Stage de Recherche
Interpolation du smile de corrélation sur les CDO

Thomas JANNAUD - X2005

Avril-Août 2008

Table des matières

Première partie

Cadre du stage

1 Présentation du stage

Cette courte partie a pour vocation à décrire le déroulement scientifique de ce stage : description du sujet, démarche, voies explorées, méthodes employées.

1.1 Introduction

Le stage de recherche de 3ème année à l'Ecole Polytechnique, en majeur mathématiques appliquées à la finance, s'est déroulé à la Société Générale, à Londres, au sein de l'équipe "Recherche Quantitative sur les Dérivés de Crédit". Cette équipe est constituée d'une dizaine de personnes, la plupart basées à Paris puisque nous n'étions que 2 (mon manager et moi) à Londres.

La première partie du stage a consisté essentiellement dans la lecture de documentation sur les CDO (le principal produit exotique traité sur notre floor) et leur pricing. La seconde partie de ce rapport traite ce sujet en profondeur. Le principal support, outre les recherches documentaires ponctuelles sur Internet, fût un cours donné à l'Ecole Centrale Paris sur les CDO [5] ainsi qu'un second article de la SG [3].

D'autre part, afin de m'habituer au contexte de mon stage, et afin de mieux "sentir" les paramètres importants des produits de crédit, j'ai implémenté leurs pricers. Pricer soit même un produit est en effet une excellente aide à la compréhension de celui-ci puisque cela nécessite de connaître les tenants et les aboutissants des différents paramètres entrant en jeu. Cela a aussi permis d'apprendre le C# par la pratique, language dérivé du Java et du Visual Basic, utilisé à la Société Générale. C'est d'ailleurs dans ce language que sont écrites les dll permettant de pricer dans cette entreprise tous les produits financiers existants.

J'ai donc pu comparer les résultats de mes pricers avec les prix plus justes de Marx, le système de pricing de la SG.

1.2 Sujet

La seconde partie du stage a consisté à entrer dans le sujet du stage proprement dit intitulé "interpolation et l'extrapolation du smile de corrélation".

Seule l'interpolation fut considérée pour l'instant, étant donné que le stage n'en est encore qu'à mi-parcours.

Pour tenter d'être clair sans entrer dans les détails que l'on donnera plus loin, disons que le pricing d'options simples (Call, Put, ...) tient compte du smile de volatilité implicité du marché. L'implicitation se fait grâce à la formule de Black-Scholes.

Il existe aujourd'hui un modèle utilisé "universellement" équivalent au modèle de Black-Scholes sur le marché des actions, mais sur le marché des dérivés de crédit. Ce modèle permet aussi d'impliciter un smile de quelque chose (la corrélation) et c'est cette donnée qui est importante.

Le problème est que le marché du crédit est moins liquide que le marché des actions : les produits traités sont plus "lourds", et valent des centaines de millions

d'euros. Ce qui fait qu'il n'y a que certaines valeurs qui se traitent et pas toutes. Pour faire un parallèle, cela revient à ce que le marché côte les puts à une date T à un strike de 5, 10, 15, 20, ... On connait alors le smile de volatilité sur ces valeurs, mais on ne sait pas ce qu'il vaut en 6,34 ou en 12. Il faut alors **interpoler** le smile entre les valeurs données. Concernant les valeurs en dessous de 5, ce n'est plus de l'interpolation, mais de **l'extrapolation**.

Mais ceci ne peut pas se faire les yeux fermés en interpolant linéairement par exemple. En effet il y a des contraintes de non-arbitrage à respecter. Ainsi un put de strike 5 a un prix plus élevé qu'un put de strike 6 par exemple. Et il faut donc faire attention à la manière d'interpoler puisque c'est elle qui détermine, *in fine*, le prix des produits.

1.3 Contraintes imposées

Une des principales difficultés du sujet réside dans une contrainte "pratique" vis-à-vis des traders, structureurs, ou toute autre personne qui aurait à utiliser notre interpolation : il faut qu'elle soit rapide. A titre d'exemple pricer un produit se fait en une ou deux secondes avec les dll de Marx (dll C# développées par la SG pour pricer tous les produits).

Le second problème est qu'il est demandé par le manager une méthode qui ne dépende pas du CDO choisi. Le problème se réduit donc à "étant donné une suite de points (abscisse, ordonnée), trouver la meilleure méthode interpolation/extrapolation qui donne pour la plupart des CDO des possibilités d'arbitrage minimales."

Cela à un impact considérable sur la manière de voir les choses et sur la stratégie à adopter.

1.4 Démarche et pistes explorées

Il y a eu 4 manières d'aborder le sujet. Les trois premières ont une une démarche "arbitrage" qui n'a pas vraiment fonctionné, et la dernière était une approche plus "interpolation pure".

Après avoir implémenté notre pricer, nous nous sommes rendus compte que l'on aurait justement besoin de calculer souvent des résultats. Nous avons choisi de créer un fichier énorme contenant des millions de résultats déjà calculés, que l'on charge dans la mémoire vive à chaque exécution. Nous avons ensuite créé un optimiseur qui part d'une fonction d'interpolation linéaire, et qui cherche à modifier localement les ordonnées afin d'obtenir une fonction qui minimise le plus possible, à un sens que l'on précisera, l'arbitrage. Si la création de ce fichier est trop longue, nous avons créé une seconde fonction d'optimisation, beaucoup moins coûteuse en temps de calcul.

Mis à part la dernière approche, on ne peut pas vraiment affirmer que les 3 premières approches se sont produites dans un ordre chronologique. La démarche fût en fait plus "brouillone" avec le recul, mais il n'y avait pas vraiment le choix non plus. On ne peut pas décider d'avoir de la chronologie dans les idées ! On a donc exploré

les 3 premières pistes plus ou moins en même temps, avançant sur 3 fronts différents, lâchant l'un quand on ne trouve plus grand chose pour mieux revenir sur un autre.

Concernant l'extrapolation, nous avons lu plusieurs documents traitant du sujet, et s'inspirant de ces sources, nous avons developpé nos propres méthodes. L'extrapolation étant un sujet sensible, nos méthodes ne sont pas divulgués ici.

Il y a aussi eu des moments un peu "difficiles", lorsque l'on se rend compte qu'il y a une erreur dans notre algorithme de calcul et que les résultats et les courbes que l'on trace depuis 2 semaines sont donc tous à refaire. Le stage ne fût donc pas aussi structuré que le rapport résultant ! Mais c'est le propre d'un stage de recherche. Ce qui a compté pour nous y retrouver était d'avoir une "grande ligne directrice" : on a eu quelques idées de piste au départ, puis en les explorant on a pensé à d'autres pistes. Mais on a toujours eu l'objectif final en vue.

1.4.1 Optimisation du spread

La première approche fût de s'intéresser à la courbe de spread obtenue en fonction d'une interpolation donnée, et d'optimiser numériquement l'interpolation afin d'obtenir la meilleure courbe de spread possible.

l'optimisation numérique consiste dans notre cas à modifier très peu les ordonnées d'une fonction, à vérifier si cela optimise plus un critère que "l'ancienne" fonction, et à réitérer le procédé. Le critère est ici "y-a t'il beaucoup d'arbitrage sur la courbe de spread résultant de l'interpolation de la corrélation ?"

Ce faisant, on se serait penché sur la forme de l'interpolation rendant optimale le spread, et en comparant plusieurs graphes, on aurait ensuite recherché une équation ou une formule permettant de retrouver directement (de manière approchée) l'interpolation sans refaire tourner l'algorithme d'optimisation.

Remarque : Par rapport au parallèle dressé plus avant, le spread est ce qui joue le rôle du prix dans le domaine du crédit.

Cela n'a malheureusement donné que peu de résultats, ce qui fût quelque peu décevant. Plusieurs raisons à cela : l'algorithme de minimisation partait d'une fonction, la modifiait localement, et décidait de la "qualité" de cette nouvelle fonction au travers d'une fonction d'évaluation.

La première raison est qu'il n'est pas si simple que cela de choisir une fonction d'évaluation : il faut jouer sur "est ce que la fonction est lisse", tout autant que "n'y a t'il pas trop d'arbitrage ?" sachant que ces questions ont toutes des réponses différentes suivant les "normes" que l'on choisit (somme des carrés des erreurs de lissages, ou des valeurs absolues ...) la calibration n'est pas simple.

D'autre part le processus de modification de la fonction "bougeait" une seule ordonnée à la fois, créant un "pic" local, même si calibré pour être de faible intensité. Ceci crée donc un défaut dans le lissage pour espérer gagner en arbitrage. Mais sur toutes les simulations l'algorithme convergeait vers une fonction qui ne nous intéressait

pas : la création d'un "pic" local s'était amplifiée avec les itérations en un seul endroit., dû à la fonction d'évaluation qui préférait augmenter un pic afin de créer une "marche" sur l'arbitrage plutôt que de lisser la fonction.

Enfin, l'idée initiale était d'utiliser la méthode du recuit simulé, qui fonctionne très bien pour le problème du voyageur de commerce par exemple. Le principe de cet algorithme est plus ou moins le même que celui que l'on a fait, à une différence près : on part d'une fonction, on la modifie légèrement, et si elle a une "qualité" meilleure que la fonction précédente, on conserve le changement. Ou bien on accepte avec une certaine probabilité une nouvelle fonction qui n'est pas forcément meilleure que la précédente (probabilité dépendant du nombre d'itérations déjà réalisées et de la différence numérique entre les 2 "qualités" évaluées).

Cet algorithme est très alléchant : en théorie il peut permettre d'éviter de converger vers un extremum local mais pas global en "sautant" avec une certaine probabilité pour "s'échapper" de ces mauvais puits attractifs. Mais en pratique, il est très délicat à mettre en oeuvre : les versions que l'on a testées voyaient leur "qualité" se dégrader progressivement plutôt que de grimper, suite à des probabilités de saut trop élevées. Les changements en ce sens ne résolvaient rien, à part à revenir à l'algorithme simple d'optimisation (basique, sans probabilités).

1.4.2 Optimisation de la Market Law

Cette seconde approche fût semblable à la première. Cette fois, ce n'est pas la courbe de spread qui était regardée, mais une courbe plus théorique de probabilité de défaut implcitée par le marché. Elle possède de fait naturellement des contraintes de continuité et de croissance. Il n'y a pas d'arbitrage réel qui correspond à cette courbe abstraite, mais on a pu se rendre compte au moyen de tests numériques qu'il y avait un rapport entre cette courbe abstraite et les prix réels du marché.

L'idée de la fonction "abstraite" associée au marché est tirée de l'article "Pricing a CDO with a smile", SG, Février 2005 et il a suffit de réajuster notre fonction d'évaluation (sans modifier quoique ce soit d'autre à notre fonction d'optimisation).

Ceci ne donnait pas de résultats très cohérents non plus, pour les raisons invoquées plus haut.

1.4.3 Recherche mathématique

Nous avons aussi décidé de chercher des conditions théoriques sur la fonction d'interpolation. Ces conditions ne sont pas faciles à trouver puisque les CDO sont des produits somme toute complexes et qu'il n'y a pas de formule fermée.

Cependant après des recherches laborieuses qui semblaient ne pas aboutir, puis en reprenant le travail et en effectuant des approximations successives nous avons réussi à formuler une contrainte théorique sur le smile de corrélation, respectant l'absence d'arbitrage.

Malheureusement l'équation différentielle obtenue est beaucoup trop compliquée pour être utilisée aisément. Et numériquement elle ne donne pas de bons résultats non plus.

1.4.4 Recherche sur l'interpolation

Puisque ces trois premières méthodes ne fonctionnaient pas, il nous a fallu réfléchir à une quatrième. Nous avons donc décidé de nous intéresser purement à l'interpolation de points, c'est à dire trouver une manière d'interpoler la plus "lisse" possible, tout en veillant à ce qu'elle respecte certaines contraintes de monotonie par exemple. Nous nous sommes inspirés de [4] Puis, étant donnée cette interpolation, on constate simplement quelle est la forme de la courbe de spread qui en résulte. Nous avons implémenté beaucoup de méthodes différentes afin de pouvoir faire un choix parmi le plus de méthodes possible et évaluer les avantages et les inconvénients de chacune.

Et ce choix s'est avéré plus approprié que les précédents. En effet, il apparaissait clair dans certains articles [2] que les discontinuités (d'une fonction ou de ses dérivées) créaient des discontinuités sur la courbe de spread, et par voie de conséquence des possibilités d'arbitrage. Le choix judicieux semble avec le recul être de faire fi de la courbe de spread résultante et de se concentrer sur des manières d'interpoler.

Il nous a donc fallu nous documenter une seconde fois sur un sujet complètement différent de la finance, beaucoup plus "maths appliquées". Mais si la documentation sur l'interpolation est plutôt bien fournie concernant l'interpolation d'une fonction par des polynômes par exemple, elle est assez faible concernant l'interpolation entre un nombre fini de points et il nous a fallu inventer nos propres méthodes.

Il y a en fait un intérêt certain à ce qu'il n'y ait pas beaucoup de documentation sur un sujet : d'une part c'est un gain de temps de ne pas avoir beaucoup d'articles à lire, et d'autre part on peut réfléchir à quelque chose de neuf puisque nos pensées ne sont pas biaisées par des lectures sur le sujet.

1.5 Conclusion

Ce stage fut l'occasion de découvrir d'une part une entreprise en fonctionnement et d'autre part de se lancer dans un projet de recherche de relativement longue durée (vis-à-vis des expériences que l'on a eu), seul, à plein temps.

En effet dans une salle de trading cohabitent beaucoup de personnes, des quants aux structureurs en passant par traders, sales, commandos, ... Il n'est pas facile de s'expliquer la manière dont les équipes s'imbriquent les unes dans les autres pour former un tout, et lire des explications sur le sujet sans l'avoir vécu n'apporte pas toujours beaucoup d'informations. C'est important d'avoir été au contact des équipes pour comprendre leur rôle.

Concernant le métier de quant, cela donne une idée de la manière dont fonctionnent les chercheurs, et de la vie qu'ils mènent. A l'aube de s'engager dans la vie active et de se décider pour un métier, choix non définitif à notre âge mais qui représente toutefois une étape importante de notre vie, il est essentiel d'avoir une petite idée sur les divers métiers qu'offre une grande banque.

Le travail de chercheur est assez particulier : on peut être sur une voie pendant 1 semaine, puis se rendre compte que la voie est bloquée et qu'il faut en choisir une autre. On a alors un sentiment au pire de "défaite" et au mieux de temps perdu. A

ce titre, la recherche a des côtés décourageants. Il faut aussi lire beaucoup d'articles pour se faire une idée des connaissances accumulées sur un sujet, et la lecture est souvent un travail fastidieux auquel il faut pourtant s'habituer. Il faut bien oser dire que certains articles ne fourmillent en général pas d'explications mais se perdent dans les généralités ce qui rend la tâche difficile.

Il y a bien entendu les "bons moments" aussi : apprendre puis comprendre de nouveaux modèles, s'intéresser à de nouvelles méthodes, discuter les résultats avec ses collègues, voir son programme marcher avec satisfaction, ...

2 Le marché du crédit

Après une définition des différents produits (CDS, CDO), nous lèverons le voile sur leur prix et la manière de les pricer.

2.1 Les CDS

2.1.1 Définition

Les CDS sont au marché du crédit ce que les actions sont au marché equity : ils représentent en quelque sorte l'entité de base, sur laquelle des produits dérivés vont être créés.

Une entité A prête de l'argent à B. B doit lui rembourser une certaine somme (un peu plus que le prêt) à la date T. Cependant A veut s'assurer contre le risque de défaut de B (faillite, ...). A fait appel à une entité externe C qui va prendre le risque de le couvrir moyennant une certaine somme payable tous les 3 mois par exemple.

Ainsi :
Si B ne fait pas défaut avant T, il rembourse complètement A.
A verse de l'argent (le **spread**) à C tous les 3 mois tant que B n'a pas fait défaut.

Si B fait défaut avant T, C verse à A l'argent que B lui doit.

Ce type de contrat porte le nom de CDS : Credit Default Swap.
En pratique, même si B fait défaut, il peut rembourser une petite partie de ce qu'il doit à A : c'est ce qu'on appelle le **taux de recouvrement**. Si B doit rembourser N en T, s'il fait défaut il délivrera bien en mal en la somme N*R. C ne devra alors rembourser à A que N*(1-R).

Dans la pratique encore, un CDS est un contrat, et les termes de celui-ci sont stipulés de manière très précise : définition de ce qu'est un "défaut" (faillite de B, incapacité à rembourser, négociation dans la manière de rembourser la dette (= "re-structuring"), ...), dates des paiements des spreads. Celles-ci sont en fait normalisées, de manière à pouvoir rendre le marché des CDS plus liquide : les maturités des CDS, ainsi que les dates de paiement de spreads sont les 20 mars, 20 juin, 20 septembre et 20 décembre.

2.1.2 Prix

C devra verser à A :
$FloatLeg = E^Q(B_\tau * N * (1-R) * 1_{\{\tau \leq T)})$
La float Leg est aussi appelée jambe de prime ; elle correspond à l'espérance sous la probabilité risque-neutre de la prime d'assurance que payera C à A. Si le taux d'intérêt était nul, on aurait $FloatLeg = N(1-R).Q(\tau \leq T)$. C'est à dire probabilité risque neutre de défaut multiplié par ce qu'il faudrait payer s'il y avait défaut.

Notations

B_t est le zéro-coupon (que l'on peut prendre égal à $\exp(-r * t)$).
τ est l'instant de défaut de B.

Quant à lui, A devra verser à C (en notant s le spread par unité de nominal) :

$$FixedLeg = E^Q(\sum_{i=1}^{n} B_{t_i} * N * s * \delta_i * 1_{\tau > t_i} + E^Q(\sum_{i=1}^{n} B_\tau * N * s * \delta_\tau * 1_{t_{i-1} < \tau < t_i})$$

(le spread se paye tous les t_i et est proportionnel à la période passée ($delta_i = t_{i-1} < \tau < t_i = 3$ mois environ). S'il y a défaut avec $t_{i-1} < \tau < t_i$, il faut donc payer le spread sur $[t_{i-1}, \tau]$ car on est protégé sur cette période.

Le spread de marché (ou fair-spread) est celui qui égalise la Floatleg et la FixedLeg.

2.1.3 Modèle

En se référant aux formules ci-dessus, on note que **le problème revient entièrement à modéliser l'instant de défaut de B**.

En pratique, τ est vu comme une variable aléatoire qui suit une loi d'extinction de Poisson : sur $[t, t + dt]$, la probabilité de défaut sachant qu'il n'y a pas eu défaut avant est $\lambda_t * dt$.

Ainsi, $P(\tau > t) = exp(-\int_0^t \lambda_s ds)$

Pour calibrer λ, comme le marché des CDS est assez liquide, on dispose des *fair-spread* d'un certain CDS sur plusieurs maturités. En supposant λ constant par morceaux, on en déduit λ par **bootstrapping**.

Exemple : on connait le prix d'un CDS (son spread) pour T = 5 ans, 7 ans et 10 ans.

On cherche $\lambda_{[0-5]}$ tel que spread($\lambda_{[0-5]}$) = spread_marché à 5 ans. Ensuite, on cherche $\lambda_{[5-7]}$ tel que notre spread égalisera celui du marché à 7 ans, puisque maintenant $\lambda_{[0-5]}$ est connu. Et ainsi de suite.

Connaissant enfin λ complètement, on peut en déduire le prix d'un CDS à toute date comprise entre 0 et 10 ans.

2.1.4 Autres modèles

Merton Ce modèle est l'un des tout premiers concernant les CDS. Il est un petit peu l'équivalent du modèle de Black Scholes pour ceux-ci.

Voici le principe : plutôt que de s'en tenir à un modèle abstrait comme celui où la probabilité de défaut suit une loi exponentielle, on préfère ici une représentation

plus économique de ce qu'est un défaut : il y a défaut quand l'entreprise ne peut pas s'acquitter de sa dette, c'est à dire si sa valeur liquide totale (actifs + passifs) est en dessous d'un certain seuil (la dette).

Formalisation :
$A_t = E_t + D_t$ où A_t représente la valeur d'une entreprise, E_t ses actifs, et D_t sa dette.

L'entreprise doit rembourser L en T. Il y aura défaut si $A_T < L$.

On suppose que $\frac{dA_t}{A_t} = r.dt + \sigma.dW_t^Q$.

Le spread doit se voir comme la différence entre le rendement sans risque et le rendement risqué.
$S(t,\ T) = \overline{Y}(t,T) - Y(t,T)$ où $Y = -\frac{\ln B(t,T)}{T-t}$ et $\overline{Y} = -\frac{\ln \overline{B}(t,T)}{T-t}$ avec $\overline{B}(t,T) = B(t,T) * E^Q(\min(\frac{A_T}{L}, 1)|F_t)$

Rq : $B(t,T)$ est le zéro-coupon payé à la date t, de maturité T. $\min(\frac{A_T}{L}, 1)$ signifie que si l'entreprise ne fait pas défaut, on récupère notre mise (1), sinon, on récupère un certain pourcentage sur ce que l'on a investi ($\frac{A_t}{L}$)

Calibration de σ :

E_t peut désigner la valeur de l'action de l'entreprise. On peut donc connaître σ_E.

$D_T = \min(A_T, L)$ d'où $D_t = E^Q(\exp(-r(T-t)) * \min(A_T, L))$
et donc $E_T = E^Q(\exp(-r(T-t)) * (A_T - L)^+)$ (hypothèse de complétude du marché)

$$\text{soit } \left\{ \begin{array}{c} E_t = A_t.N(d_1) - B(t,T).L.N(d_2) \text{ (formule de Black Scholes)} \\ \sigma_E = \frac{A_t}{E_t}.N(d_1).\sigma \end{array} \right\}$$

En inversant ce système numériquement (d_1 et d_2 dépendent de σ_E) on peut connaître A_t et σ.

