitraire et peut tre
        modifi suivant l'ordre de grandeur des valeurs des positions souhaites
```

```
        val = np.random.randint(1, 10000) # idem pour la borne suprieure de l'intervalle
            des valeurs
        i = np.random.choice(pos_long)
        j = np.random.choice(pos_court)
        if corr_matrix[i, j] == 1:
            tab[i, 0] += val
            tab[j, 0] -= val

    tab = tab.astype(int)
    return tab, names
```

5.6 Maximisation du flux

Algorithme implémentant la résolution grâce à une maximisation de flot. Nous utilisons l'algorithme de maximisation de flot de la bibliothèque *Scipy* ([6]).

```
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import maximum_flow

"""
Algrithme implementant l'optimisation par flux maximal
"""
class Flow(Algo):
    def __init__(self, tab, names):
        super().__init__(tab, names)
        self.name = "Flow"

    def optimize(self):
        # ajout de la source et du puits
        self.names.append("source")
        self.names.append("puits")

        # construction des arretes du graphe
        edges = []
        for name_long in self.pos_long.keys():
            for name_court in self.pos_court.keys():
                if name_court in self.correlations[name_long]:
                    edges.append((name_long, name_court))
        for name_long in self.pos_long.keys():
            edges.append(("source", name_long))

        for name_court in self.pos_court.keys():
            edges.append((name_court, "puits"))

        # construction de la matrice d'adjacence
        matrix = np.zeros((len(self.names), len(self.names)))
        for e in edges:
            i = self.names.index(e[0])
            j = self.names.index(e[1])
            val = 0
            if e[0] == "source":
                val = self.pos_long[e[1]]
            elif e[1] == "puits":
                val = self.pos_court[e[0]]
            else:
                val = self.pos_long[e[0]]

            matrix[i][j] = abs(val)

        # resolution du probleme
```

```
        source_index = len(self.names) - 2
        puits_index = len(self.names) - 1

        self.result = maximum_flow(csr_matrix(matrix.astype(int)), source_index,
            puits_index)
        flow_array = self.result.flow.toarray()