Remarque : Inconvénients du modèle :
_ Nous n'avons pas implémenté ce modèle. Cependant la documentation à ce sujet laisse à entendre que ce modèle sous-évalue les spreads sur les petites maturités, ou de manière équivalente que la probabilité de défaut est sous-évaluée par rapport à celle implicitée sur le marché.
_ D'autre part, ce modèle ne donne pas de temps de défaut : il prévoit un défaut à maturité T, ou rien ; il ne prend pas en compte le "passé" avant T.

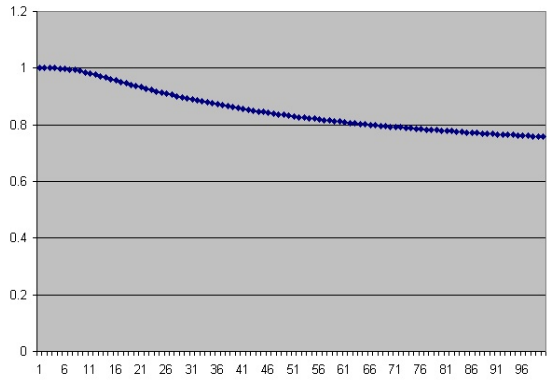


FIG. 1 – allure de la probabilité de défaut donnée par le modèle de Merton, en fonction du temps

Modèle "Credit Grade" Ce modèle fut développé par JP Morgan en 2001. Il reprend le modèle de Merton dans la mesure où il se base sur une donnée économique encore une fois et pas abstraite.

Principe : une entreprise a une valeur V_t (qui ne représente pas exactement le cours de l'action, mais plutôt la capacité de l'entreprise à rembourser sa dette, sa qualité de crédit).

Quand cette valeur descend en dessous d'une certaine valeur barrière (fixe ou aléatoire) il y a défaut.

Implémentation :

$\frac{dV_t}{V_t} = \sigma dW_t$ où V_t est la "valeur" d'une entreprise qui doit rembourser L en T.

Il y aura défaut si $V_t \leq L$.

On peut prendre L log-normale : $L = \overline{L}\exp(\lambda Z - \frac{\lambda^2}{2})$ où Z suit une loi normale centrée réduite.

$\tau = \inf\{t > 0, V_t < L\}$

Calibration de σ :

On a surtout accès à la valeur de l'action d'une entreprise, et il faut donc avoir un V_t qui dépende de L et de S_t (cours de l'action).

Comme conditions aux limites on peut poser :

_ lorsque $S \to 0, V_t$ est de l'ordre de grandeur de L

_ et lorsque $S \to \infty, V_t \sim S$.

Par exemple $V_t = S + L$ convient.

D'autre part, $V_t = f(S)$ donc par application de la formule d'Itô, on obtient :

$\sigma V_t.dW_t = f'(S).(\sigma_S.S.dW_t + \alpha.dt) + \frac{1}{2}.f''(S).\sigma_S^2.dt$
d'où
$\sigma V_t = f'(S).\sigma_S.S$
soit $\sigma = \sigma_S.\frac{S}{V}.\frac{\partial V}{\partial S}$

On peut donc choisir "arbitrairement" $\sigma = \sigma_S.\frac{S}{S+L}$. Et afin d'avoir une volatilité non stochastique, on peut utiliser une valeur de référence S^* constante au lieu de S, et de même pour σ_S.

Résultats :
Cherchons la loi de défaut donnée par ce modèle, et commençons par un résultat utile :

Proposition I.1 *Si W_t est un mouvement brownien, alors* $P(\inf_{s\leq t}(W_s - \alpha.s) > b) = N(\frac{-b-a.t}{\sqrt{t}}) - e^{-2a.b}N(\frac{b-a.t}{\sqrt{t}})$

Il s'ensuit que la loi de défaut s'écrit $P(\tau > t) = P(\inf_{s\leq t} V_s > \overline{L}\exp(\lambda Z - \frac{\lambda^2}{2})) = P(\inf_{s\leq t}\exp(\sigma W_s - \frac{\sigma^2}{2}s) > \frac{\overline{L}}{V_0}\exp(\lambda Z - \frac{\lambda^2}{2})) = P(\inf_{s\leq t}(\sigma W_s - \frac{\sigma^2}{2}s) > \ln(\frac{\overline{L}}{V_0}) + \lambda Z - \frac{\lambda^2}{2}) = P(\inf_{s\leq t}(W_s - \frac{\sigma}{2}s) > \frac{1}{\sigma}\ln(\frac{\overline{L}}{V_0}) + \frac{\lambda}{\sigma}Z - \frac{\lambda^2}{2\sigma}) = \int_{\mathbb{R}}\varphi(z)dz.(N(\frac{-F(z)-\frac{\sigma^2 t}{2}}{\sigma\sqrt{t}}) - e^{-F(z)}N(\frac{F(z)-\frac{\sigma^2 t}{2}}{\sigma\sqrt{t}}))$ où $F(z) = \ln(\frac{\overline{L}}{V_0}) + \lambda z - \frac{\lambda^2}{2})$

Voyons l'allure, en prenant $\frac{\overline{L}}{V_0} = \frac{1}{2}$, $\sigma = 0.3$ et $\lambda = 0.3$.

Conclusion Les lois de probabilités des temps de défaut générées par ces 2 derniers modèles _qui se basent sur des raisonnements économiques_ sont assez complexes, et impliquent une utilisation de Monte-Carlo puisqu'il n'y a pas de formule fermée (ou alors simuler la loi, stocker beaucoup de ses valeurs et rechercher un résultat dans ces données ; et encore on peut avoir besoin de la dérivée, ou d'intégrer...).

En outre, le modèle de Merton ne donne au final qu'une probabilité de défaut à maturité fixée (au lieu de donner en plus le temps de défaut associé) et il suffit donc de connaître un seul réel (la probabilité) plutôt que de simuler beaucoup de trajectoires. Son utilité fût en fait de pricer les premiers CDS, quand le marché n'en était encore qu'à ses débuts.

Le second modèle semble plus "précis", ayant un sens encore économique mais donnant en plus de la probabilité de défaut l'instant de défaut. Mais la loi de celui-ci est très complexe. Et comme on a pu s'en rendre compte dans le pricing d'un CDS, ce qui importe au final est la densité de la loi de l'instant de défaut τ, pas la manière dont on arrive à cette loi. Si la loi exponentielle n'a certes aucun fondement économique (quoique dans la "nature" elle a un sens concret), quand elle est bien paramétrée (par une fonction λ en escalier) elle approche remarquablement la loi donnée par un modèle plus terre-à-terre.

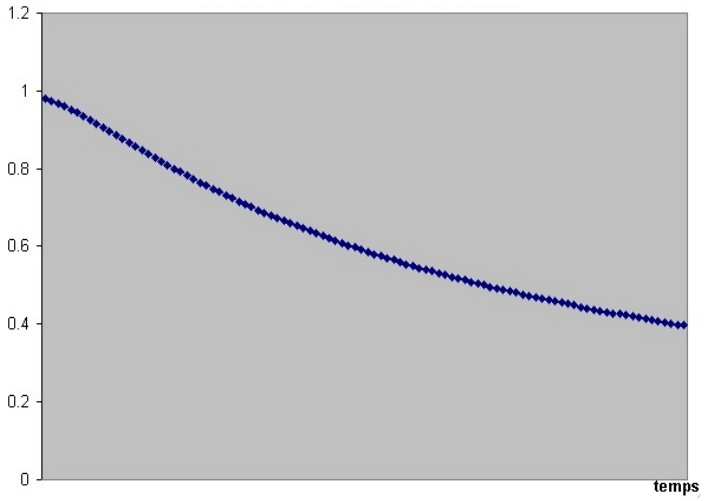


FIG. 2 – Probabilité de défaut donnée par le modèle Crédit Grade

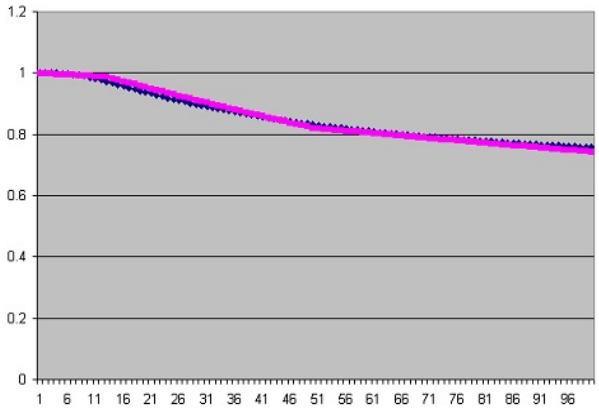


FIG. 3 – Comparaison entre les probabilités données par le modèle de Merton et la loi exponentielle

Exemple : En bleu, la probabilité donnée par le modèle de Merton (barrière fixe).

En rose, une loi exponentielle généralisée avec λ en escalier (3 "marches" seulement, déterminées "à la main"). Nous n'avons pas fait beaucoup d'effort pour calculer les λ et cependant la loi exponentielle en escalier approche très bien la loi donnée par le modèle de Merton.

Conclusion La loi exponentielle se révèle très flexible, et très simple à mettre en place. D'autre part elle permet des calculs exacts dans ceux de la Fixed Leg et de la Float Leg.

2.2 Les CDO

2.2.1 Définition

CDO est l'abbréviation de "Collateralised debt obligation" : un CDO est en quelque sorte un "panier de CDS" : on regroupe un certain nombre de CDS (une centaine en pratique) en un seul produit. Acheter un CDS revient à vendre de la protection : on reçoit un spread régulièrement, mais on paye s'il y a un défaut. Le CDO fonctionne sur le même principe : acheter une **part** de CDO revient à vendre de la protection. Mais le CDO est un produit plus complexe que le CDS : ici on peut décider quelle part de risque on prend (et par voie de conséquence quelle sera notre rendement risqué).

Pour simplifier, considérons 100 CDS avec des nominaux de 1 et une recovery de 0, payant tous le même spread s.

Tant qu'il n'y a aucun défaut, il y a 100 * s de spread qui est payé. Dès qu'il y a un défaut, 99 * s, puis ainsi de suite.

On crée abstraitement les "tranches" [0-1%[, [1%-2%[, ... Acheter une "tranche" revient à vendre de la protection mais uniquement sur cette tranche.

Exemple : on achète la tranche [3%-4%[. On reçoit une certaine quantité d'argent (à définir) régulièrement. La contrepartie, c'est qu'**on rembourse le 4ème défaut** parmi l'ensemble des 100 CDS (s'il arrive avant la maturité T) et on cesse alors de recevoir notre argent/prime de risque.

Ainsi, s'il y a 0, 1, 2 ou 3 défauts cela ne change rien pour nous : on reçoit ce qu'il était prévu initialement. Qu'il y en ait 4 ou plus revient aussi au même : on paye la prime (=on rembourse le défaut) et on cesse de recevoir notre spread.

En pratique, les "tranches" ne sont pas aussi étroites : [0%-3%[, [3%-6%[, [6%-9%[, [9%-12%[, [12%-22%[, [22%-100%] (il s'agit des tranches junior, mezzanine, senior et super-senior), et par ailleurs il n'y a pas exactement 100 CDS et ceux-ci n'ont pas tous la même recovery, et ils ne payent pas tous le même spread. Il se peut donc que la tranche sur laquelle on vend de la protection ne soit "croquée" que sur une partie. Les recovery sont par contre connues à l'avance.

Dire que l'on vend de la protection sur la tranche [a%, b%[signifie que l'on couvre la partie du montant des défauts ($= \sum_{\text{i fait défaut}} N_i * (1 - R_i)$) comprise entre $a\% * \sum_{i=1}^{N} N_i$

et $b\% * \sum_{i=1}^{N} N_i$.

On paye alors le défaut correspondant, et on continue de recevoir du spread sur le reste de la tranche "non touchée" (que l'on continue d'assurer).

Il est assez clair que plus la tranche que l'on assure est "basse", plus l'on prend de risques. Il faut donc que l'on soit mieux rémunéré sur ces tranches.

Il y a donc une sorte de "redistribution" de la somme des spreads reçus : on paye les tranches senior (au prix convenu au départ), puis les mezzanine (s'il reste de l'argent), puis enfin les junior (s'il reste de l'argent). Mais ces derniers sont payés au prorata du risque qu'ils encourent.

2.2.2 Prix

Encore une fois, le spread reçu sur une tranche se calcule en égalisant en espérance risque-neutre ce que l'on va recevoir en spread, et ce que l'on risque de payer.

Notations

B_t est le zéro-coupon (que l'on peut prendre égal à $\exp(-r * t)$).
$L_t^{[K_1,K_2]}$ = perte à l'instant t sur la tranche $[K_1, K_2]$ (K_1, K_2 en pourcents) = $(L_t - K_1)^+ - (L_t - K_2)^+$ où L_t est la perte cumulée à l'instant t sur le portefeuille (= somme des défauts sur les CDS du panier, jusqu'à t).

$FloatLeg = E^Q(\int_0^T B_t.dL_t^{[K_1,K_2]})$

$FixedLeg = E^Q(s * \sum_{i=1}^{n} B_{t_i} * \int_{t_{i-1}}^{t_i} (K_2 - K_1 - L_t^{[K_1,K_2]})dt)$

Le *fair-spread* est le spread s qui égalise FloatLeg et FixedLeg.

Cette image rend bien l'intuition que l'on a sur les prix des CDO :
une tranche équity va être beaucoup plus risquée si la corrélation est faible, et donc son prix est beaucoup plus fort. Il suffit en effet qu'il n'y ait qu'un ou deux défauts pour faire "tomber" la tranche et cela est plus probable dans le cas d'une très faible corrélation (indépendance des défauts). C'est l'inverse qui se produit sur les tranches senior : elles seront atteintes s'il y a beaucoup de défauts, et cela arrivera plus facilement si les noms sont très corrélés. (dans le cas le plus extrême, la corrélation vaut 1, et la probabilité que tous les noms fassent défaut est la probabilité d'un seul défaut ; dans le cas où la corrélation vaut 0, la probabilité vaut le produit de toutes les probabilités... qui est quasi nul)
Entre ces tranches, les tranches mezzanines sont un peu sujettes à plusieurs tendances et il est plus difficile d'expliquer simplement leur prix en fonction de la corrélation.

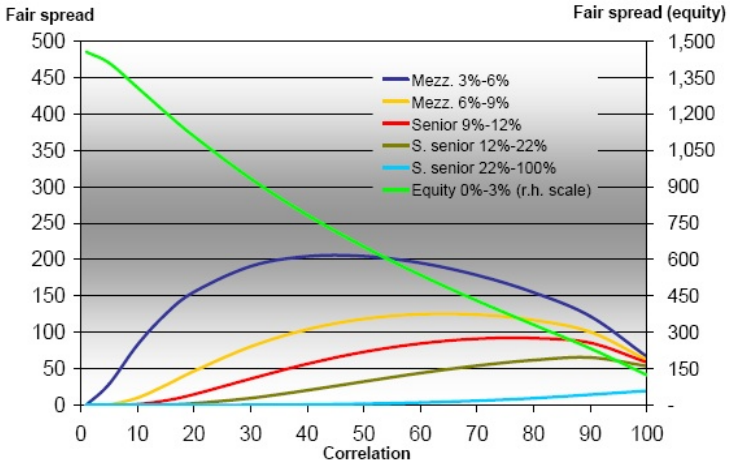


FIG. 4 – Fair spread en fonction de la corrélation

2.2.3 Modèle

Modéliser L_t revient à modéliser les instants des temps de défaut $\tau_1, \tau_2, ...$ Ce qui compte, cette fois, c'est la dépendance entre ces variables aléatoires. En effet, les temps de défaut ne sont pas des objets purement abstraits, ils correspondent à des situations bien réelles (faillite, ...) et quand "l'économie mondiale" ne va pas très bien, c'est que beaucoup d'entreprises risquent de faire défaut : il y a bien une **corrélation entre les temps de défaut**.

Plusieurs modèles ont été inventés pour traiter de ces variables aléatoires. En pratique, c'est celui de la copule gaussienne qui est utilisé.

Explication :

- pour les CDS, on connait la loi de probabilité $P(\tau \geq t)$. On n'a donc besoin que d'une variable aléatoire uniforme sur [0-1].

- ici on veut corréler les temps. On **garde** les lois de probabilité $P(\tau \geq t)$ que l'on suppose connues. Il nous faut donc simuler un vecteur aléatoire corrélé sur $[0,1]^n$ et pour ce faire on utilise le résultat suivant : si X est une variable aléatoire, et F() sa fonction de partition, alors F(X) suit une loi uniforme sur 0-1. (Résultat démontré en Appendice)

On choisit Z, ϵ_1, ... des gaussiennes centrées réduites indépendantes ainsi que $\rho_1, \rho_2, ...$

On pose $X_i = \rho_i.Z + \sqrt{1-\rho_i^2}.\epsilon_i$ de telle sorte que les Xi sont des gaussiennes centrées réduites et $Corr(X_i, X_j) = \rho_i.\rho_j$

Alors $(F(X_1), F(X_2), ...)$ est une copule gaussienne de variables corrélées (F(X) suit une loi uniforme sur 0-1).

En pratique toutefois, plutôt que de choisir $\rho_1, \rho_2, ...$on prend $\rho_1 = \rho_2 = ... = \rho$ afin de faciliter les calculs et l'harmonisation des modèles sur les marchés.

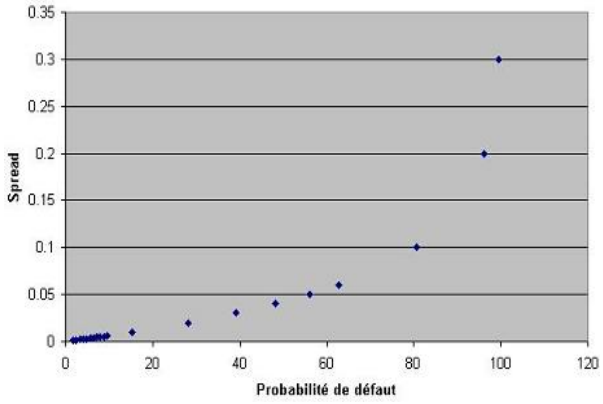


FIG. 5 – Spread d'un CDS en fonction de la probabilité de défaut

Remarque : Z représente ici la "variable de marché", ou encore "comment va l'économie mondiale/nationale".

_ Si X_1, X_2, ... sont des variables aléatoires réelles quelconques, alors C : $\left\{ \begin{array}{c} [0,1]^n \to [0,1] \\ (x_1, ...x_n) \to C(x_1, ..., x_n) \end{array} \right\}$ est une copule si $\forall (x_1, ..., x_n), C(x_1, ..., x_n) = P(X_1 \leq x_1, ..., X_n \leq x_n)$.

La notion de Copule est surtout théorique puisqu'il n'existe pas (encore !) de formule exprimant par exemple la fonction C pour n gaussiennes corrélées.

2.3 Implémentation des pricers

2.3.1 Pricer de CDS

Le pricer de CDS a été implémenté de 2 manières : par MonteCarlo et par calcul direct.

Il nous faut en effet calculer la FloatLeg et la FixedLeg (divisée par le spread, que l'on appelle aussi RBPV), qui sont toutes les deux des espérances de quelque chose.

La loi étant connue, on peut soit calculer les espérances correspondantes par MonteCarlo ($Espérance = \lim_{n\to\infty}(\frac{1}{n}\sum_{i=1}^{n} X_i)$) soit par intégrale sur la loi ($Espérance = \int x d\mu(x)$).

Ces 2 méthodes sont équivalentes en temps et en précision.

Les difficultés dans l'implémentation consistent souvent dans la connaissance des fonctions elles-mêmes. (ex : $\int_{-\infty}^{x} \exp(-\frac{x^2}{2})dx$; cf Appendice pour l'implémentation de cette fonction)

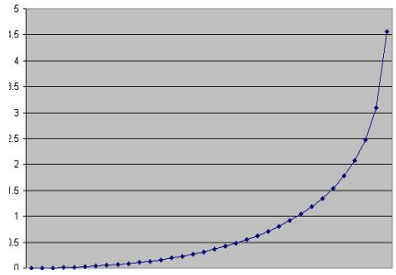


FIG. 6 – $y = -xln(1-x)$

Cette courbe représente le spread versé par un CDS en fonction de la probabilité de défaut estimée, en supposant l'intensité de défaut constante.

Sur une très grande plage (en pratique la probabilité de défaut d'une entreprise dépasse rarement les 50% !), le spread est affine en la probabilité de défaut à horizon fixé. On peut essayer de comprendre le résultat sur les équations :

prenons $r = 0$ pour simplifier. Il vient $FloatLeg = N(1-R).P_{défaut}$ et $FixedLeg = E^Q(\sum_{i=1}^{n} N*s*\delta_i*1_{\{\tau>t_i)}) + E^Q(\sum_{i=1}^{n} N*s*\delta_\tau*1_{\{t_{i-1}<\tau<t_i)}) \approx N*s*\sum_{i=1}^{n} \delta_i*E^Q(1_{\{\tau>t_i)}) = N*s*\alpha*\sum_{i=1}^{n} P(\tau > t_i) \approx N*s*\alpha*E^Q(\tau)$ (δ_i représente plus ou moins une durée de 3 mois, qui est variable en nombre de jours suivant la date à laquelle on se trouve)

Ainsi le fair spread vaut $s^* \approx \beta \frac{P_{défaut}}{E^Q(\tau)}$

Si τ a une intensité de défaut constante, $P_{défaut} = 1 - e^{-\lambda T}$ et $E^Q(\tau) = \frac{1}{\lambda}$ d'où $s^* \approx \beta.P_{défaut}.\lambda = c.P_{défaut}.\ln(1 - P_{défaut})$

$$\boxed{s \approx c.P_{défaut}.\ln(1 - P_{défaut})}$$

Cette approximation permet bien d'obtenir le même type de courbe que la courbe de fair-spread simulée.

2.3.2 Pricer de CDO

Cette fois 3 implémentations ont été effectuées : par MonteCarlo, par récurrence, et par loi Beta.

Par Monte-Carlo La programmation par Monte Carlo est "sans surprises" bien que quelque peu longue : il faut en effet tirer $\tau_1, \tau_2, ...$ puis trier ces temps par ordre croissant, et retracer toute la "vie" du produit CDO, c'est à dire tous les primes à payer, quels coupons l'on reçoit, ...

Cela marche très bien mais le temps de calcul est assez long (1 minute). On peut réduire ce temps à 10 secondes en faisant 6 fois moins de simulations, sans réduire significativement la précision. En effet, rappelons que la convergence en précision de

Monte Carlo est en $\frac{1}{\sqrt{N}}$ où N est le nombre de simulations. Cependant, pour le calcul de delta (nécessitant les dérivées partielles de certains résultats) la meilleure précision est nécessaire.

Notons qu'on peut réduire un peu le temps de calcul en remarquant que le cas où il n'y a aucun défaut n'est pas forcément un événement rare, et que l'on peut donc enregistrer la valeur de la jambe de protection pour ce cas.

Par récurrence La programmation par récurrence rejoint le calcul "exact" par intégrale du pricer de CDS. Le principe est de calculer exactement toutes les probabilités P(Lt = a) pour tout a, par récurrence.

L'idée est que l'on a "a priori" quelque chose comme $P_{N+1\ émetteurs}(L_t = a) = P_{N\ émetteurs}(L_t = a) * P(\tau_{n+1} > T) + P_{N\ émetteurs}(L_t = a - l_{n+1}) * P(\tau_{n+1} \leq T)$

En fait, ceci est faux car il n'y a pas indépendance entre les τ_i ! Mais l'égalité reste vraie quand on conditionne suivant la variable Z "de marché". Il faut donc tout calculer pour chaque z et calculer l'intégrale ensuite et l'on a en réalité :

$P_{N+1émetteurs}(L_t = a|Z) = P_{N\ émetteurs}(L_t = a|Z) * P(\tau_{n+1} > T|Z)$
$+P_{N\ émetteurs}(L_t = a - l_{n+1}|Z) * P(\tau_{n+1} \leq T|Z)$

Implémentation :

Pour commencer il faut trouver une "loss default unit", i.e le plus grand nombre u tel que toutes les pertes des CDS (les $N_i.(1 - R_i)$) soient multiples de ce nombre u. C'est en quelque sorte le PGCD de ces pertes unitaires, mais un PGCD réel et non plus entier. On peut encore le calculer par l'algorithme d'Euclide qui fonctionne toujours. La seule difficulté est qu'un ordinateur a une précision parfaite sur les entiers (compris jusqu'à 4.3 milliards environ, ce qui laisse souvent une marge confortable) mais pas sur les nombres à virgule flottante.

Ainsi de temps en temps on peut obtenir que 5,0 - 4,0= 0.999999999 par exemple, ce qui fausse ensuite les calculs de PGCD. Il ne faut donc pas comparer en égalité un nombre flottant à 0, mais plutôt écrire If (abs(x) $\leq$ 0.00000001) then .... C'est une manière de faire fonctionner l'algorithme d'Euclide pour calculer le nombre u.

Etant maintenant assurés que tous les défauts sont un multiple de u, on sait que $\exists N \epsilon \mathbb{N} / \sum_{i=1}^{n} N_i.(1 - R_i) = N.u$

Il faut construire une suite de "matrices" M_i indexées sur Z, t, et $\{1, ..., N\}$ tel que $M_i(z, j, t) = P_{i\ émetteurs}(L_t = j.u|Z = z)$.

Il ne sert à rien de stocker tous les tableaux tri-dimensionnels M_i, juste le dernier suffit, puisque pour calculer M_{i+1} on a juste besoin de M_i.

On comprend que cet algorithme consomme immanquablement du temps, quoique l'on fasse pour l'optimiser : il faut "discrétiser $\mathbb{R}$" (pour la variable aléatoire Z qui suit une loi gaussienne), discrétiser [0, T] pour connaitre les probabilités de défaut à chaque instant t afin de calculer Fixed Leg et Float Leg. Il faut enfin faire ceci autant de fois qu'il y a de noms dans le CDO. On peut faire varier Z entre -5 et 5 disons, étant donné la densité de probabilité de la loi gaussienne.

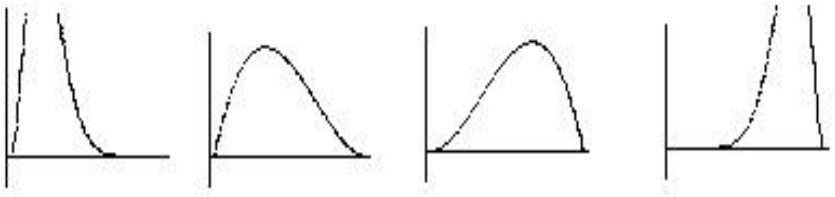


FIG. 7 – loi Beta pour différentes valeurs de a et b. Successivement (a, b) = (3, 10), (2, 3), (3, 2), (10, 3)

Une fois ces matrices calculées, on connait $P(L_t = j.u|Z = z)$ pour tout j, z et t, probabilité sur le portefeuille total. En intégrant sur z, on obtient $P(L_t = j.u)$ pour tout t et j.

Une fois les probabilités calculées, on peut calculer la FloatLeg et la FixedLeg avec les "formules" que l'on a. (on a en fait besoin de calculer $E^Q(L_t^{[K_1,K_2]})$). Le temps de calcul est similaire à MonteCarlo (pour une précision équivalente) et il est encourageant de voir apparaître les mêmes résultats (au pourcent près) donnés par ces 2 méthodes.

L'inconvénient de cette méthode par récurrence est qu'elle est beaucoup moins flexible que Monte Carlo : si un trader désire un résultat rapidement (10 secondes) il peut appeler le pricing par MonteCarlo avec un nombre de simulations faible. Ce n'est pas vraiment le cas avec le pricing par récurrence. (même si l'on peut jouer sur le pas de l'intégrale sur Z, et la précision sur les instants t aussi, dans la grille que l'on bâtit)

Par loi Beta _ La programmation par "loi Beta" : on peut supposer que lorsque l'on a un nombre élevé de CDS dans le panier, la loi de perte cumulée suit une loi Beta(a_t, b_t).

En effet, quand t est petit, il y a une grande probabilité pour qu'il n'y ait eu encore aucun défaut, puis au fur et à mesure que le temps augmente, il y a de plus en plus de défauts (aussi dû à la corrélation intrinsèque des défauts). On règle les paramètres a_t, b_t de telle sorte que espérance et variance d'une loi Beta(a_t, b_t) égalisent espérance et variance de L_t.

On en déduit alors $E^Q(L_t^{[K_1,K_2]})$ (en les calquant sur la loi Beta).

Implémentation :

Soit $l_t = \frac{L_t}{\sum_{i=1}^n L_i} = \frac{L_t}{\sum_{i=1}^n N_i(1-R_i)}$. On rappelle que L_t correspond à la perte courante à l'instant t. l_t est simplement la normalisée de la perte : $l_t \epsilon [0, 1]$.

La loi beta de paramètres a et b est une loi continue sur [0, 1], de densité $f_{a,b}(x) = \frac{x^{a-1}(1-x)^{b-1}}{B(a,b)}.1_{0\leq x\leq 1}$ où $B(a, b) = \int_0^1 u^{a-1}(1-u)^{b-1}du$

Elle est définie pour $a, b > 0$ et a pour moyenne et variance :

$\mu = \frac{a}{a+b}$ et $\sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}$.

Preuve En admettant que $B(a,b) = \frac{\Gamma(a).\Gamma(b)}{\Gamma(a+b)}$ où $\Gamma(x)$ est la fonction Gamma usuelle, qui vérifie $\Gamma(x+1) = x\Gamma(x)$, on a :
$\mu = \frac{1}{B(a,b)} \int_0^1 u.u^{a-1}(1-u)^{b-1}du = \frac{1}{B(a,b)} \int_0^1 u^a(1-u)^{b-1}du$
$= \frac{B(a+1,b)}{B(a,b)} = \frac{\frac{\Gamma(a+1).\Gamma(b)}{\Gamma(a+b+1)}}{\frac{\Gamma(a).\Gamma(b)}{\Gamma(a+b)}}$
$= \frac{\Gamma(a+1).\Gamma(a+b)}{\Gamma(a).\Gamma(a+b+1)} = \frac{a}{a+b}$

De même $\sigma^2 = \frac{B(a+2,b)}{B(a,b)} - \mu^2$
$= \frac{\frac{\Gamma(a+2).\Gamma(b)}{\Gamma(a+b+2)}}{\frac{\Gamma(a).\Gamma(b)}{\Gamma(a+b)}} - (\frac{a}{a+b})^2$
$= \frac{\Gamma(a+2).\Gamma(a+b)}{\Gamma(a).\Gamma(a+b+2)} - (\frac{a}{a+b})^2$
$= \frac{a(a+1)}{(a+b)(a+b+1)} - \frac{a^2}{(a+b)^2}$
$= \frac{a}{a+b}(\frac{a+1}{a+b+1} - \frac{a}{a+b}) = \frac{ab}{(a+b)^2(a+b+1)}$ ∎

Concernant le calibrage de a_t, b_t pour tout t, il nous faut calculer la moyenne et la variance de L_t afin d'identifier a_t et b_t.

$E(l_t) = \frac{E(L_t)}{\sum_{i=1}^n N_i(1-R_i)} = \frac{\sum_{i=1}^n (1-R_i).N_i.p_i(t)}{\sum_{i=1}^n N_i(1-R_i)}$. Il n'y a pas lieu de se soucier ici de la corrélation contrairement au cas de l'espérance.

$Var(l_t) = E(l_t^2) - E(l_t)^2$
$= \frac{E(\sum_{i=1}^n (1-R_i).N_i.1_{\tau_i \leq t})^2}{(\sum_{i=1}^n N_i(1-R_i))^2} - \frac{(\sum_{i=1}^n (1-R_i).N_i.p_i(t))^2}{(\sum_{i=1}^n N_i(1-R_i))^2}$
$= \frac{1}{(\sum_{i=1}^n L_i)^2}(\sum_{i\sim j} L_iL_jP(\tau_i \leq t, \tau_j \leq t) + \sum_i L_i^2.(p_i(t) - p_i^2(t)))$
$= \frac{1}{(\sum_{i=1}^n L_i)^2}(\sum_{i\sim j} L_iL_jN_2(N^{-1}(p_i(t)), N^{-1}(p_j(t)), \rho_{ij}) + \sum_i L_i^2.(p_i(t) - p_i^2(t)))$

On connait les lois $p_i(t)$ puisque ce sont des lois de Poisson de paramètre λ_t pris en escalier et déduit par bootstrapping des prix des CDS. (cf la part sur les CDS)

D'autre part il y a l'existence de $c_i(t)$ telle que $P(\tau_i \leq t) = P(X_i <= c_i(t))$ et donc $P(\tau_i \leq t, \tau_j \leq t) = P(X_i <= c_i(t), X_j <= c_j(t))$

Comme les X_i sont des variables gaussiennes centrées réduites corrélées, on a bien $P(\tau_i \leq t, \tau_j \leq t) = N_2(N^{-1}(p_i(t)), N^{-1}(p_j(t)), \rho_{ij})$

où $N_2(a,b,\rho) = P(X_1 < a, X_2 < b)$ sachant que X_1 et X_2 sont gaussiennes centrées réduites de corrélation ρ.

On a donc accès, numériquement, à $E(l_t)$ et $Var(l_t)$**.** On note ces variables μ_t et σ_t^2.

Il faut maintenant résoudre le système :

$$\left\{ \begin{array}{c} \mu_t = \frac{a_t}{a_t+b_t} \\ \sigma_t^2 = \frac{a_t.b_t}{(a_t+b_t)^2(a_t+b_t+1)} \end{array} \right\}$$

En notant $\chi_t = \frac{\mu_t(1-\mu_t)}{\sigma_t^2} - 1$ on peut vérifier que $\left\{ \begin{array}{c} a_t = \mu_t.\chi_t \\ b_t = (1-\mu_t)\chi_t \end{array} \right\}$ convient.

Les résultats sont similaires à ceux donnés par récurrence et Monte Carlo pour de faibles strikes, mais pas sur de plus grands strikes.

Le temps de calcul est similaire aussi (de l'ordre d'une minute). Encore une fois on peut gagner en temps en diminuant la précision lors des calculs d'intégrale par exemple.

Ces implémentations de pricers ainsi que des lectures d'articles ont permis de comprendre le fonctionnement du marché des CDS et des CDO. Il fallait ensuite comparer les pricers créés avec celui utilisé par la Société Générale (Marx). Les résultats furent concordants.

Remarque : On notera encore une fois la difficulté d'implémenter certaines fonctions en informatique. (exemple : $N^2(x, y, \rho) = P(X <= x, Y <= y)$ où X et Y sont gaussiennes centrées de corrélation ρ (cf Appendice pour l'implémentation de cette fonction) ; ou encore la loi Beta : $\int_0^x u^{a-1}(1-u)^{b-1}du$)

_ les pricers ont été implémentés sur Excel en VBA. C'est cet environnement qui a été choisi, en premier lieu, pour sa facilité à lire les données du marché d'Excel. D'autre part Excel implémente déjà la loi Beta ce qui est pour le moins providentiel.

_ Enfin, il n'est pas forcément facile de créer un pricer qui s'adapte à la taille du marché. Dans toutes nos simulations d'essais il y avait un nombre fixé (10) de CDS dans le panier. Mais il faut donc rentrer 10 lignes de données CDS (pour différentes maturités) afin d'avoir accès à la loi λ(t). Il faut aussi rentrer 10 nominaux, 10 taux de recovery, ...

et tout ceci prend de la place sur la feuille et n'est pas aisément paramétrable.

2.4 Base correlation - Compound correlation

Grâce au modèle de la copule gaussienne on peut impliciter la corrélation utilisée pour pricer n'importe quelle tranche côtée du marché. Il y a donc une corrélation implicitée pour la tranche [0, 3%], [3, 6%], ... Si l'on souhaite pricer une tranche [2, 4%] par exemple, on a besoin de connaître la corrélation sur cette tranche. On peut alors interpoler les corrélations des tranches [0, 3%] et [3, 6%] par exemple. C'est cette manière de faire qui a été utilisée pendant un certain temps jusqu'à ce qu'on se rende compte qu'elle n'était pas très pratique : en effet, ce n'a pas trop de sens que de moyenner des corrélations sur 2 tranches différentes. Cela ne donne pas des résultats très précis. Et d'autre part il y a beaucoup de tranches différentes de type [x, y%].

A la compound correlation est ainsi préférée la base correlation. Celle-ci dénote la correlation que l'on utiliserait sur une tranche [0, x%]. On a ainsi une dimension en moins sans perte de généralités (connaitre les prix des CDO sur toutes les tranches [0, x%] permet de pricer n'importe quel [x, y%]). Et d'autre part il y a plus de sens à interpoler entre deux tranches [0, a%] et [0, b%] connues afin de déterminer la corrélation sur [0, x%].

Il existe un dernier type de corrélation : la local correlation. Celle-ci correspond à la compound correlation sur des tranches très fines [x, x+dx%]. Celle-ci a aussi un sens puisqu'elle a naturellement vocation à être continue. Cependant on ne peut pas l'impliciter des données de marché, et d'autre part la connaître ne permet pas de pricer aussi simplement les produits que lorsque l'on connait la base correlation par exemple. Cependant étant donnée la base-correlation on en déduit facilement la local correlation. On peut ainsi vérifier qu'une interpolation donnée sur la base correlation produit une local correlation sans "surprises", ce qui fournit un critère supplémentaire.

Deuxième partie

Interpolation du smile de corrélation

3 Contraintes

3.1 Condition de non arbitrage

Rappelons qu'il nous faut trouver une manière d'interpoler le smile de corrélation de façon à ce qu'il n'y ait pas d'arbitrage possible.

Il nous faut donc comprendre dans un premier temps où peut se trouver l'arbitrage sur le spread afin de chercher à l'éliminer.

Une première idée fût de penser que le spread sur la tranche [0, K] était une fonction décroissante de K, puisque l'on prend a priori plus de risque sur les tranches les plus basses.

Mais il s'avère que ceci est faux : si la corrélation est très élevée, on peut prendre plus de risque en assurant une tranche equity qu'une tranche senior.

Par contre, si on s'intéresse à des tranches de largeur fixe, le résultat devient vrai :

Proposition II.1 *Pour des raisons d'arbitrage,* ***la courbe de spread sur les tranches [K, K+dK] doit être décroissante et strictement positive****.*

Preuve Le caractère positif de la courbe de spread est naturel : on vend de la protection sur une tranche, c'est à dire que l'on reçoit un spread régulièrement jusqu'à ce qu'il y ait un défaut (ou pas) et auquel cas on paye la prime d'assurance (positive). Si le spread était négatif on n'aurait aucune raison de vouloir assurer le "client", car on serait perdant dans tous les cas.

_ Concernant la décroissance : Considérons a, b et δ tels que $a, b, \delta > 0$, $a + \delta < b$ et $b + \delta \leq 1$.

Supposons que le spread sur $[a, a+\delta]$ soit inférieur à celui ci sur $[b, b+\delta]$ ($s_{a,\delta} < s_{b,\delta}$)

Stratégie d'arbitrage : On achète de la protection sur $[a, a + \delta]$ et on vend de la protection sur $[b, b + \delta]$.

On considère plusieurs cas, suivant le montant accumulé des défauts :

Si $L_t < a$, on touche à chaque date de paiement $s_{b,\delta} - s_{a,\delta} > 0$.

Si $a \leq L_t \leq a + \delta$, on touche à chaque date de paiement $s_{b,\delta} - \frac{a+\delta-L_t}{\delta}.s_{a,\delta} > 0$.

Si $a + \delta < L_t \leq b$, on touche à chaque date de paiement $s_{b,\delta} > 0$.

Si $b < L_t \leq b + \delta$, on touche à chaque date de paiement $\frac{b+\delta-L_t}{\delta}.s_{b,\delta} > 0$.

Sinon on ne touche rien.

On est donc gagnant dans tous les cas sur les montants des spreads perçus. Il faut maintenant vérifier que l'on ne risque pas de payer de prime. On s'intéresse cette fois à L_T.

Si $L_T < a$, on ne paye ni ne touche aucune prime.

Si $a \leq L_T \leq a + \delta$, on nous payera la jambe de prime $(L_T - a).N > 0$ car on est **acheteur de protection** sur $[a, a + \delta]$.

Si $a + \delta < L_T \leq b$, on nous a payé la jambe de prime sur $[a, a + \delta]$ $(= \delta.N > 0)$

Si $b < L_T \leq b+\delta$, on nous a payé la jambe de prime sur $[a, a+\delta]$ $(= \delta.N)$ mais on paye celle sur $[b, L_T]$ (soit $(L_T - b).N$) donc au final on touche $(\delta - (L_T - b)).N > 0$

Sinon on touche la prime sur $[a, a + \delta]$ mais on paye la prime sur $[b, b + \delta]$ et elles se compensent exactement.

Au final, si $s_{a,\delta} < s_{b,\delta}$ il y a une possibilité de réaliser un arbitrage.

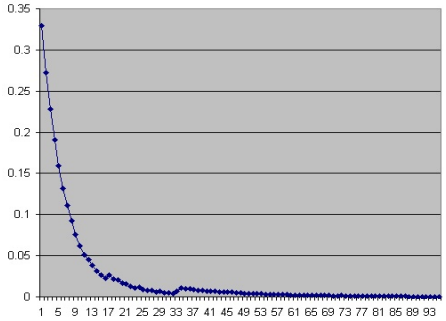


FIG. 8 – Spread sur tranchette, calculé par Marx. Des arbitrages sont possibles aux alentours de 16% et 33%.

Le spread sur des tranches très fines de même taille est donc décroissant. ∎

Exemple : Afin de travailler sur "quelque chose de solide" et d'avoir des résultats tangibles, Marx a été préféré : il y a certainement moins de bugs dans le pricer de la Société Générale, qui est utilisé depuis des années, que dans le notre !

Pour un marché donné (iTrax4) avec environ 120 noms dans le panier, ainsi que les corrélations données par le marché, MarxExcel a calculé les spread, fixedleg et floatleg sur toute une étendue de strikes, avec un faible pas.

Ceci mène à la problématique du stage : comment interpoler le smile de corrélation afin de rendre la courbe de spread non arbitrable ?

3.2 Arbitrage théorique : Market Law of Losses

Une deuxième contrainte de non-arbitrage, théorique cette fois, est apportée par l'article [2]. Il est fait mention de la "Market law of Losses", c'est à dire "étant donnés les prix du marché, quelle loi pour les pertes (en probabilités) peut-on en déduire ?".

$$P(Loss < K) = P(Loss < K, \rho_K) - Skew(K, \rho_K) * Rho(K, \rho_K)$$

Notation : $P(Loss < K, \rho)$ est la probabilité calculée grâce à la copule gaussienne.
$Skew(K, \rho_K) = \frac{\partial \rho_K}{\partial K}$ représente la sensibilité de ρ par rapport à K.
Enfin, $Rho(K, \rho) = -\int_0^K \frac{\partial L}{\partial \rho}(x, \rho)dx$.

Remarque : Cette formule est démontrée en appendice.