        for name_long in self.pos_long.keys():
            for name_court in self.pos_court.keys():
                if name_court in self.correlations[name_long]:
                    i = self.names.index(name_long)
                    j = self.names.index(name_court)
                    self.solution[(name_long, name_court)] = flow_array[i][j]
                    self.values[self.names.index(name_long)] -= flow_array[i][j]
                    self.values[self.names.index(name_court)] -= flow_array[i][j]
```

5.7 Exemple d'un résultat au format Excel

				1	3	4	6	11	18	19		TOTAL
			AVANT	374792	487153	369435	394038	302641	376382	354153		2658594
			APRES	0	0	0	0	0	0	0		0
	AVANT	APRES	RWA	0	0	0	0	0	0	0		0
0	128688	0	0		128688	0						
2	77759	0	0			77759		0				
5	33790	0	0			33790						
7	124366	0	0	124366			0			0		
8	179868	0	0	179868	0		0					
9	200283	0	0		200283		0		0	0		
10	342263	0	0	70558	158182		113523	0	0	0		
12	264849	0	0	0	0	257886			6963	0		
13	158484	0	0	0			158484	0				
14	357537	0	0	0	0	0		302641	54896	0		
15	259092	0	0	0	0	0	122031			137061		
16	227349	0	0	0		0	0	0	227349			
17	304266	0	0		0		0	0	87174	217092		
TOTAL	2658594	0	0									

Figure 10: Exemple d'un résultat au format Excel

5.8 Tableau de corrélations

AED	BAM	CHF	CNY	DKK	HKD	HRK	ILS	INR	LBP	MAD	MKD	MOP	PEN	PHP	SGD	THB	TWD	USD	EUR												
ALL	BAM	BGN	CHF	CZK	DKK	HKD	HRK	HUF	ILS	MAD	MOP	PLN	RON	SGD	THB	TWD	EUR														
AUD	CAD	DKK	SGD																												
BAM	AED	ALL	BGN	CHF	CNY	CZK	DKK	HKD	HRK	HUF	ILS	LBP	MAD	MKD	MOP	NOK	PEN	PLN	RON	RSD	SEK	SGD	THB	TWD	USD	EUR					
BGN	ALL	BAM	CHF	CNY	CZK	HUF	ILS	MAD	MKD	MOP	NOK	PEN	PLN	RON	RSD	SEK	SGD	THB	TWD												
CAD	AUD	ILS	PHP	SGD	THB																										
CHF	AED	ALL	BAM	BGN	CNY	CZK	DKK	HKD	HRK	HUF	ILS	LBP	MAD	MKD	MOP	NOK	RON	RSD	SGD	THB	TWD	USD	EUR								
CNY	AED	BAM	BGN	CHF	DKK	HKD	HRK	ILS	INR	LBP	MAD	MKD	MOP	PEN	PHP	RON	SGD	THB	TWD	USD	EUR										
CZK	ALL	BAM	BGN	CHF	DKK	HRK	HUF	ILS	MAD	MKD	NOK	PLN	RON	SEK	SGD	EUR															
DKK	AED	ALL	AUD	BAM	CHF	CNY	CZK	HKD	HUF	ILS	LBP	MAD	MKD	MOP	NOK	PLN	RON	RSD	SEK	SGD	THB	TWD	USD								
EUR	AED	ALL	BAM	CHF	CNY	CZK	HKD	HUF	ILS	LBP	MAD	MKD	MOP	NOK	PEN	PLN	RON	RSD	SEK	SGD	THB	TWD	USD								
HKD	AED	ALL	BAM	CHF	CNY	DKK	HRK	ILS	INR	LBP	MAD	MKD	MOP	PEN	PHP	SGD	THB	TWD	USD	EUR											
HRK	AED	ALL	BAM	CHF	CNY	CZK	HKD	HUF	ILS	LBP	MAD	MKD	MOP	NOK	PLN	RON	RSD	SEK	SGD	THB	TWD	USD									
HUF	ALL	BAM	BGN	CHF	CZK	DKK	HRK	MAD	MKD	NOK	PLN	RON	RSD	SEK	SGD	EUR															
IDR	ILS	MAD	MYR	PHP	SGD	THB																									
ILS	AED	ALL	BAM	BGN	CAD	CHF	CNY	CZK	DKK	HKD	HRK	IDR	INR	LBP	MAD	MKD	MOP	PEN	PHP	RON	RSD	SGD	THB	TWD	USD	EUR					
INR	AED	CNY	HKD	ILS	LBP	MAD	MOP	PHP	SGD	THB	TWD	USD																			
KRW	SGD																														
LBP	AED	BAM	CHF	CNY	DKK	HKD	HRK	ILS	INR	MAD	MKD	MOP	PEN	PHP	SGD	THB	TWD	USD	EUR												
MAD	AED	ALL	BAM	BGN	CHF	CNY	CZK	DKK	HKD	HRK	HUF	IDR	ILS	INR	LBP	MKD	MOP	MYR	NOK	PEN	PHP	PLN	RON	RSD	SEK	SGD	THB	TWD	USD	EUR	
MKD	AED	BAM	BGN	CHF	CNY	CZK	DKK	HKD	HRK	HUF	ILS	LBP	MAD	MOP	PEN	PLN	RON	RSD	SEK	SGD	THB	TWD	USD	EUR							
MOP	AED	ALL	BAM	BGN	CHF	CNY	DKK	HKD	HRK	ILS	INR	LBP	MAD	MKD	PEN	PHP	SGD	THB	TWD	USD	EUR										
MYR	IDR	MAD	SGD	TWD																											
NOK	BAM	BGN	CHF	CZK	DKK	HRK	HUF	MAD	PLN	RON	SEK	SGD	EUR																		
PEN	AED	BAM	BGN	CNY	HKD	ILS	LBP	MAD	MKD	MOP	PHP	SGD	THB	TWD	USD	EUR															
PHP	AED	CAD	CNY	HKD	IDR	ILS	INR	LBP	MAD	MOP	PEN	SGD	THB	TWD	USD																
PLN	ALL	BAM	BGN	CZK	DKK	HRK	HUF	MAD	MKD	NOK	RON	SEK	SGD	EUR																	
RON	ALL	BAM	BGN	CHF	CNY	CZK	DKK	HRK	HUF	ILS	MAD	MKD	NOK	PLN	RSD	SEK	SGD	EUR													
RSD	BAM	BGN	CHF	DKK	HRK	HUF	ILS	MAD	MKD	RON	SGD	THB	TWD	EUR																	
SEK	BAM	BGN	CZK	DKK	HRK	HUF	MAD	MKD	NOK	PLN	RON	EUR																			
SGD	AED	ALL	AUD	BAM	BGN	CAD	CHF	CNY	CZK	DKK	HKD	HRK	HUF	IDR	ILS	INR	KRW	LBP	MAD	MKD	MOP	MYR	NOK	PEN	PHP	PLN	RON	RSD	THB	TWD	USD	EUR
THB	AED	ALL	BAM	BGN	CAD	CHF	CNY	DKK	HKD	HRK	IDR	ILS	INR	LBP	MAD	MKD	MOP	PEN	PHP	RSD	SGD	TWD	USD	EUR							
TWD	AED	ALL	BAM	BGN	CHF	CNY	DKK	HKD	HRK	ILS	INR	LBP	MAD	MKD	MOP	MYR	PEN	PHP	RSD	SGD	THB	USD	EUR								
USD	AED	BAM	CHF	CNY	DKK	HKD	HRK	ILS	INR	LBP	MAD	MKD	MOP	PEN	PHP	SGD	THB	TWD	EUR												

Figure 11: Tableau de corrélations