$P(Loss < K)$ doit être croissante, mais rien ne semble indiquer mathématiquement parlant que s'il y a des décroissances locales de cette fonction, il y a de petits arbitrages à faire sur le spread. Mais pourtant c'est ce qui se passe : sur ce graphique on a

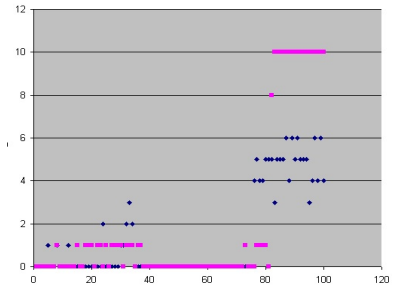


FIG. 9 – en bleu : arbitrage local sur le spread ; en rouge : décroissance locale de la Market Law of Losses.

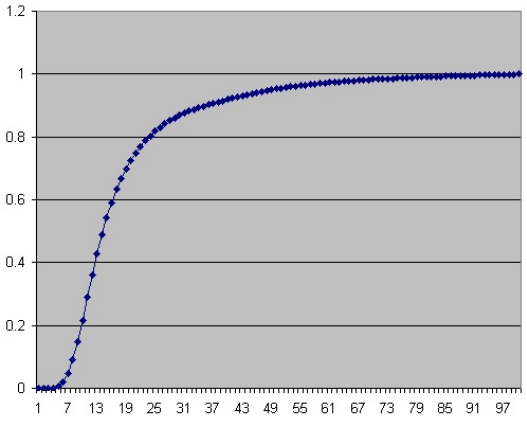


FIG. 10 – Probabilité de marché donnée par l'interpolation linéaire sur la corrélation

comparé (pour l'interpolation linéaire) les décroissances des tranches de spread, et les décroissances locales de la Market Law of Losses.

On a en fait subdivisé [0, 100%] en intervalles de taille 1%, et on a cherché des décroissances locales et des arbitrages locaux tous les 0.1%. On compte ensuite sur chaque intervalle de taille 1% combien il y a de "problèmes", soit un nombre entier entre 0 et 10.

Ceci a été fait pour mieux rendre compte des coïncidences entre l'arbitrage local et la décroissance locale de la Market Law of Losses : sinon, s'il y a quelque chose sur l'arbitrage pur en 5.6% et rien en 5.7% et inversement sur la loi, cela ne serait pas compté alors que ça "mériterait" de l'être. En quelque sorte cela permet d'introduire une "tolérance à la coïncidence" entre les deux lois.

On se rend compte qu'il y a une corrélation plus qu'heureuse entre les arbitrages locaux sur le spread et sur la loi de probabilité.

La courbe est très lisse et croissante. Elle vérifie aussi $\lim_{K\to 0\%} P(k,\rho_k) = 0$ ainsi que $\lim_{K\to 100\%} P(k,\rho_k) = 0$.

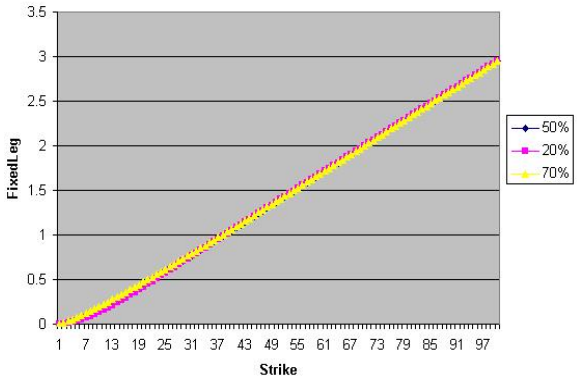


FIG. 11 – Fixed Leg en fonction de la corrélation

Certains articles demandent à ce que la courbe soit concave en justifiant par le fait que $P(L \leq K)$ est qui doit naturellement être croissante, a pour dérivée $dP = P(K \leq L \leq K + dK)$ qui est décroissante. Il n'y a aucune raison à cela puisque cette probabilité doit dépendre aussi de la corrélation (dépendante de K).

3.3 Contrainte Mathématique

Rappel : on avait
$FloatLeg = E^Q(\int_0^T B_t.dL_t^{[K_1,K_2]})$

$$FixedLeg = E^Q(s * \sum_{i=1}^{n} B_{t_i} * \int_{t_{i-1}}^{t_i} (K_2 - K_1 - L_t^{[K_1,K_2]})dt)$$

et donc $fairspread_{[0,K]} = \frac{FloatLeg}{FixedLeg(s)/s} = \frac{E^Q(\int_0^T B_t.dL_t^K)}{E^Q(s*\sum_{i=1}^{n} B_{t_i} * \int_{t_{i-1}}^{t_i} (K - L_t^K)dt)}$ c'est à dire $\frac{e^{-r.T}E^Q L_T^K + \int_0^T r.e^{-r.t}E^Q(L_t^K)dt}{E^Q(\sum_{i=1}^{n} B_{t_i} * \int_{t_{i-1}}^{t_i} (K - L_t^K)dt)}$

(on a intégré par partie l'intégrande de la FloatLeg dL_t qui est en fait un "peigne de Dirac" stochastique, et on a supposé que le zéro coupon s'écrit $B_t = e^{-r.t}$)

En estimant que $r \approx 0, FloatLeg \approx E^Q L_T^K$.

Comme on peut le remarquer, la FixedLeg ne dépend que très peu de la corrélation ce qui peut sembler curieux au premier abord. Une explication possible est que s'il y a un défaut qui intervient, a priori c'est plutôt vers la fin de la maturité qu'au début, de sorte qu'il y a quand même du spread qui a été reçu par le vendeur de protection. Une autre explication est que la Fixed Leg est calculée sur des tranches "equity" c'est

à dire de la forme [0-K%]. Ainsi comme les tranches basses ne sont pas sensibles de la même manière à la corrélation que les tranches plus hautes, sur une tranche equity une sorte de nivellement se crée sur la Fixed Leg.

Toujours est-il que d'après ces courbes, on peut naturellement poser $FixedLeg \approx \alpha.K$ du moins pour un strike au dela de 10%.
Finalement, $spread_{[0-K\%]} \approx \frac{E^Q L_T^K}{\alpha.K}$.

La grandeur plus qui nous intéresse plus est, nous l'avons dit, le spread sur tranchelette, soit $spread_{[K-K+dK]} = \frac{\partial FloatLeg_K}{\partial FixedLeg_K}$.

Avec les approximations effectuées, on obtient $spread_{[K-K+dK]} \approx \beta.\partial_K(E^Q L_T^K)$

Nous nous rapprochons d'une contrainte purement formelle sur le smile de corrélation. Il nous faudrait pour cela connaître une formule fermée pour L_T^K. Ceci n'est pas une tâche aisée, et nous allons avoir recours à plusieurs hypothèses et approximations :

Sans aucune hypothèse, en notant pour le crédit j : 0 pour l'événement "pas de défaut avant la maturité" de probabilité $p_j(z)$ et 1 pour l'événement contraire, on a :

$$E\ L_T^K = \sum_{(i_1,...,i_N)\epsilon\{0,1\}^N} P(i_1,...i_N) \sum_{j=1}^{N} L_j * i_j$$

$$= \sum_{(i_1,...,i_N)\epsilon\{0,1\}^N} \int_{-\infty}^{+\infty} \prod_{j=1}^{n} p_{i_j}^{1-i_j}(z)(1-p_{i_j}(z))^{i_j}.\phi(z).dz \sum_{j=1}^{N} L_j * i_j$$

Faisons tout d'abord l'hypothèse d'"homogénéité" : nous considérons que les noms du paniers sont "homogènes", c'est à dire qu'ils ont tous le même taux de recouvrement, la même fonction de probabilité de défaut et le même nominal. Cette "hypothèse" a déjà été utilisée, afin de considérer dans nos optimisations numériques qu'un panier "standard" était suffisament représentatif, et qu'une bonne méthode d'interpolation sur le smile pour ce panier serait aussi bonne sur n'importe quel autre panier (iTraxx, ...)

Alors on peut cette fois-ci sommer sur le nombre de défauts dans le panier.

Cela donne $E\ L_T^K = \sum_{(i_1,...,i_N)\epsilon\{0,1\}^N} P(i_1,...i_N) \sum_{j=1}^{N} L_j * i_j$

$$= \sum_{k=0}^{N} P(k\ defauts) * L * k$$

$$= L * \sum_{k=0}^{N} k.P(k\ defauts)$$

$$= L * \sum_{k=0}^{N} k.C_N^k. \int_{-\infty}^{+\infty} p^{N-k_j}(z)(1-p(z))^k.\phi(z).dz$$

C'est cette formule qui a été utilisée pour calculer les probabilités de défaut dans notre pricer.

Mais rappelons que l'on cherche à obtenir une formule fermée. Etant donné l'intrication des probabilités due à la corrélation, nous sommes contraints de faire une hypothèse plus forte encore que celle d'"homogénéité" : celle de "large-pool".

Nous faisons donc l'hypothèse de "large-pool", c'est à dire que l'on considère qu'il y a N noms dans le panier, tous "homogènes", de probabilité de défaut $p(t)$ et que $N \gg 1$.

Nous rappelons que le défaut est caractérisé par le fait que $X_i = Z.\rho + \epsilon_i.\sqrt{1-\rho^2}$, qui suit une loi normale centrée réduite, passe en dessous de la "barre" $N^{-1}(p(t))$ de telle sorte que la probabilité soit effectivement de $p(t)$.

De fait, nous avons, pour la loi de la probabilité de perte totale cumulée :
$Q(L < K) = Q\{(1-R).\frac{1}{N}\sum_{j=1}^{N} 1_{\{Z.\rho+\epsilon_j.\sqrt{1-\rho^2} \leq N^{-1}(p(t))\}} < K\}$

Remarque : R désigne le taux de recouvrement, pris à 40% dans la pratique.

Quand N est très grand, $\frac{1}{N}\sum_{j=1}^{N} 1_{\{Z.\rho+\epsilon_j.\sqrt{1-\rho^2} \leq N^{-1}(p(t))\}}$ tend "intuitivement" vers la $P(Z.\rho + \epsilon_j.\sqrt{1-\rho^2} \leq N^{-1}(p(t)))$.

Cela est vrai conditionnellement à Z car on a alors une somme de variables aléatoires indépendantes et identiquement distribuées. $Q(L < K)$ vaut alors 0 ou 1 selon que les probabilités de défaut conditionnées à Z sont faibles ou fortes.

Ainsi $Q(L < K \mid Z) = 1_{N(\frac{N^{-1}(p(t))-Z.\rho}{\sqrt{1-\rho^2}}) < \frac{K}{1-R}}$

$= 1_{-Z < \frac{\sqrt{1-\rho^2}N^{-1}(\frac{K}{1-R})-N^{-1}(p(t))}{\rho}}$

et donc comme Z et -Z ont même loi,

$Q(L < K) = N(\frac{\sqrt{1-\rho^2}N^{-1}(\frac{K}{1-R})-N^{-1}(p(t))}{\rho})$

Enfin, on peut prouver que $E^Q\ L_T^K = \int_0^K Q(K > L_T > x).dx$.

En effet, $E^Q\ L_T^K = \int_0^K x.Q(L_T^K \epsilon [x, x+dx] = \int_0^K \int_0^x Q(L_T^K \epsilon [x, x+dx])dudx = \int_0^K (\int_u^K Q(L_T^K \epsilon [x, x+dx]))du = \int_0^K Q(L_T^K \epsilon [u, K])du = \int_0^K Q(K > L_T^K > u)du$

Cette espérance dépend de manière cachée de ρ mais on a $\frac{\partial E^Q L_T^K}{\partial K} = K.\frac{dQ(L_T > K)}{dK}$.

On peut donc dire en première approximation que
$FloatLeg \approx E^Q\ L_T^K = \int_0^K Q(L_T^K > x).dx$ et donc que

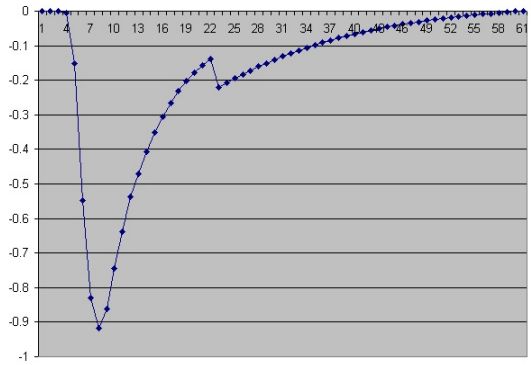


FIG. 12 – Premier terme : $K.\frac{dQ(L_T>K)}{dK}$

$$\frac{\partial FloatLeg}{\partial K} \approx \frac{\partial E^Q L_T^{K,\rho(K)}}{\partial K} = K.\frac{dQ(L_T>K)}{dK} + \frac{d\rho}{dK}.\frac{\partial E^Q L_T^{K,\rho}}{\partial \rho}$$
$$\text{soit } \frac{\partial FloatLeg}{\partial K} \approx K.\frac{dQ(L_T>K)}{dK} + \frac{d\rho}{dK}.\frac{\partial E^Q L_T^{K,\rho}}{\partial \rho}$$
$$\approx K.\frac{dQ(L_T>K)}{dK} + \frac{d\rho}{dK}.\int_0^K \frac{\partial Q(L_{T,\rho}^K>x)}{\partial \rho} dx$$

L'idée est d'espérer que le premier terme soit numériquement plus fort que le second, et ainsi d'obtenir en première approximation une contrainte purement formelle sur $\frac{d\rho}{dK}$, car $\frac{dQ(L_T>K)}{dK}$ est connu.

On rappelle que **la contrainte principale est que la dérivée de la jambe de prime (la float leg) soit décroissante.**

Le premier terme, $K.\frac{dQ(L_T>K)}{dK}$, est négatif. Mais la jambe de prime est croissante en le strike, et donc la somme des 2 termes est positive. On peut donc en déduire que le second est plus gros que le premier, mais encore rien pour l'instant ne peut donner d'indications sur les variations de ces termes : après tout, c'est cela qui compte.

Etant donné que l'on n'a pas de formule pour le second terme (on connait l'intégrande, mais pas l'intégrale), nous sommes obligés d'avoir recours au calcul numérique.

Résultats :

Il arrive que la théorie est peu en concordance avec la pratique numérique, et c'est ce qui se passe ici. Les intégrales numériques de dérivées partielles d'approximations ne donnent pas un résultat concordant : la FloatLeg est bien croissante, et a une belle allure concave. Sa dérivée devrait être positive. Cependant la somme des deux termes que nous étudions donne numériquement un résultat négatif, si bien que l'on ne peut pas conclure.

Le second terme semble en effet plus ératique que le premier.

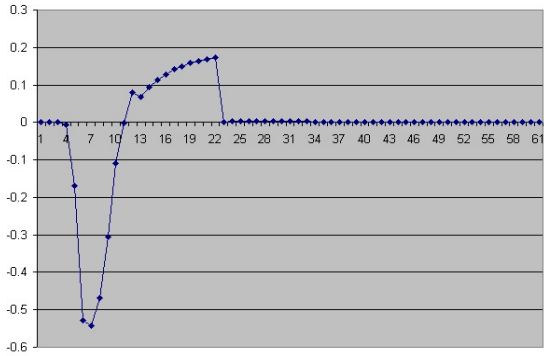


FIG. 13 – Second terme : $\frac{d\rho}{dK}.\int_0^K \frac{\partial Q(L^K_{T,\rho}>x)}{\partial \rho}dx$

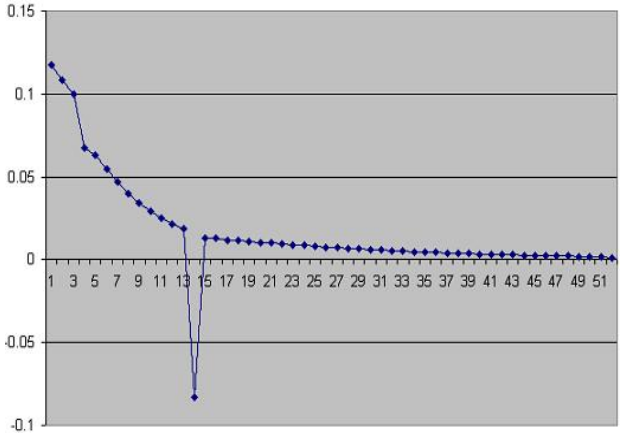


FIG. 14 – dérivée du premier terme

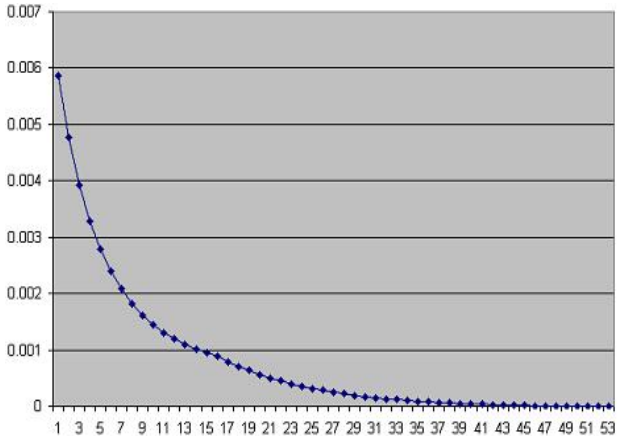


FIG. 15 – dérivée de la Float Leg

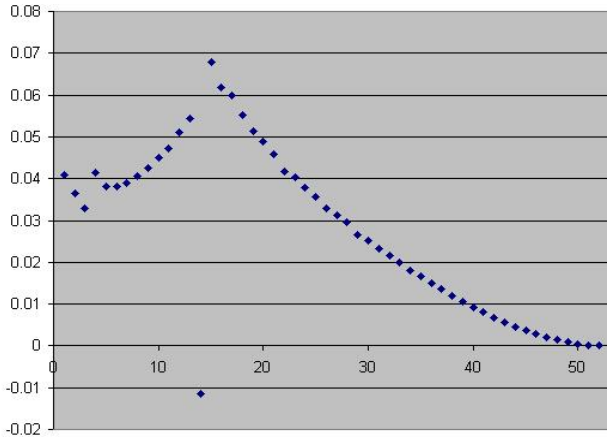


FIG. 16 – Ratio des dérivées

Lorsque l'on trace la dérivée du premier terme, et celle de la floatleg puis leur ratio, on obtient ces graphes(coupés au dessous de 7% de strike)

Dans une certaine mesure, on peut estimer que l'on a donc

$$boxed\frac{\partial FloatLeg}{\partial K} \approx \alpha \frac{d(K.\frac{dQ(L_T>K)}{dK})}{dK}$$

et la loi de Q est donnée par

$$boxedQ(L<K) = N(\frac{\sqrt{1-\rho^2}N^{-1}(\frac{K}{1-R}) - N^{-1}(p(t))}{\rho})$$

On obtient donc la contrainte formelle suivante sur $\rho(K)$:

$$boxed\frac{d^2(K.\frac{dQ(L_T>K)}{dK})}{dK^2} < \mathbf{0}$$

Pour résumer cette partie, on a vu qu'une contrainte d'absence d'arbitrage est que le spread sur les tranchettes doit être décroissant.

Or spread sur tranchette $= spread_{[K-K+dK]} = \frac{\partial FloatLeg_K}{\partial FixedLeg_K}$.

et par ailleurs la jambe de prime (fixedLeg) est quasi linéaire ce qui donne spread sur tranchette $= spread_{[K-K+dK]} \approx \alpha.\partial FloatLeg_K$ qui doit être décroissant, et donc $\partial FloatLeg_K$ doit être décroissant.

Ce dernier terme est la somme de $K.\frac{dQ(L_T>K)}{dK}$ et de $\frac{d\rho}{dK}.\int_0^K \frac{\partial Q(L^K_{T,\rho}>x)}{\partial \rho}dx$

Mais cette dernière intégrale est numériquement parlant très instable Mais ses variations sont faibles par rapport à celles du premier terme, et finalement on a encore une fois : $\frac{\partial^2 FloatLeg_K}{\partial^2 K} \approx \beta.\frac{d(K.\frac{dQ(L_T>K)}{dK})}{dK}$ d'où le résultat final approché :

la condition de non-arbitrage se réduit à $\frac{d(K.\frac{dQ(L_T>K)}{dK})}{dK} < 0$ (sachant que la loi de Q est connue explicitement dans notre modèle).

On a donc enfin une condition purement formelle.

Malheureusement celle-ci est trop délicate pour être utilisable en pratique :

$\frac{d(K.\frac{dQ(L_T>K)}{dK})}{dK} < 0$

se réduit à $K.\frac{d^2(Q(L_T>K))}{dK^2} + \frac{dQ(L_T>K)}{dK} < 0$

où $Q(L<K) = N(\frac{\sqrt{1-\rho^2}N^{-1}(\frac{K}{1-R})-N^{-1}(p(t))}{\rho}) = N(\frac{\sqrt{1-\rho^2}a_K-b}{\rho})$

soit $\frac{dQ(L_T>K)}{dK} = \phi(\frac{\sqrt{1-\rho^2}a_K-b}{\rho}).(\frac{\rho'}{\rho^2}(b - \frac{a}{\sqrt{1-\rho^2}}) - \frac{\sqrt{1-\rho^2}}{\rho}.\frac{1}{\phi(N^{-1}(\frac{K}{1-R}))})$ où $N(x)$ est la fonction de répartition de la loi gaussienne et $\phi(x)$ sa dérivée ($\phi(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$)

3.4 Mise en oeuvre

A priori, pour chaque CDO donné (et ses points de corrélations donnés par le marché) il existe au moins une manière de chercher une "bonne interpolation". Seulement, ceci n'est pas compatible avec Marx et avec la réalité : on ne peut pas faire de chaque cas un cas particulier, et calculer la meilleure interpolation possible. Une raison est que cela prendrait vraiment trop de temps a priori, temps que les traders n'ont pas.

Par contre, on peut faire tourner l'ordinateur 4 heures de suite une bonne fois pour toutes pour calculer la meilleure interpolation possible pour un CDO "type", enregistrer le résultat, et appliquer cette interpolation à tous les CDO (donc aucun temps de calcul).

L'idée retenue est donc de calculer plusieurs courbes d'interpolation, et de chercher une fonction qui ressemble le plus aux courbes obtenues (de la même manière que l'interpolation linéaire ne dépend que des points d'entrée).

L'idée de l'optimisation consiste à trouver une fonction d'interpolation qui passe par les points donnés par le marché, qui soit aussi lisse que possible, et qui minimise les possibilités d'arbitrage sur la courbe de spread.

On peut même se demander s'il existe une interpolation permettant de rendre la courbe non interpolable. Auquel cas faudrait-il trouver la meilleure interpolation possible au sens "minimiser le potentiel d'arbitrage de la courbe de spread" (à un sens qui reste à définir).

Le pricer Marx étant assez lent, nous avons décidé d'enregistrer les résultas obtenus après une nuit de calculs dans un fichier texte, que l'on charge dans la mémoire vive afin de pouvoir se concentrer sur l'interpolation une bonne fois pour toutes, et non plus sur les calculs de prix.

3.5 Données

Pour le moment, le marché "donne" les corrélations sur quelques tranches ([0%-3%], [0%-6%], [0%-9%], [0%-12%], [0%-22%], ...) et l'interpolation est faite linéairement ou quadratiquement (de manière à avoir une courbe C^1 au lieu de continue seulement).

Valeurs prises par la corrélation implicite :

Strike	3%	6%	9%	12%	22%
Corrélation	17%	26%	34%	41%	57%

On note cette fois encore dans la croissance de la courbe de légères convexités locales. Ces points d'inflexion sont à l'origine des arbitrages sur la courbe de spread sur tranchelet.

(cf les points autour de 35% de strike) et le saut sur la courbe de spread plus haut.

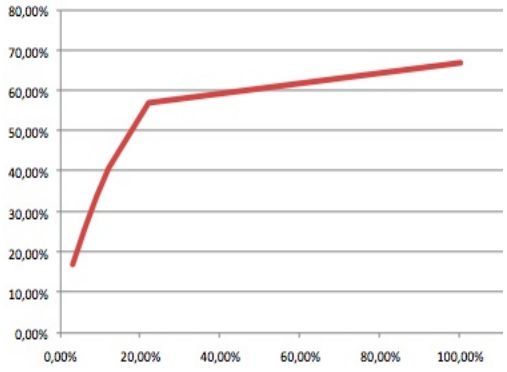


FIG. 17 – Corrélation implicitée du marché

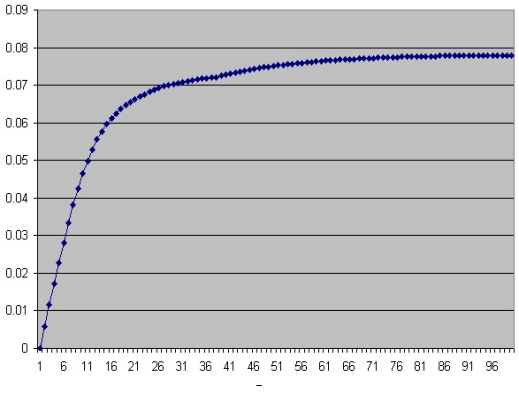


FIG. 18 – Courbe de la Float Leg

4 L'interpolation

L'interpolation s'est faite chronologiquement en plusieurs étapes :

_ initialement l'interpolation s'est basée sur les conditions de non-arbitrage sur la courbe de spread : l'interpolation se faisait purement numériquement par optimisation, sans chercher à lisser le smile de corrélation, en prenant pour point de départ l'o. L'optimisation de fonction sous contraintes est cependant un problème difficile et notre algorithme faisait converger rapidement la suite de fonctions d'interpolation vers une fonction peu lisse et dont le spread final n'avait pas une allure agréable.

Cette première approche a donc été mise de côté en faveur d'une seconde, dont l'idée, eue plus tôt, semblait moins prometteuse mais qui a pourtant porté ses fruits.

_ l'accent s'est porté sur l'interpolation pure, c'est à dire trouver à partir d'un ensemble de points donnés une courbe dont l'allure et le lissage soient les plus convaincants possible.

Ce qui gênait dans cette dernière méthode, c'est qu'il n'y avait plus vraiment de tenant économique au problème. Mais c'est pourtant cette méthode qui réussit le plus : comme on le faisait remarquer dans la partie concernant la "market law of losses" (c'est à dire la loi de pertes sur le portefeuille implicitée par le marché), ce sont les discontinuités sur la fonction de corrélation et sur ses dérivées qui se répercutent en des discontinuités et des irrégularités sur la fonction de spread sur tranchelette (donnant lieu à des opportunités d'arbitrage).

Appliquer notre optimiseur numérique sur une fonction interpolée par les méthodes que l'on a employées n'arrangeait pas vraiment l'allure du spred sur tranchette, bien que l'idée fût tentante.

4.1 Problèmes généraux d'interpolation

L'interpolation est un problème difficile au sens où il existe plusieurs solutions à un seul jeu de données. (il existe en effet une infinité de polynômes passant par n points $(x_1, y_1)...(x_n, y_n)$)

A titre d'exemple :

Ces 3 courbes passent toutes par 3 points donnés. Cependant les tendances qu'elles donnent pour le "futur" sont très différentes les unes des autres, et cela s'en patît sur les énormes écarts entre les 2 derniers points d'interpolation.

Il y a donc une infinité de façon d'interpoler un jeu de données, et c'est un problème subjectif que de déterminer quelle courbe interpole le mieux, lorsqu'il n'y a pas de contraintes.

Afin de comprendre comment réagit l'interpolation vis-à-vis du smile de corrélation nous avons créé un logiciel très pratique, en C#. A la base celui-ci ne devait que calculer le spread et pricer des CDO, puis nous l'avons fait évoluer pour qu'il optimise le smile de corrélation, et progressivement nous avons eu finalement un outil capable de nous aider dans la compréhension de notre problème.

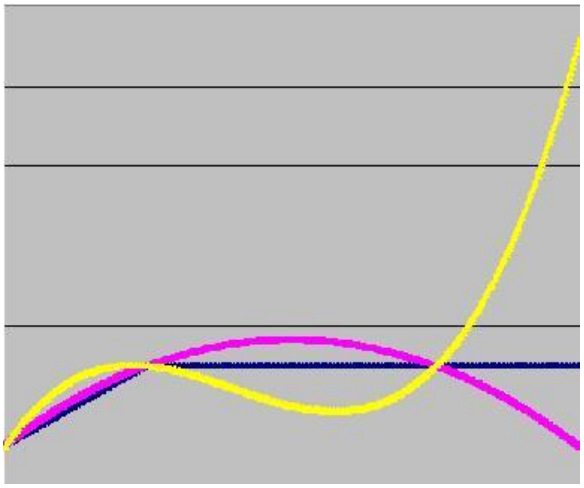


FIG. 19 – exemple d'interpolation

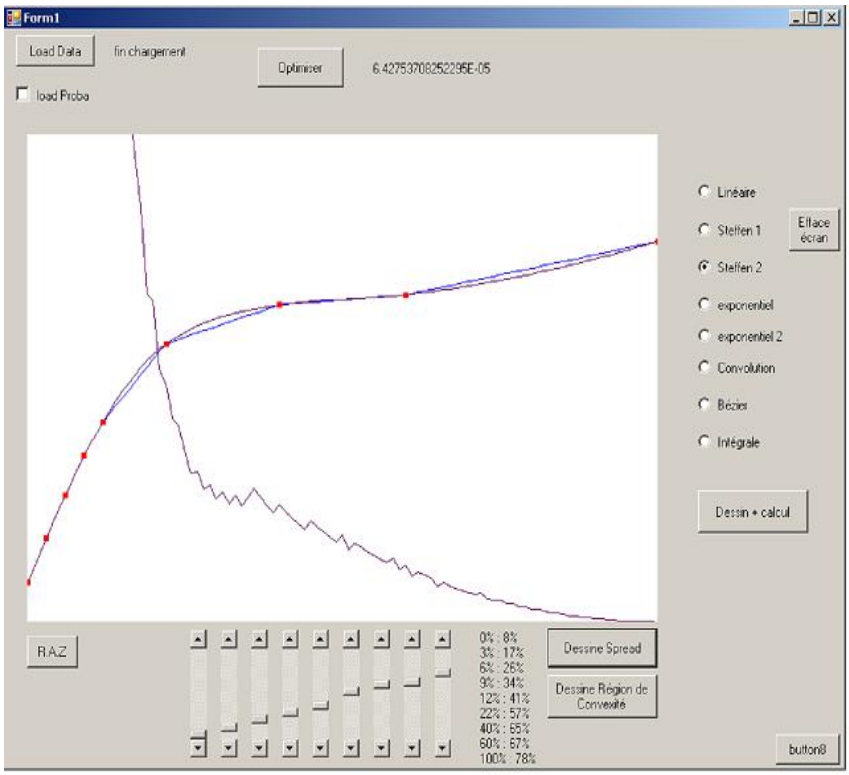


FIG. 20 – Environnement logiciel créé en C#

Les boutons du haut correspondent au chargement des données (la première partie de ce rapport expliquait comment calculer le spread d'un CDO, et comment nous avions optimisé les temps de calcul en créant un fichier assez lourd contenant des calculs déjà faits ce qui permet une utilisation particulièrement souple de ce présent logiciel)

Les boutons sur la droite correspondent au type d'interpolation que l'on choisit entre les points (nous en avons implémentés 8 en tout), et les boutons du bas permettent de faire varier dynamiquement l'interpolation en modifiant les ordonnées de points (dont les abscisses ont été décidées à l'avance afin de respecter les données que l'on obtient généralement du marché : par exemple le marché iTraxx côte seulement les tranches [0%, 3%], [3%, 6%], [6%, 9%], [9%, 12%], [12%, 22%] et [22%, 100%]).

Tout en bougeant ces "ascenseurs", la courbe d'interpolation est redessinée (et on peut vérifier qu'elle passe bien par les petits carrés rouges), et la courbe de spread sur tranchelettes déduite de cette courbe est aussi re-dessinée (ici il s'agit de la courbe quasi-décroissante, présentant quelques "pics").

Cet outil fût très pratique pour concevoir et vérifier des méthodes d'interpolation, permettant donc de vérifier "en temps réel" une méthode théorique.

Notons que la programmation orientée objet (notamment en C#) rend très simple la possibilité de passer des fonctions en paramètre.

4.2 Stabilité

Une notion naturelle bien que jamais définie dans les nombreuses documentations sur l'interpolation est la stabilité des interpolations.

En effet, beaucoup de documents traitent de l'interpolation d'une fonction, c'est à dire qu'ils cherchent une fonction (ou une suite de fonctions) approximant une autre plus compliquée, et s'interrogent sur la convergence de cette méthode (exemple : polynômes de Legendre, de Tchebychev, ...). La fonction à interpoler est connue, et les polynômes approchant la fonction essaient de passer par de plus en plus de points sur le graphe de la fonction de référence.

Pour les problèmes d'interpolation de données, on ne peut pas prendre "de plus en plus de points" : le problème est d'une toute autre nature, et il s'agit de définir les règles au cas par cas suivant le problème que l'on a.

Ici par exemple, les données du marché sont par "défaut" croissantes et convexes. On peut donc imposer à nos méthodes d'interpolations de garantir la convexité et la monotonie de nos jeux de données.

Ceci garantira par la même occasion la "stabilité" de notre système, sens que l'on n'a encore pas défini. En effet, si l'interpolation d'un graphe monotone est encore monotone, alors cela signifie que l'interpolation reste bornée vis à vis du jeu de donnée en entrée.

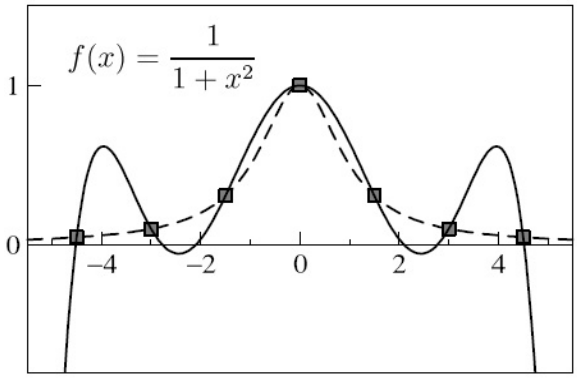


FIG. 21 – phénomène de Runge

Voici un exemple bien connu issu de la théorie de l'interpolation polynômiale des fonctions continues : il s'agit d'interpoler la fonction $f(x) = \frac{1}{1+x^2}$. La suite de polynômes de Lagrange dont les abscisses sont prises équiréparties sur $[-5, 5]$ ne converge pas vraiment vers la fonction $f(x)$ et même les polynömes divergent. Ceci est du au fait que pour passer par n points le polynôme Pn doit se courber de plus en plus entre les points afin de corriger sa trajectoire. Les coefficients des monômes le constituant sont donc de plus en plus grands pour se corriger les uns les autres.

Remarque : Afin d'éviter ces divergences, aussi connues sous le nom de "Phénomène de Runge", une bonne manière d'interpoler les fonctions grâce aux polynômes de Lagrange est d'utiliser des abscisses de Tchebychev au lieu d'abscisses équiréparties.

Il faut veiller à ce qu'il ne se produise pas le même type de dérèglement dans nos interpolations. Une difficulté est de formaliser ce raisonnement.

Dans notre cas, où les données forment un ensemble monotone et convexe, nous demandons à ce que les interpolations conservent la monotonie et la convexité, tout en passant par les points d'entrée.

Mais on peut aussi demander aux fonctions d'interpolation d'être stable par rapport aux données, et de ne pas diverger lorsque les données n'ont plus de propriété de convexité ni de monotonie.

Notation : Nous noterons I(x) une fonction d'interpolation associée à des abscisses $(x_1, ..., x_n)$ fixes et des ordonnées $(y_1, ..., y_n)$ mobiles.

On propose donc, outre la conservation de la monotonie et de la convexité , de s'intéresser aux 3 propriétés suivantes :

_ invariance de l'interpolation par changement de repère. Ceci traduit le fait que l'interpolation doit être purement géométrique et ne doit pas dépendre en toute logique du choix du repère, ce qui semble assez naturel.

_ continuité de la fonction qui à un n_uplet (les abscisses fixes $(x_1, ..., x_n)$) associe la fonction d'interpolation correspondante :

$$\left\{ \begin{array}{c} \mathbb{R}^n \to C([x_1, x_n] \to \mathbb{R}) \\ (y_1, y_2, ..., y_n) \to I(x_1, ...x_n, y_1, ..., y_n) \end{array} \right\}$$

La continuité est définie sur l'espace d'arrivée par la norme supérieure sur usuelle des fonctions continues sur un intervalle.

_ "bornitude" : $\exists A$ / $\forall (y_1, ..., y_n), \forall x \epsilon [x_1, x_n], |I_{(x_1,...x_n)}(x)| < A. \max_i, |y_i|$. Cette propriété traduit le fait qu'il y a une sorte de stabilité. Ainsi des interpolations sensibles au phénomène de Runge ne "passeraient" pas ce test, du moins on l'espère. Sinon la valeur minimale de A vérifiant cette propriété pourrait être un bon indicateur.

4.3 Explication des différentes interpolations

4.3.1 Linéaire

Cette méthode est la plus simple de toutes. Elle consiste uniquement à relier les points, triés par abscisse croissante, par des droites.

Sur $[x_i, x_{i+1}]$, la fonction d'interpolation vaut en x : $y_i + \frac{x-x_i}{x_{i+1}-x_i}.(y_{i+1} - y_i)$

Cette méthode est invariante par changement de repère puisque "purement géométrique". En outre elle est naturellement continue aussi puisque : $\forall x, \exists i / I(x) = y_i + \frac{x-x_i}{x_{i+1}-x_i}.(y_{i+1} - y_i)$. Ainsi les points $(x, I(x))$ de l'interpolation linéaire dépendent continuement des y_i (qui sont en nombre fini) et ce sur un intervalle fixe fermé borné $[x_1, x_n]$.

Enfin elle vérifie le critère de bornitude avec la constante $A = 1$.

La convexité et la monotonie sont aussi respectées par cette méthode.

Le principal problème de cette méthode est bien entendu dû au fait que l'interpolation linéaire ne donne pas une courbe naturelle, puisque affine par morceaux, alors qu'on cherche quelque chose de lisse (cf contrainte sur la "implied market law of losses")

4.3.2 Méthode de Steffen

Cette méthode est aussi appelée méthode du spline cubique. Elle diffère de la méthode du spline de lissage sur laquelle nous reviendrons plus loin.

Cette méthode est tirée de [6]. Elle a été crée dans le but de conserver la monotonie sur un jeu de données.

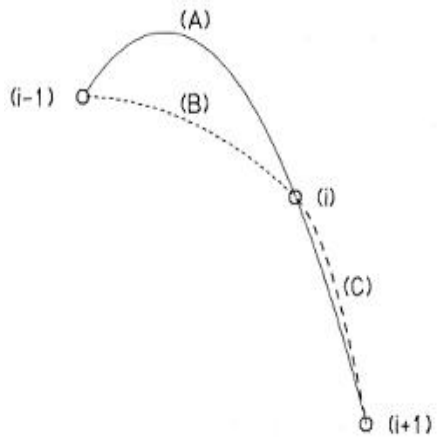


FIG. 22 – Exemple de non monotonie de la parabole passant par A, B et C

Principe : le défaut de l'interpolation linéaire est son manque de "lissage" autrement dit elle n'est pas continûment dérivable. Plutôt que d'utiliser une fonction affine par morceaux on va utiliser cette fois-ci une fonction polynômiale de degré 3 par morceaux. Puisqu'il y a plus de degrés de liberté sur les coefficients on va pouvoir obtenir un résultat C^1.

Réalisation : Etant donnés $\forall i, (y_i, y'_i)$ où les dérivées en chaque point sont donnés, sur $[x_i, x_{i+1}]$, on cherche a_i, b_i, c_i et d_i tels que $f_i(x) = a_i.(x - x_i)^3 + b_i.(x - x_i)^2 + c_i.(x - x_i) + d_i$

Remarque : Il ne faut pas confondre avec l'interpolation d'Hermite : on veut justement éviter le phénomène de Runge.

Il faut bien sûr choisir $d_i = y_i, c_i = y'_i$ puis s'en déduisent $b_i = \frac{3.s_i - 2.y'_i - y'_{i+1}}{h_i}$ et $a_i = \frac{y'_{ii} + y'_{i+1} - 2.s_i}{h_i^2}$ où $h_i = x_{i+1} - x_i$ et $s_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$

Il reste à déterminer les y'_i avec soin. Pour cela nous ne considérons que les points intérieurs (c'est à dire $1 < i < n$). L'idée est qu'il existe une unique parabole passant par $(x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1})$. Si cette parabole est monotone sur $[x_{i-1}, x_{i+1}]$, on va prendre pour y'_i la pente de cette parabole en x_i.

Sinon, on choisit la plus petite pente p, en valeur absolue, telle que les paraboles passant par $(x_{i-1}, y_{i-1}), (x_i, y_i)$ et ayant pour pente p en x_i et passant par $(x_i, y_i), (x_{i+1}, y_{i+1})$ et ayant pour pente p en x_i soient toutes les deux monotones sur leurs intervalles respectifs. Gardons à l'esprit que ceci est un choix purement arbitraire.

La parabole passant par $(x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1})$ a pour équation $y = a.(x - x_i)^2 + b.(x - x_i) + y_i$

Soit $\left\{ \begin{array}{l} y_{i+1} = a.(x_{i+1} - x_i)^2 + b.(x_{i+1} - x_i) + y_i \\ y_{i-1} = a.(x_{i-1} - x_i)^2 + b.(x_{i-1} - x_i) + y_i \end{array} \right\}$ ou encore avec nos notations $\left\{ \begin{array}{l} y_{i+1} = a.h_i^2 + b.h_i + y_i \\ y_{i-1} = a.h_{i-1}^2 + b.h_{i-1} + y_i \end{array} \right\}$

on a donc $h_{i-1}^2.y_{i+1} - h_i^2.y_{i-1} = b.h_i.h_{i-1}.(h_{i-1} - h_i) + (h_{i-1}^2 - h_i^2).y_i$

soit $b = \frac{h_{i-1}^2.y_{i+1} - h_i^2.y_{i-1} + (h_i^2 - h_{i-1}^2).y_i}{h_i.h_{i-1}.(h_{i-1} - h_i)} = p_i^* = \frac{s_{i-1}.h_i + s_i.h_{i-1}}{h_{i-1} + h_i}$

Il nous faut encore chercher une condition sur la pente en x_i pour qu'elle soit

monotone sur l'intervalle $[x_{i-1}, x_{i+1}]$. Cela arrive si et seulement si, en notant p_i la pente de la parabole en x_i, les paraboles passant par $(x_{i-1}, y_{i-1}), (x_i, y_i)$ et ayant pour pente p_i en x_i et passant par $(x_i, y_i), (x_{i+1}, y_{i+1})$ et ayant pour pente p_i en x_i soient toutes les deux monotones sur leurs intervalles respectifs.

Leurs équations sont respectivement $y = y_i + p_i.(x - x_i) + \frac{y_{i-1} - y_i - p_i.h_{i-1}}{h_{i-1}^2}(x - x_i)^2$ et $y = y_i + p_i.(x - x_i) + \frac{y_{i+1} - y_i - p_i.h_i}{h_i^2}(x - x_i)^2$

Les extremas ont pour abscisses respectives $x_i - \frac{h_{i-1}.p_i}{2(p_i - s_{i-1})}$ et $x_i + \frac{h_i.p_i}{2(p_i - s_i)}$. Pour que ces abscisses soient en dehors des intervalles, il faut que $0 \leq \frac{p_i}{s_{i-1}} \leq 2$ et $0 \leq \frac{p_i}{s_i} \leq 2$.

Finalement :

_ Si s_{i-1} et s_i sont de signes distincts, alors il ne peut pas y avoir de parabole passant par $(x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1})$ qui soit monotone. On choisit alors $y_i' = 0$. (D'après les petits calculs que l'on a fait c'est la seule condition permettant d'avoir la branche gauche et la branche droite monotones sur leurs intervalles respectifs)

_ Sinon : si p_i^* (pente "naturelle" de la parabole passant par les 3 points consécutifs) vérifie $|p_i^*| \leq 2.|s_i|$ et $|p_i^*| \leq 2.|s_{i-1}|$ alors on peut choisir $y_i^{'} = p_i^*$

_Sinon, on prend $y_i' = 2.signe(s_i).\min(|s_{i-1}|, |s_i|)$

Cet algorithme est bien entendu purement "heuristique", mais il permet de prouver qu'avec les choix de y_i' que l'on a fait, la monotonie des données est conservée par l'interpolation.

Remarque : Concernant les points extremaux y_1 et y_n, nous avons un petit peu le choix. L'interpolation est cubique par morceau, et C^1 sur l'intérieur du domaine. Mais nous avons implémenté 2 méthodes différentes ("Steffen 1" et "Steffen 2"), l'une raccordant les points extrémaux par des polynômes du second degré (en se basant sur la donnée de y_{n-1}, y_{n-1}' et y_n) l'autre effectuant un raccordement C^2 (en se basant sur la donnée de $y_{n-1}, y_{n-1}', y_{n-1}''$ (puisque l'on connait l'équation de la courbe sur laquell est le point $n-1$) et y_n)

Preuve Considérons des "données monotones" $(x_1, ..., x_n)$ et $(y_1, ..., y_n)$.

Par monotonie il suffit de prouver que l'interpolation I correspondante est monotone sur chaque intervalle $[x_i, x_{i+1}]$.

Fixons i avec $1 < i < n-1$.

Sur $[x_i, x_{i+1}]$, la courbe est cubique d'équation $f_i(x) = a_i.(x-x_i)^3 + b_i.(x-x_i)^2 + c_i.(x-x_i) + y_i$.

Sa dérivée vaut $f_i'(x) = 3a_i(x-x_i)^2 + 2b_i(x-x_i) + c_i$.

soit $f_i'(t) = 3(y_i' + y_{i+1}' - 2s_i).t^2 + 2(3s_i - 2y_i' - y_{i+1}')t + y_i'$ où $t = \frac{x-x_i}{x_{i+1}-x_i}$ varie entre 0 et 1 sur l'intervalle considéré.

En posant $\beta_t = \frac{f_i'(t)}{s_i}$, $\beta_1 = \frac{y_i'}{s_i} \geq 0$ et $\beta_2 = \frac{y_{i+1}'}{s_i} \geq 0$ (par "construction") on obtient

$\beta_t = (\sqrt{\beta_1}(1-t) - \sqrt{\beta_2}t)^2 + 2t(1-t)(3 + \sqrt{\beta_1\beta_2} - \beta_1 - \beta_2)$

soit $\beta_t = U^2 + VW$

$V = 2t(1-t) \geq 0$ sur notre intervalle. Le terme qui pourrait poser problème est W. Mais on a par "construction" encore une fois : $0 \leq \beta_1, \beta_2 \leq 2$.

Il suffit donc de prouver que $\forall 0 \leq x, y \leq \sqrt{2}, x^2 + y^2 - xy \leq 3$ afin de prouver que W reste positif.

A y fixé, on a une fonction de x seulement qui prend son maximum en 0 ou en $\sqrt{2}$. En 0 cela donne y^2 plus petit que 2 et donc que 3. En $\sqrt{2}$, on doit avoir $y^2 - \sqrt{2}y \leq 1$. Encore un polynôme du second degré, qui vaut 0 en 0 et en $\sqrt{2}$ (et qui est convexe).

La preuve est faite pour les intervalles intérieurs.

Considérons l'intervalle $[x_{n-1}, x_n]$ dans le cas d'un raccordement du second degré :

On a $f(x) = y_{n-1} + y_{n-1}'(x - x_{n-1}) + a(x - x_{n-1})^2$ et $a = \frac{s_{n-1} - y_{n-1}'}{h_{n-1}}$. D'autre part on a encore la condition $|y_{n-1}'| \leq 2|s_{n-1}|$. En faisant l'hypothèse que l'on nous a donné un jeu de données croissant, $y_{n-1}' \geq 0$ et donc pour vérifier la croissance sur l'intervalle $[x_{n-1}, x_n]$, comme on a une parabole, il suffit de vérifier $f'(x_n) \geq 0$.

Or $f'(x_n) = 2a.h_n + y_{n-1}' = 2(s_{n-1} - y_{n-1}') + y_{n-1}' = 2s_{n-1} - y_{n-1}' \geq 0$.

La condition est donc vérifiée. On peut prouver de même sur le premier intervalle.

Concernant le raccordement C^2 on ne peut pas conclure. ∎

En pratique cette méthode donne de très bons résultats.

Concernant les 3 propriétés :

_ il y a encore respect de la continuité dans l'interpolation : les fonctions cubiques dépendent continûment des n-uplets $(y_1, ..., y_n)$ et $(y_1', ..., y_n')$. D'autre part les calculs de ces derniers dépendent encore continûment de $(y_1, ..., y_n)$

_ propriété de changement d'échelle : elle est vérifiée aussi.

Preuve La preuve se fait en 2 étapes : premièrement, il faut montrer qu'étant donné 2 points A et B du plan, l'unique courbe cubique passant par A et B, et ayant les pentes données en ces points ne change pas après modification de l'échelle. (les pentes changeant en conséquence bien entendu). Il faut ensuite vérifier que l'algorithme de choix du n-uplet $(y_1', ..., y_n')$ donne le même n-uplet après changement d'échelle.

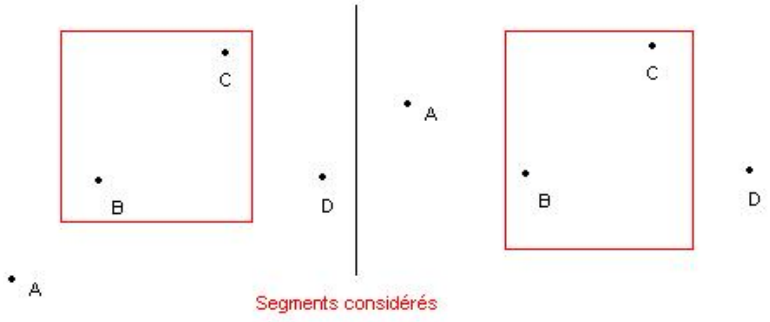


FIG. 23 – Différents cas à considérer

_ Pour simplifier, considérons dans un repère R les points $A(0,0)$ et $B(a,b)$ qui ont pour équivalent dans R' $A(0,0)$ et $B(a\prime,b\prime)$. Les pentes dans R valent y'_0 et y'_1. Dans R', il faut tester avec $\frac{b'a}{ba'}y'_0$ et $\frac{b'a}{ba'}y'_1$.

L'unique courbe cubique passant par A et B et ayant pour pentes y'_0 et y'_1en A et B a pour équation dans R : $f(x) = \frac{y'_0+y'_1-2\frac{b}{a}}{a^2}x^3 + \frac{3\frac{b}{a}-2y'_0-y'_1}{a}x^2 + y'_0.x$

L'unique courbe cubique passant par A et B et ayant pour pentes $\frac{b'a}{ba'}y'_0$ et $\frac{b'a}{ba'}y'_1$en A et B a pour équation dans R' : $g(x) = \frac{y'_0\cdot\frac{b'a}{ba'}+y'_1\cdot\frac{b'a}{ba'}-2\frac{b'}{a'}}{a'^2}x'^3 + \frac{3\frac{b'}{a'}-2y'_0\cdot\frac{b'a}{ba'}-y'_1\cdot\frac{b'a}{ba'}}{a}x'^2 + y'_0.\frac{b'a}{ba'}.x'$

La courbe est invariante par changement d'échelle si $(x\prime, y\prime = g(x\prime))$ point de R' est sur la courbe représentée par le même point (x, y) dans R.

Avec $x' = \frac{a'}{a}x$ et $y' = \frac{b'}{b}y$ il faut vérifier si on a bien $y = f(x)$ sachant que $y\prime = g(x\prime)$.

Or $\frac{b'}{b}y = g(\frac{a'}{a}x)$ soit $y = \frac{b}{b'}g(\frac{a'}{a}x) = ... = f(x)$ (il suffit d'écrire, il n'y a aucune astuce particulière)

La courbe est donc bien invariante par changement d'échelle. ∎

_ propriété de "bornitude" : elle est vérifiée aussi avec une constante de 1. On a même plus fort que la bornitude en fait : pour tous points A et B "fixes", la courbe interpolatrice a ses ordonnées comprises entre celles de A et celle de B, sur le segment $[x_A, x_B]$. (de même que pour l'interpolation linéaire) Ceci est toujours vrai, pas seulement dans notre cas où les données sont monotones.

Preuve On a vu que la monotonie est conservée par l'interpolation ce qui est en soit une preuve partielle de bornitude.

Le graphique montre la disjonction de cas que l'on opère.

On considère la courbe sur [B, C]. (Autres cas possibles : D est au dessus de C, mais alors bornitude par monotonie pour le cas 1, ou symétrie par rapport à l'axe des ordonnées du cas 2)

Cas 1 : Par "construction", on a $y'_C = 0$. On a alors

Cas 2 : Par "construction", on a $y'_B = 0$ et $y'_C = 0$. On a alors entre B et C $I(x) = \frac{-2s_i}{h_i^2}(x-x_b)^3 + \frac{3s_i}{h_i}(x-x_b)^2 + y_B = y_B + s_i.h_i.\frac{(x-x_b)^2}{h_i^2}(3-2.\frac{x-x_b}{h_i})$

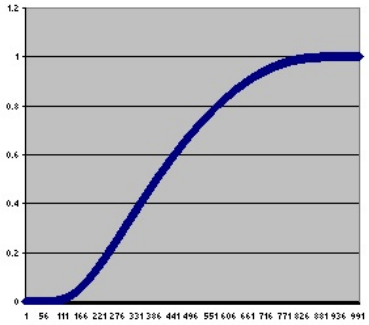


FIG. 24 – Exemple de fonction f vérifiant ces contraintes

$= y_B + (y_C - y_B).\frac{(x-x_b)^2}{h_i^2}(3 - 2.\frac{x-x_b}{h_i}) = y_B + (y_C - y_B).X^2(3 - 2X)$ et X varie entre 0 et 1.

Or $x \to x^2(3-2x)$ est croissante sur $[0,1]$ donc et varie de 0 à 1. Le résultat est donc prouvé. ∎

4.3.3 Interpolation Exponentielle

En reprenant l'idée de l'article [1] traitant de la "Market Law of Losses", expliquant que les discontinuités sur le smile de corrélation entraînent des discontinuités et donc sûrement des arbitrages sur la courbe de spread de tranchelette, et en voyant que l'interpolation de Stefen donnait de bons résultats alors que C^1 seulement, nous avons décidé de rechercher une interpolation C^∞.

Principe : soit f une fonction C^∞ valant 0 en 0 et 1 en 1, avec toutes les dérivées nulles en 0 et en 1. (cf Appendice pour une formule)

Considérons que l'on doive raccorder les points A et B, et où les suites des dérivées successives en ces 2 points sont connues : $a_0, a_1, ..., a_n$ et de même avec $b_0, b_1, ..., b_n$ (où $a_0 = y_A$ et $b_0 = y_B$).

Alors sur $[x_A, x_B]$, la fonction $g(x) = (a_0 + a_1.(x - x_A) + \frac{a_2}{2}.(x - x_A)^2 + \frac{a_3}{3!}.(x - x_A)^3 + ...)(1 - f(\frac{x-x_A}{x_B-x_A})) + (b_0 + b_1.(x_B - x) + \frac{b_2}{2}.(x_B - x)^2 + \frac{b_2}{3!}.(x_B - x)^3 + ...).f(\frac{x-x_A}{x_B-x_A})$ est C^∞et vérifie que toutes ses dérivées en A et B respectivement valent $a_0, a_1, ..., a_n$ et $b_0, b_1, ..., b_n$.

Afin de créer une suite de dérivées en chaque point, nous utilisons l'algorithme de l'interpolation de Stefen, qui nous fournit au moins la première dérivée pour chaque point (ceci pour "exponentielle 1") et même, connaissant l'équation de la fonction cubique sur chaque segment, la dérivée seconde. Pour "exponentielle 2" nous prenons comme valeurs des dérivées premières en chaque point la moyenne entre la pente à gauche et à droite (vis à vis des différences finies) : si A, B et C sont 3 points qui se succèdent ; AB ayant une pente α et BC une pente β, alors on prend $b_1 = \frac{\alpha+\beta}{2}$. Les dérivées d'ordre supérieur sont toujours prises nulles. Aux points extrémaux on

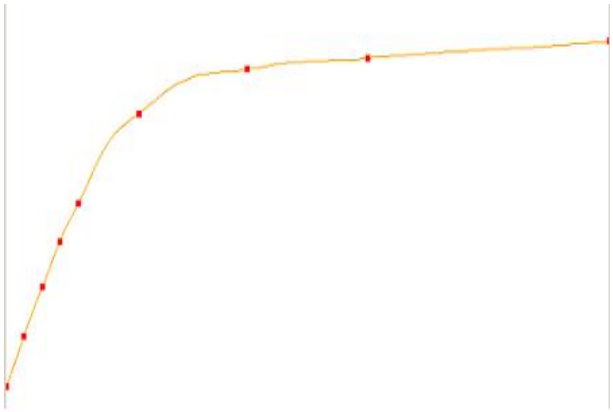


FIG. 25 – Résultat de l'interpolation exponentielle

ne prend bien sûr que la valeur de la "différence finie" entre le point extrémal en question et son point adjacent.

Ce procédé d'interpolation n'est pas vraiment à la hauteur de nos attentes. Il y a plusieurs raisons à cela, que l'on découvre "après coup" :

_ la première c'est que si l'interpolation est effectivement C^∞ elle n'en reste pas moins très linéaire aux abords des points. Et pour cause. La fonction f que l'on a choisit dans le graphe précédent (ralliant 0 à 1de manière C^∞) est beaucoup trop "plate" aux alentours de 0 et de 1. Ce qui fait que sur $[x_A, x_B]$, la fonction $g(x) = (a_0 + a_1.x + \frac{a_2}{2}.x^2 + \frac{a_3}{3!}.x^3 + ...)(1 - f(x)) + (b_0 + b_1.(1-x) + \frac{b_2}{2}.(1-x)^2 + \frac{ab}{3!}.(1-x)^3 + ...).f(x)$

se réduit à (on simplifie en prenant [0,1]) $g(x) = (a_0 + a_1.x).(1 - f(x)) + (b_0 + b_1.(1-x)).f(x)$

Vers 0 : $f(x) = 0$ pendant assez longtemps donc $g(x) = a_0 + a_1.x$ pendant assez longtemps aussi.

Et idem en 1 : $g(x) = b_0 + b_1(1-x)$

_ La seconde raison est que l'on crée un point d'inflexion preque "automatiquement" en interpolant sur chaque segment. Alors que si l'on a un jeu de données convexe à interpoler, il ne devrait logiquement pas y avoir de point d'inflexion.

Exemples : Cette image décrit l'interpolation exponentielle réalisée entre A (abscisse 0, ordonnée 0, dérivée 1) et B (abscisse 1, ordonnée 0.5, dérivée 0). Signe que notre fonction n'est pas parfaite, elle s'élève au dessus de B avant de redescendre, au lieu de ne jamais dépasser l'ordonnée de ce point, comme on l'aimerait.

Autre exemple :

FIG. 26 – Point d'inflexion non désiré

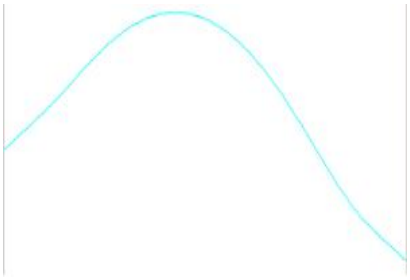


FIG. 27 – Autre exemple de point d'inflexion non désiré

Cette fois : A(abscisse 0, ordonnée 0.5, dérivée 1) et B (abscisse 1, ordonnée 0.1, dérivée -1). Alors que la fonction pourrait être convexe elle ne l'est pas (on note un léger point d'inflexion vers l'abscisse 1/4).

Par contre : en choisissant des points initiaux qui demandent naturellement une interpolation à point d'inflexion, les résultats sont assez satisfaisants.

Une manière de bien interpoler serait peut être de choisir 2 types de fonction différents. Notre manière actuelle d'interpoler de manière exponentielle pour les points nécessitant un point d'inflexion, et une autre méthode pour les graphes ne nécessitant

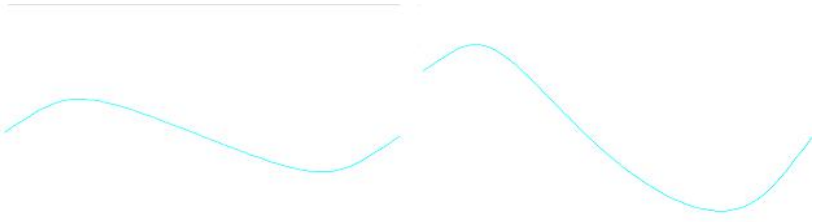


FIG. 28 – A(0.5 ; 1) et B (0.5 ; 1) sur le premier graphe, et A(0.7, 1) et B(0.5 ; 2) sur le second.

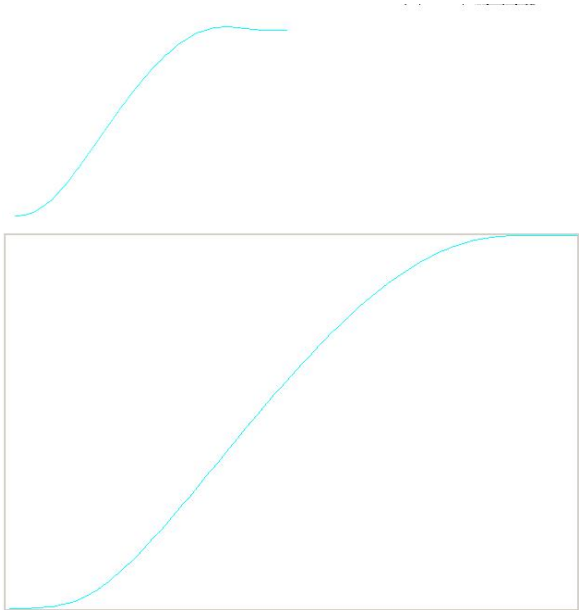


FIG. 29 – Invariance graphique par changement de repère

pas naturellement de point d'inflexion dans l'interpolation (cf les 2 premiers "contre-exemples" que l'on a donné dans cette partie). Seulement il est difficile de généraliser une telle méthode : supposons que l'on ait une fonction f, C^∞, convexe, croissante, partant de A(0,0) et arrivant en B(1,1), de premières dérivées respectives 2 et 0. On n'a pas trouvé de moyen simple, à partir de f, de reconstituer une fonction g, C^∞, convexe, croissante, partant de A(0,0) et arrivant en B(1,1), de premières dérivées respectives 4 et 0.

Caractéristiques de l'implémentation actuelle :

_ continuité par rapport aux ordonnées : elle est vérifiée, puisque l'on prend pour dérivées aux points fixes soit les dérivées données par la méthode de Steffen (continues) soit les différences finies (continues aussi). Ensuite on a une formule qui dépend encore continuement de ces dérivées, et ce sur un nombre fini de points et un intervalle borné.

_ invariance par changement de repère : celle-ci n'est pas respectée en toute généralités, mais l'est avec notre méthode.

En effet considérons notre interpolation avec A(0,0) de dérivée première nulle et de dérivée seconde 2, et B(1,1) de dérivées 0. (dans R', A(0,0) dérivée seconde $(\frac{b}{a})^2$ et B(a, b) dérivées 0).

La fonction d'interpolation a dans R l'équation $y = x^2.f(x) + (1 - f(x))$ et dans R' : $y' = (\frac{b}{a})^2 x'^2.f(\frac{x'}{a}) + b(1 - f(\frac{x'}{a}))$ soit dans R une courbe d'équation $b.y = b^2x^2.f(x) + b(1 - f(x))$ ou encore $y = b.x^2.f(x) + (1 - f(x))$. Et cela ne donne pas lieu aux deux même courbes ! (si b est différent de 1)

on voit nettement sur le second graphe que la courbe passe au dessus du point B, alors que cela ne se produit pas sur le second. (premier : "fenêtre" (0,0)-(0.5, 0.5) ; second : "fenêtre" (0,0) - (1,1))

Par contre dans notre méthode on n'utilise les dérivées que jusqu'au premier ordre. Et en considérant les formules du contre exemple, on voit bien que le 'b' disparaitrait complètement.

_ "bornitude" de cette méthode : on peut le prouver pour les 2 méthodes à la fois : choix des dérivées de Stefen ou choix de la moitié des différences finies à gauche et à droite du point. Dans les 2 cas on a en chaque point x_i : $|y'_i| \leq 2.\frac{|y_{i+1}-y_i|}{x_{i+1}-x_i}$ et $|y'_i| \leq 2.\frac{|y_i-y_{i-1}|}{x_i-x_{i-1}}$ ou bien $|y'_i| \leq \frac{|y_{i+1}-y_{i-1}|}{x_{i+1}-x_{i-1}}$

Ainsi entre i et $i+1$: $y = (y_i + y'_i.(x-x_i))(1-f(t)) + (y_{i+1} + y'_{i+1}(x-x_{i+1}))f(t)$ et donc $|y| \leq \max(|y_i| + (x_{i+1}-x_i).|y'_i|, |y_{i+1}| + (x_{i+1}-x_i).|y'_{i+1}|)$

Or $(x_{i+1}-x_i).|y'_i| \leq 2|y_{i+1}-y_i|$ dans tous les cas. Soit $(x_{i+1}-x_i).|y'_i| \leq 2 * 2\max_i(|y_i|) = 4\max_i(|y_i|)$. Idem pour le deuxième cas.

On a donc finalement $|y| \leq 5\max_i(|y_i|)$ et la constante A vaut donc 5. On peut en fait réduire cette approximation quelque peu grossière et avoir à 3 en remarquant que $t.f(1-t)$ est plus petit que $\frac{1}{2}$ au lieu de 1.

Cela reste encore grossier puisque avec notre programme nous voyons bien qu'il n'y a pas d'énormes disparités entre les données et cette interpolation. Cependant la constante A doit être strictement supérieure à 1 étant donné les dessins où l'on montrait la non convexité de cette interpolation. (entre $A(x_A, y_A)$ et $B(x_B, y_B)$ on peut avoir y n'appartient pas à $[y_A, y_B]$).

4.3.4 Interpolation par Bézier

Les courbes de bézier étant connues pour donner de jolies courbures, il a paru très naturel de vouloir les utiliser ici. Historiquement, les courbes de Bézier furent inventées par l'ingénieur du même nom, en 1962, chez Renault, qui cherchait à modéliser certaines pièces automobiles par ordinateur.

Pour n+1 points de contrôle $P_1, ..., P_n$, on définit une courbe de Bézier par l'ensemble des points $\sum_{i=0}^{n} B_i^n(t)P_i, t\epsilon[0,1]$ et où les B_i^n sont les polynômes de Bernstein. Le polygône est appelé « polygône convexe de Bézier ».

Définition : Pour un degré n, il y a n+1 polynômes de Bernstein définis, sur l'intervalle [0,1], par $B_i^n = \binom{n}{i}X^i(1-X)^{n-i}$

Remarque : Puisque $\sum_{i=0}^{n} B_i^n(t) = 1$, alors la courbe est correctement définie. Chaque point de la courbe peut être vu comme un barycentre des points de contrôle P_i.

Remarque : Les courbes de Bézier dépendent de l'ordre que l'on donne aux points $P_1, ..., P_n = \{(x_1, y_1), ..., (x_n, y_n)\}$. On considèrera toujours pour nos interpolations que ces points correspondent aux $A_1, ..., A_n = \{(x_1, y_1), ..., (x_n, y_n)\}$ (les n points fixes à interpoler) où $x_1 < ... < x_n$.

Remarque : Nous n'avons pas implémenté l'interpolation de Lagrange (ou de Hermite) pour les raisons que l'on a données plus tôt ("phénomène de Runge"). Mais il ne faut pas se fier aux apparences : une courbe de Bézier n'est pas une courbe polynômiale. En effet $x(t)$ et $y(t)$ dépendent polynômialement du temps, mais l'on n'a pas $y = P(x)$ où P serait un polynôme. En effet, $x = Q(t)$ et l'on aurait alors $y = P(Q(t))$ où Q et $P(Q)$ doivent avoir même degré (n). Ainsi P serait de degré 1 et la courbe obtenue serait affine. Ce qui n'est bien sûr pas le cas (dans le cas général tout du moins : $\overrightarrow{P_1P_2}$ est en effet le vecteur tangent à la courbe en $t = 0$ et $\overrightarrow{P_{n-1}P_n}$ celui en $t = 1$.

En fait une courbe de Bézier passe toujours pas P_1 et P_n. Elle ne passe généralement pas par les autres points. Notre problème consiste donc à trouver $P_2, ..., P_{n-1}$tels que la courbe de Bézier associée soit une interpolation "exacte".

Commençons par prouver quelques résultats de notre cru concernant les courbes de Bézier :

_ Etant donnés $A_1, ..., A_n = \{(x_1, y_1), ..., (x_n, y_n)\}$ il existe n points uniques aux abscisses fixes $x_1, ..., x_n$ tels que la courbe de Bézier issue de ces points passe par $A_1, ..., A_n$. On peut alors parler de "l'interpolation de Bézier aux points $A_1, ..., A_n$".

Etablissons en premier lieu un lemme intermédiaire avant d'attaquer la preuve.

Lemme II.2 *Montrons d'abord que $t \to x(t)$ est strictement croissante lorsque les x_i sont triés par ordre croissant comme c'est le cas. On suppose dans un premier temps que les x_i sont tous positifs.*

Preuve On a $x(t) = \sum_{i=0}^{n} \binom{n}{i} x_i.t^i.(1-t)^{n-i}$.

Fixons $0 \leq t < u \leq 1$. Soit k_0 tel que $\forall k \leq k_0, t^k(1-t)^{n-k} \geq u^k(1-u)^{n-k}$ et $\forall k > k_0, t^k(1-t)^{n-k} < u^k(1-u)^{n-k}$. k_0 existe car en $k = 0, t^k(1-t)^{n-k} = (1-t)^n > (1-u)^n$ et en $k = n, , t^k(1-t)^{n-k} = t^n < u^n$ et d'autre part $s_k = \frac{t^k(1-t)^{n-k}}{u^k(1-u)^{n-k}}$ vérifie $s_k = (\frac{1-t}{1-u})^n(\frac{\frac{t}{1-t}}{\frac{u}{1-u}})^k$ et comme $t < u$, $\frac{t}{1-t} < \frac{u}{1-u}$ et ainsi (s_k) est une suite décroissante supérieure à 1 en 0 et inférieure à 1 en n. k_0 est donc "l'instant de passage" de s_k sous 1.

Ceci fait, on a $x(u) - x(t) = \sum_{k=0}^{n} x_k(b_k^n(u) - b_k^n(t))$ (b_k^n polynôme de Bernstein)

soit $x(u) - x(t) = \sum_{k>k_0} x_k(b_k^n(u) - b_k^n(t)) - \sum_{k \leq k_0} x_k(b_k^n(t) - b_k^n(u))$ où chaque terme est une pondération positive des x_k. (positifs eux aussi comme on l'a supposé)

Ainsi $x(u) - x(t) > x_{k_0}(\sum_{k>k_0} b_k^n(u) - b_k^n(t)) - x_{k_0}(\sum_{k \leq k_0} b_k^n(t) - b_k^n(u))$. Or $\sum_{k=0}^{n} b_k^n(t) = \sum_{k=0}^{n} b_k^n(u) = 1$ donc $x(u) - x(t) > x_{k_0}(\sum_{k=0}^{n} b_k^n(u) - \sum_{k=0}^{n} b_k^n(t)) = 0$.

Ceci prouve le lemme en partie.

Maintenant si les x_i ne sont pas tous positifs : soit A tel que $\forall i, (x_i + A) \geq 0$. $x(t) = -A + \sum_{i=0}^{n} \binom{n}{i}(x_i + A).t^i.(1-t)^{n-i}$ et les $(x_i + A)$ sont encore triés par ordre croissant et sont maintenant positifs. $x(t)$ est donc encore strictement croissant en t. ∎

Preuve Prouvons maintenant l'existence et l'unicité d'une courbe de Bézier issue de (n+1) points aux abscisses fixes $x_0, ..., x_n$ (triées par ordre croissant) passant par $A_0, ..., A_n = \{(x_0, y_0), ..., (x_n, y_n)\}$.

Premièrement, $t \to x(t)$ est croissante d'après notre lemme précédent, et continue (car polynômiale). De plus, $x(0) = x_0$ et $x(1) = x_n$. Ainsi, comme $t \to x(t)$ est strictement croissante, par le théorème des valeurs intermédiaires, $\exists!(t_0, ..., t_n)\epsilon[0,1]^{n+1}$ tel que $\forall i, x(t_i) = x_i$. Bien sûr, $t_0 = 0, t_n = 1$. Les t_i sont encore strictement croissants.

La fonction d'interpolation de Bézier s'écrit :

$(z_0, ..., z_n) \to B(z_0, ..., z_n) = (y'_0, ..., y'_n)$. Il nous faut montrer qu'il existe un unique n+1 uplet $(z_0, ..., z_n)$ qui donne $(y_0, ..., y_n)$. (les ordonnées des points A_i)

En fait B est linéaire : $B(z_0, ..., z_n) = (y'_0, ..., y'_n)$ et $y'_j = \sum_{i=0}^{n} B_i^n(t_j)z_i$.
Sa matrice est donc : $B_{i,j} = B_j^n(t_i)$ soit

$$B = \begin{matrix} \binom{n}{0}t_0^0(1-t_0)^n & \binom{n}{1}t_0^1(1-t_0)^{n-1} & ... & \binom{n}{n}t_0^n(1-t_0)^0 \\ \binom{n}{0}t_1^0(1-t_1)^n & ... & ... & \binom{n}{n}t_1^n(1-t_1)^0 \\ ... & ... & ... & ... \\ \binom{n}{0}t_n^0(1-t_n)^n & ... & ... & \binom{n}{n}t_n^n(1-t_n)^0 \end{matrix}$$

$$\det(B) = \prod_{i=0}^{n}\binom{n}{i}.\det\begin{pmatrix} t_0^0(1-t_0)^n & t_0^1(1-t_0)^{n-1} & ... & t_0^n(1-t_0)^0 \\ t_1^0(1-t_1)^n & ... & ... & t_1^n(1-t_1)^0 \\ ... & ... & ... & ... \\ t_n^0(1-t_n)^n & ... & ... & t_n^n(1-t_n)^0 \end{pmatrix}$$

$$= \prod_{i=0}^{n}\binom{n}{i}.\prod_{i=1}^{n-1}(1-t_i)^n.\det\begin{pmatrix} 1 & 0 & ... & 0 \\ (\frac{t_1}{1-t1})^0 & ... & ... & (\frac{t_1}{1-t1})^n \\ (\frac{t_2}{1-t_2})^0 & ... & ... & (\frac{t_2}{1-t_2})^n \\ ... & ... & ... & ... \\ (\frac{t_{n-1}}{1-t_{n-1}})^0 & & & (\frac{t_{n-1}}{1-t_{n-1}})^n \end{pmatrix}$$

$$= \prod_{i=0}^{n}\binom{n}{i}.\prod_{i=1}^{n-1}(1-t_i)^n.\det\begin{pmatrix} 0 & 0 & 0 & 1 \\ (\frac{t_1}{1-t1})^1 & ... & ... & (\frac{t_1}{1-t1})^n \\ (\frac{t_2}{1-t_2})^1 & ... & ... & (\frac{t_2}{1-t_2})^n \\ ... & ... & ... & ... \\ (\frac{t_{n-1}}{1-t_{n-1}})^1 & & & (\frac{t_{n-1}}{1-t_{n-1}})^n \end{pmatrix}$$

$$= \prod_{i=0}^{n}\binom{n}{i}.\prod_{i=1}^{n-1}(1-t_i)^n.\det\begin{pmatrix} 0 & 0 & 0 & 1 \\ (\frac{t_1}{1-t1})^1 & ... & ... & (\frac{t_1}{1-t1})^{n-1} \\ (\frac{t_2}{1-t_2})^1 & ... & ... & (\frac{t_2}{1-t_2})^{n-1} \\ ... & ... & ... & ... \\ (\frac{t_{n-1}}{1-t_{n-1}})^1 & & & (\frac{t_{n-1}}{1-t_{n-1}})^{n-1} \end{pmatrix}$$

et ce dernier est un déterminant de Vandermonde (transposé mais cela revient au même) et comme les t_i sont distincts, B est donc inversible, ce qui prouve le résultat final. ∎

_ Concernant la propriété de changement d'échelle, comme l'application est linéaire elle le respecte.

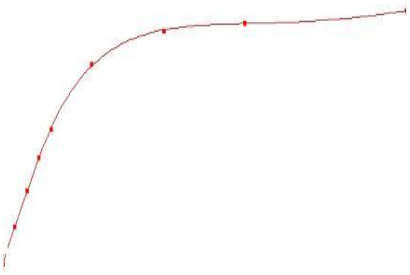


FIG. 30 – Exemple d'interpolation par Bézier très réussie

_ Concernant la propriété de bornitude, celle-ci est forcément vérifiée (les applications linéaires en dimension finie sont continues). La constante A est à relier à la norme de B^{-1}.

On peut clairement majorer la norme infinie (celle que l'on a choisie pour la "bornitude") de B par 1. Comme $1 \leq ||B||.||B^{-1}||$ on a $||B^{-1}|| \geq 1$. Mais cela reste encore très grossier.

_ Concernant la continuité par rapport au changement des points : encore respectée puisqu'une courbe de Bézier dépend continuement sur un intervalle fermé borné ([0,1]) des points, et qu'ici le système est linéaire en dimension finie par rapport aux points, donc continu aussi.

Avec les données de base, l'interpolation est vraiment agréable, lisse et la courbe passe sans trop de difficultés par tous les "points rouges" = les valeurs du marché que l'on doit "cibler".

Implémentation : il s'agit d'inverser la matrice dont on a calculé le déterminant dans la preuve ci-dessus afin de connaître les ordonnées des points P_i. Ceci fait, on calcule une bonne fois pour toutes les instants t_i tels que $x(t_i) = x_i$ (par dichotomie ; unicité de ces instants prouvée par notre premier résultat) puis il n'est pas compliqué de faire calculer par l'ordinateur la formule définissant P_t en fonction des t_i.

Remarque : Si l'on cherche à déterminer heuristiquement ces ordonnées, on part de $P_1, ..., P_n = \{(x_1, y_1), ..., (x_n, y_n)\}$ (les points que l'on a à interpoler). On calcule ensuite la distance de la courbe de Bézier à ces points, c'est à dire la somme des carrés des différences entre les y_i et l'ordonnée de la courbe de Bézier au point d'abscisse x_i. Reste ensuite à modifier les y_i (sauf y_1 et y_n qui sont naturellement des points de la courbe) jusqu'à approcher à ϵ près notre but. A nombre d'itérations fixé, l'algorithme converge assez bien en général. Mais parfois il y a divergence : une sorte de phénomène de Runge se produit et les P_i ont leurs ordonnées qui grandissent afin de satisfaire les contraintes.

Etant donné que l'on a une matrice à inverser, de dimension souvent faible, cet algorithme heuristique semble inutile. Il donne cependant de très bonnes courbes de tendance, comme vous pouvez le constater sur les figures.

FIG. 31 – Interpolation difficile, mais bonne courbe de tendance.

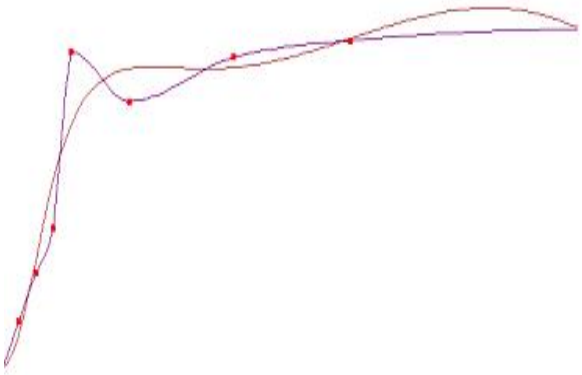


FIG. 32 – Steffen et Bézier en vis à vis

Cette dernière est effectivement faite pour passer par tous les points (ce que l'on veut). Même si quand on y réfléchis bien la courbe de Bézier semble mieux afficher la "tendance" de la courbe.

En fait l'interpolation de Bézier est très bonne quand les points par lesquels ont doit passer suivent un "spline naturel". Sinon quand il y a une "imperfection" comme c'est le cas dans l'exemple ci-dessous, il est normal que l'interpolation soit mauvaise puisque les polynômes ne sont pas faits pour modéliser des imperfections (Diracs, ...) et ils y réagissent très mal.

4.3.5 Interpolation par convolution

L'idée de cette méthode vient d'un résultat bien connu : Si $f \epsilon L^1$ et $g \epsilon C^\infty$, alors $f_* g \epsilon C^\infty$. Et si l'on choisit $g_n \to \delta_0$ alors $f_* g_n \to f$. (Résultat de la théorie des distributions)

Bref la convolution peut lisser une fonction sans trop la modifier. On a donc décidé d'utiliser cette méthode en partant non pas de l'interpolation linéaire, mais de l'interpolation qui a donné les meilleurs résultats jusqu'ici : l'interpolation de Steffen. On convole ensuite cette fonction, et on regarde si le résultat n'est pas trop éloigné des points que l'on veut atteindre. Au besoin on modifie légèrement les ordonnées, on réinterpole selon Steffen, on convole le tout, et on recommence jusqu'à atteindre avec une précision fixée à l'avance chacun des points que l'on a en entrée.

Implémentation : dans la pratique cette idée ne fonctionne pas aussi bien que cela. On a en effet un nombre fini de points espacés régulièrement. La convolution revient donc à prendre pour chaque point une sorte de moyenne des points autour, avec une pondération. Normalement la "vraie convolution" peut aller chercher des points très éloignés et les pondérer très peu par rapport aux points voisins de celui en question, mais quitte à mettre un coefficient quasi nul, autant ne pas en mettre dans la pratique.

Nous avons donc choisi de prendre $y^n_{t+1} = \frac{y^n_t + 0.7 * (y^{n-1}_t + y^{n+1}_t) + 0.3 * (y^{n-2}_t + y^{n+2}_t)}{1+0.7+0.7+0.3+0.3}$ (tout en veillant aux effets de bords) et d'itérer sur la suite (y_t) obtenue.

Donc aux points d'abscisses fixes $A_0, ..., A_n = \{(x_0, y_0), ..., (x_n, y_n)\}$ par lesquels on doit passer, on choisit d'abord d'interpoler selon Steffen, de convoler, et de modifier les y^i_t jusqu'à se rapprocher des y_i. Le même algorithme "d'optimisation" que celui pour Bézier peut être utilisé (modification d'une fonction, choix de la garder ou non suivant la distance à une autre fonction, ...).

Seulement il se trouve que le résultat obtenu est très très très proche de l'interpolation de Steffen (en tout cas sur les cas usuels du smile de corrélation) : la convolution ne modifie presque pas la fonction initiale ce qui fait qu'elle passe presque par les points initiaux et qu'il y a très peu à modifier pour que cela se produise.

On peut voir sur cette image (en vert : convolution, en violet Steffen) que cette dernière est très proche de Steffen sur les points sans trop de "pics" mais que dès qu'il y en a (vers la gauche) la convolution explose.

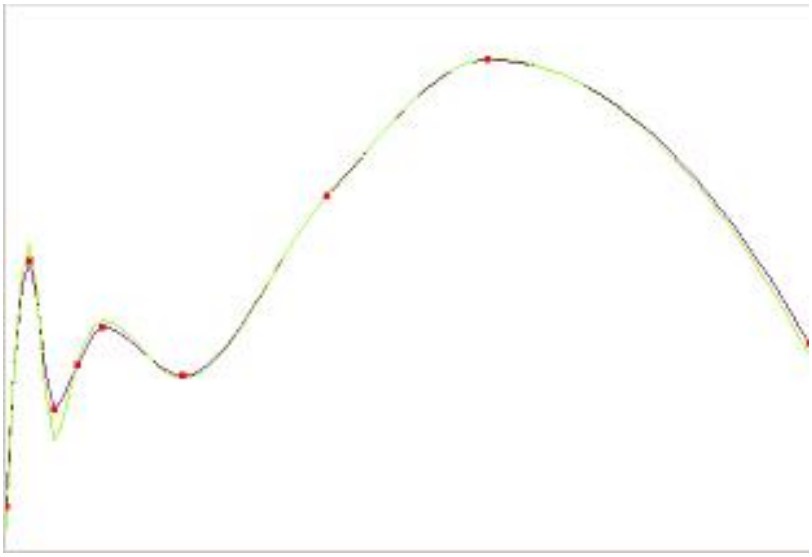


FIG. 33 – Comparaison Steffen - interpolation par convolution

Pour prouver théoriquement l'existence et l'unicité des ordonnées à avoir afin d'atteindre tous les points voulus, il faut comme pour Bézier résoudre un système sauf que cette fois-ci il est beaucoup trop compliqué puisqu'il dépend de l'interpolation de Steffen.

Sinon, la convolution que l'on crée est en elle-même linéaire.

Sa matrice est :

$$\begin{array}{ccccccccc}
\frac{1}{2} & \frac{.7}{2} & \frac{.3}{2} & & & & & & \\
\frac{.7}{1+2*.7+.3} & \frac{1}{1+2*.7+.3} & \frac{.7}{1+2*.7+.3} & \frac{.3}{1+2*.7+.3} & & & & & \\
\frac{.3}{1+2*.7+2*.3} & \frac{.7}{1+2*.7+2*.3} & \frac{1}{1+2*.7+2*.3} & \frac{.7}{1+2*.7+2*.3} & \frac{.3}{1+2*.7+2*.3} & & & & \\
 & \frac{.3}{1+2*.7+2*.3} & \frac{.7}{1+2*.7+2*.3} & \frac{1}{1+2*.7+2*.3} & \frac{.7}{1+2*.7+2*.3} & \frac{.3}{1+2*.7+2*.3} & & & \\
 & & \frac{.3}{1+2*.7+2*.3} & \frac{.7}{1+2*.7+2*.3} & \frac{1}{1+2*.7+2*.3} & \frac{.7}{1+2*.7+2*.3} & \frac{.3}{1+2*.7+2*.3} & & \\
 & & & \ldots & \ldots & \ldots & \ldots & \ldots &
\end{array}$$

et ce n'est pas un problème facile que d'étudier sa norme ou son inversibilité, même si les matrices plus ou moins à diagonale dominante le sont souvent. (D'ailleurs les matrices à diagonale dominante le sont : Hadamard)

Cependant on ne modifie pas le caractère inversible ou non d'une matrice en multipliant une ligne ou une colonne par un coefficient non nul.

On multiplie donc la première et la dernière ligne par $\frac{2}{1+2*.7+2*.3}$ et la seconde et l'avant dernière par $\frac{1+2*.7+.3}{1+2*.7+2*.3}$.

En notant alors $a = \frac{1}{1+2*.7+2*.3}$ et $b = \frac{.7}{1+2*.7+2*.3}$ et $c = \frac{.3}{1+2*.7+2*.3}$ l'inversibilité

de notre matrice est équivalente à celle de

$$\begin{array}{cccccc} a & b & c & & & \\ b & a & b & c & & \\ c & b & a & b & c & \\ & c & b & a & b & c \\ & & c & b & a & b \\ & & & ... & ... & ... \end{array}$$

Le calcul du déterminant par récurrence, en développant ce dernier suivant une ligne ou une colonne, est difficile : les matrices successives obtenues sont fréquemment "nouvelles".

Quoi qu'il en soit, cette méthode est à rejeter, en tout cas lorsque le nombre de points à disposition est faible. L'interpolation par "convolution" ne s'adapte vraiment pas bien aux brusques changement dans les donnée. Mais étant donné notre problème (interpolation d'un jeu de donnée à forte pente initiale), il n'y a pas de raison de vouloir moyenner des points avec d'autres surtout lorsque la pente est très forte et que l'effet de bord ne permet pas de compenser à gauche.

Sinon les "splines de lissages" qui reposent sur le même principe fonctionnent très bien.

4.3.6 Interpolation par intégrale

Cette méthode, comme celle de l'exponentielle ou l'idée d'utiliser des courbes de Bézier en "inversant les points" ou de la convolution est de nous. L'idée de cette interpolation-ci repose sur 3 principes : le premier est qu'il n'y a que peu d'interpolations existantes et qu'il faut donc se reposer dessus (Steffen, Bézier) : nos fonctions "faites maisons" ne donnent pas de bons résultats et ne font vraiment pas le poids face à des méthodes qui ont fait leurs preuves, sont mondialement connues, et ont perduré dans le temps. Le second principe est que le lissage est quelque chose généralement introduit par intégration (exemple : la convolution) car l'intégration ajoute un degré de plus à la continuité ou à la dérivabilité. Le troisième principe est qu'en général l'itération ou le passage à la limite apportent souvent des degrés de dérivabilité supplémentaires (comme les itérations de Picard). Par exemple si l'on possède une méthode de lissage, on peut lisser une première fois, puis appliquer cette méthode au résultat obtenu et ainsi de suite.

Notre idée est donc étant donné un jeu de points $A_1, ..., A_n = \{(x_1, y_1), ..., (x_n, y_n)\}$ (les n points fixes à interpoler), d'associer à chacun une dérivée, d'interpoler entre ces dérivées par Stefen (qui donne un résultat toujours satisfaisant) puis d'intégrer cette courbe dérivée obtenue. Rien n'indique encore une fois que cette manière de procéder permet de retomber sur $A_1, ..., A_n = \{(x_1, y_1), ..., (x_n, y_n)\}$. Il faut donc comme pour Bézier ou la convolution modifier les dérivées en ces points en tâtonnant jusqu'à obtenir une courbe passant par tous les points.

Le résultat obtenu est très encourageant (cf Figure).

Il ressemble en fait à l'interpolation de Steffen, mais notre jeu de données initial ne laisse en fin de compte que peu de liberté aux interpolations.

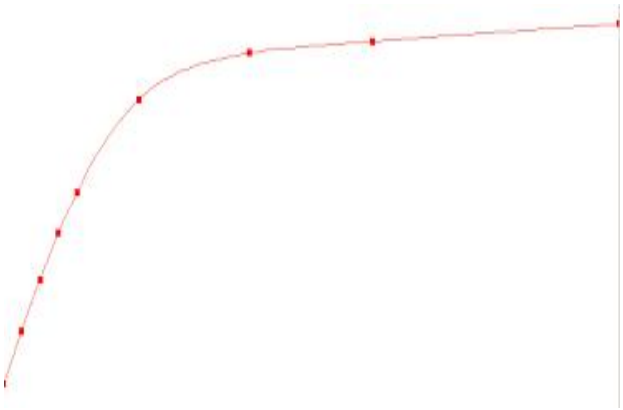


FIG. 34 – Interpolation par intégrale

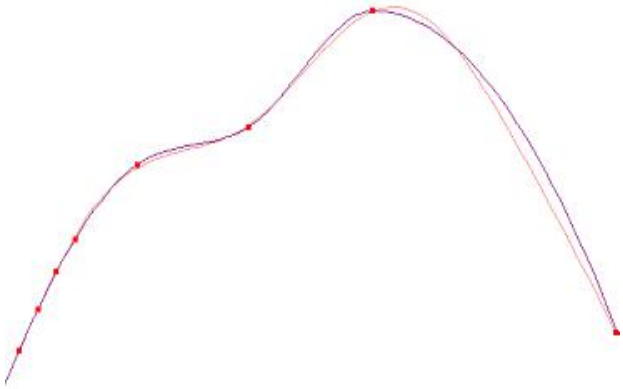


FIG. 35 – Comparaison Steffen - Interpolation par intégrale

Ci-dessous un comparatif avec l'interpolation de Steffen : (Steffen est en violet, la notre en orange)

On peut noter que notre interpolation possède une certaine "inertie" contrairement à Steffen. Ainsi au maximum de ce dernier graphique, Steffen replonge directement, puisque son interpolation ne donne que des courbes entre les points. Avec l'intégrale, la courbe continue de monter encore un peu après le point. C'est à notre avis un avantage : il n'y a aucune raison pour qu'une interpolation qui passe par des points doit modifier sa pente en ces points là exactement.

L'inconvénient de cette méthode est comme toujours lorsqu'il faut résoudre un système numérique son imprécision. Ainsi sur des jeux de données "gentils", l'interpolation atteint exactement les points (de même que Bézier et consorts) tandis que si on injecte plus de "pics" dans les données, notre interpolation ne suit pas. Mais ceci

est peut être dû au fait que l'on a que 100 points sur tout notre graphique. Si l'on s'autorise un maillage toujours plus fin il y a fort à parier que l'on puisse parvenir à nos fins.

Implémentation : nous utilisons la même fonction d'optimisation : choix initial des dérivées en les points fixes, interpolation de Steffen sur ces points, puis calcul de la fonction intégrande résultante, et calcul de la "distance aux points à atteindre" que l'on itère en modifiant les dérivées en les points clé, jusqu'à tomber sous une barre de ϵ.

L'intégration se fait basiquement, en prenant pour ordonnée de départ l'ordonnée du premier point par lequel on doit passer, puis en sommant les $\Delta x.y'(x)$.

Concernant le choix initial des y'_i on décide de prendre pour pentes entre les points : $y'_i = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$ (et $y'_n = y'_{n-1}$).

On pourrait pousser le vice à interpoler par la méthode intégrale entre les dérivées (plutôt qu'avec Steffen) (interpolation qui utilisera l'interpolation de Steffen) et ainsi gagner encore un cran sur la dérivabilité. Cependant on ne l'a pas fait pour des questions de temps dans un premier temps (interpoler une fois par l'intégrale est tolérable puisque derrière Steffen est en temps constant, mais interpoler plusieurs fois implique un temps puissance 2 puisque l'interpolation intégrale nécessite d'inverser un système numériquement et donc d'itérer en recalculant des interpolations intégrale.

Conclusion Concernant les interpolations, cette dernière nous semble finalement la plus agréable : l'interpolation de Steffen fonctionne très bien, mais elle a été développée surtout pour bien réagir face aux données "monotones". Lorsque les données ne le sont plus, elle ne donne pas de formes très naturelles contrairement à l'interpolation intégrale que l'on a développé : celle-ci bénéficie en fait d'une sorte "d'inertie" qui lisse plus les disparités et qui permet de différencier en général les points de changement de monotonie avec les points "fixes" par lesquels on doit passer.

Les interpolations pures et simples donnent de bien meilleurs résultats que notre optimiseur numérique de fonction : celui-ci permet effectivement de faire diminuer le "défaut" de spread par rapport aux critères que l'on impose, mais le résultat obtenu n'est jamais acceptable puisque l'optimiseur crée naturellement des "sauts" à certains endroits afin d'être pénalisé quelque part pour ne pas être pénalisé à beaucoup d'autres endroits.

5 L'optimisation

Nous avons tenté d'optimiser la courbe de spread pour la rendre non arbitrable grâce à un algorithme se basant sur 2 fonctions principales : une première procédure modifie une fonction faiblement, et une seconde teste la fonction obtenue afin de déterminer si cette dernière est plus acceptable qu'originellement.

Ces deux procédures offrent cependant leur lot de surprises que nous détaillons.

5.1 Algorithme d'optimisation

L'algorithme s'était dans un premier temps basé sur la méthode du recuit-simulé : la procédure de test renvoie un nombre représentant le "défaut" par rapport au caractère arbitrable de la courbe de spread. Partant d'une solution donnée, en la modifiant, on en obtient une seconde. Soit celle-ci améliore le critère que l'on cherche à optimiser, on dit alors qu'on a fait baisser l'energie du système, soit celle-ci le dégrade. Si on accepte une solution améliorant le critère, on tend ainsi à chercher l'optimum dans le voisinage de la solution de départ. L'acceptation d'une "mauvaise" solution permet alors d'explorer une plus grande partie de l'espace de solution et tend à éviter de s'enfermer trop vite dans la recherche d'un optimum local.

Tout ceci se fait avec une température qui décroit continûment avec le nombre d'étapes et tend vers 0.

A chaque itération de l'algorithme une modification élémentaire de la solution est effectuée. Cette modification entraîne une variation ΔE de l'énergie du système (toujours calculée à partir du critère que l'on cherche à optimiser). Si cette variation est négative (c'est-à-dire qu'elle fait baisser l'énergie du système), elle est appliquée à la solution courante. Sinon, elle est acceptée avec une probabilité $e^{-\frac{\Delta E}{T}}$ de sorte que lorsque T tend vers 0 les changements ne sont presque plus acceptés, alors qu'initialement, il y a une plus grande liberté.

On itère ensuite selon ce procédé en gardant la température constante.

Le seul problème est que la décroissance de T est très difficile à gérer. Dans tous nos cas nous n'avons obtenu que des processus faisant augmenter l'énergie, ce qui n'est pas normal. Nous avons donc décidé d'arrêter cette méthode et de n'accepter que les solutions faisant décroître l'énergie.

5.1.1 Modification locale de fonction

Concernant la modification, il y a là aussi plusieurs manières de faire : dans le cas d'une fonction cherchée croissante, on peut être un peu plus "précis" concernant la modification.

Nous avons souvent 100 points (x_i, y_i). Nous décidons donc de tirer au sort i puis de modifier y_i.

Une première manière est de choisir y_i aléatoirement entre y_{i-1} et y_{i+1}. Cela conserve la croissance de la courbe.

Une seconde manière de faire est de multiplier y_i par $0.995 + 0.01U$ où U est uniforme sur [0-1]. Cela permet de modifier localement la fonction de 0.5% environ et c'est cette méthode qui fonctionne le mieux.

Une troisième manière de faire est d'agir de même avec les pentes : $d_i = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$. On choisit de modifier la pente d'un certain coefficient : $y_i = y_{i-1}+(0.995+0.01U)(x_i - x_{i-1})d_i$.

La modification n'est pas ce qui pose le plus de problèmes quoique dans notre cas, on souhaite obtenir une courbe de corrélation croissante.

5.1.2 Critère d'évaluation

Nous avons mis en exergue deux critères d'arbitrage : la non-décroissance du spread de tranchelette, et la négativité de celui-ci.

A ceci nous pouvons rajouter que nous souhaitons obtenir une courbe la plus lisse possible.

Il y a encore plusieurs manières d'évaluer le défaut d'une courbe de spread donnée.

Par exemple on peut sommer toutes les valeurs absolues des fois où le spread est négatif. Ou bien le carré de cela, ou ...

Par rapport à la non-décroissance, si le spread sur [a, a+dx] est inférieur à celui sur [b, b+dx] avec $a < b$, alors il y a arbitrage. Il faudrait donc ajouter tous les arbitrages possibles s'il y a une faible valeur de spread sur tranchelette à un endroit.

Ainsi on peut ajouter aux valeurs absolues des spreads négatifs la somme $\sum_i \sum_{j>i}(spread(j) - spread(i))^+$ ou bien le carré de ces nombres.

On peut aussi ne pas regarder sur tous les j supérieurs à i, mais juste sur certains immédiatement supérieurs, de manière à ce que s'il y a une petite erreur on ne soit pas trop pénalisés sur toute une série.

Concernant le critère de "lissitude" de la courbe, nous choisissons d'ajouter au défaut la valeur $\sum_i((2s_i - s_{i+1} - s_{i-1})^+)^2$, puisque la courbe de spread tracée semble être convexe. On pénalise donc la décroissance.

Il y a donc A que l'on peut imputer à la négativité du spread, B à la non décroissance, et C à la non lissitude.

La difficulté consiste ensuite à chercher des coefficients α, β, γ et à retourner $\alpha A + \beta B + \gamma C$. C'est à dire qu'il faut "doser" les quantités calculées pour retourner quelque chose de cohérent.

Paramètre de comptage de croissance du spread Simulations de smile de corrélation /courbe de spread optimisé en prenant originellement l'interpolation linéaire.

On s'intéresse au nombre de "j" à prendre en compte après un indice i. Dans l'ordre avant optimisation, puis 1, 5, 10 et 20 (sur 100 indices au total)3

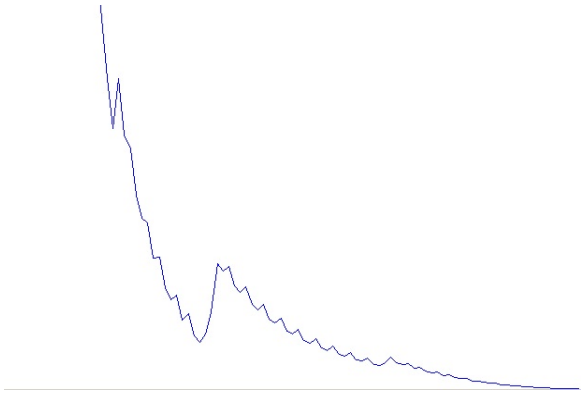


FIG. 36 – Interpolation linéaire avant optimisation numérique

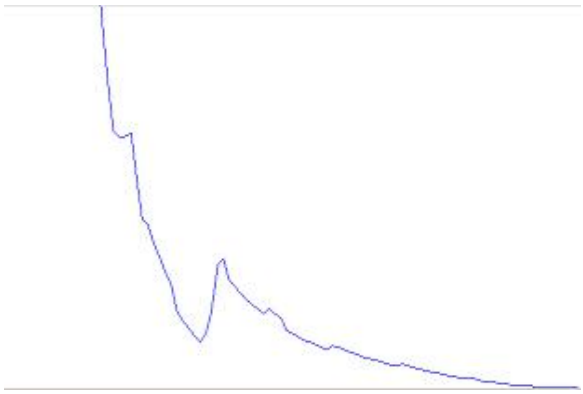


FIG. 37 – Interpolation linéaire après optimisation numérique. Croissance comptée sur 1 suivant

FIG. 38 – Interpolation linéaire après optimisation numérique. Croissance comptée sur 5 suivants

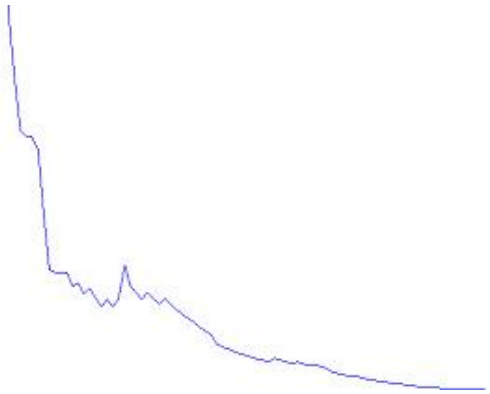


FIG. 39 – Interpolation linéaire après optimisation numérique. Croissance comptée sur 10 suivants

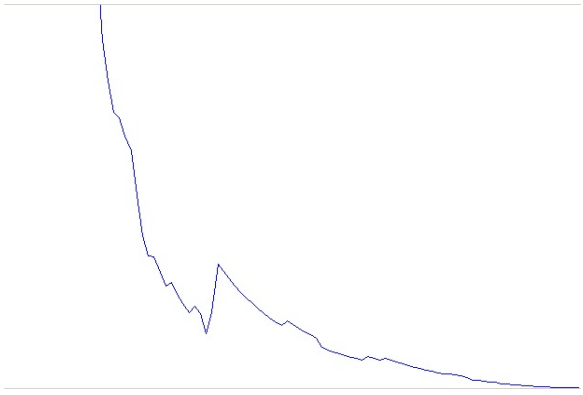


FIG. 40 – Interpolation linéaire après optimisation numérique. Croissance comptée sur 20 suivants

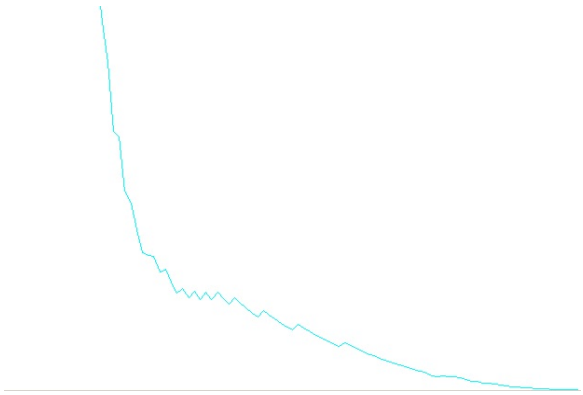


FIG. 41 – Interpol. de Steffen après optimisation numérique. Croissance comptée sur 10 suivants, norme 1

On voit que lorsque l'on prend 1 par exemple, l'optimiseur préfère sacrifier un seul endroit de la courbe en créant une grosse croissance, tandis qu'avec des plus grands nombres ceci ne "passe plus" puisque un saut est compté 5 fois, 10 fois, 20 fois. (pour chaque point avant le saut, soit 1+2+3+4+5 fois, 1+2+3+...+10 fois, 1+2+...+20 fois)

Par contre prendre en compte les 10 ou 20 voisins suivant sur un point donné n'a pas beaucoup d'influence.

Choix de la norme Intéressons nous maintenant au choix de la norme : "norme 1" ou "norme 2". Les images précédentes ont été faites avec une optimisation numérique qui se basait sur une interpolation linéaire avec une norme 1.

La norme 1 ou 2 traduit le fait que l'on prend la valeur absolue ou le carré de l'écart entre un spread et un autre sensé être inférieur (compté seulement si ce dernier est supérieur).

Nous choisissons ici l'interpolation de Steffen avec la norme 1 (5 puis 10 "suivants") puis pour la norme 2 (même nombres).

Sur ces graphiques on se rend compte que le "nombre de suivants" est finalement pas très important tant qu'il est supérieur à 5.

Par contre la norme 1 donne de meilleurs résultats que la norme 2 : contrairement à ce qui est prévu celle-ci lisse la courbe sur de grandes parties mais privilégie les "sauts" brusques.

5.2 Résultats

La courbe de spread est en fait très sensible à la corrélation et si l'optimiseur modifie la courbe de spread on ne voit pas forcément beaucoup de changements sur la

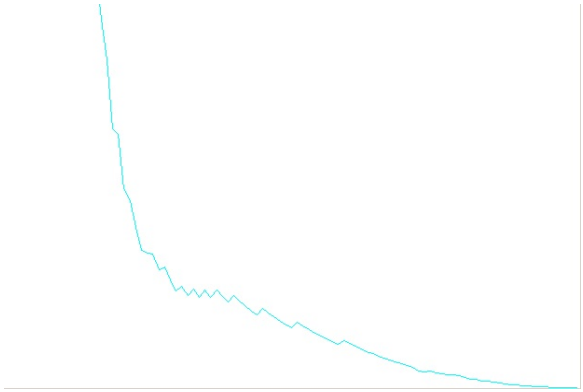


FIG. 42 – Interpol. de Steffen après optimisation numérique. Croissance comptée sur 20 suivants, norme 1

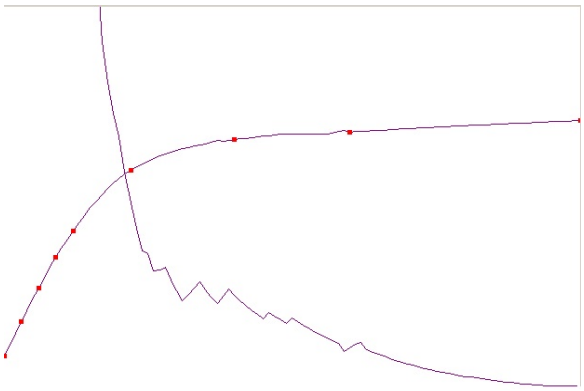


FIG. 43 – Interpol. de Steffen après optimisation numérique. Croissance comptée sur 10 suivants, norme 2

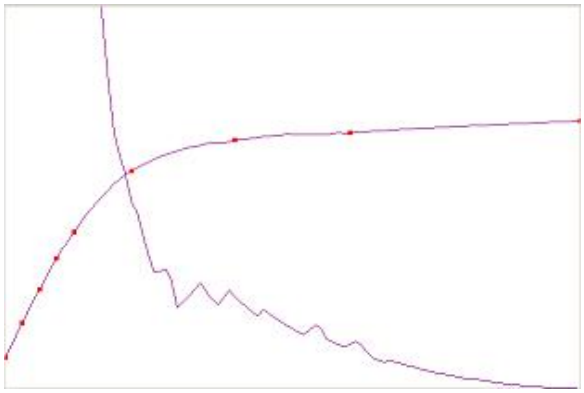


FIG. 44 – Interpol. de Steffen après optimisation numérique. Croissance comptée sur 20 suivants, norme 2

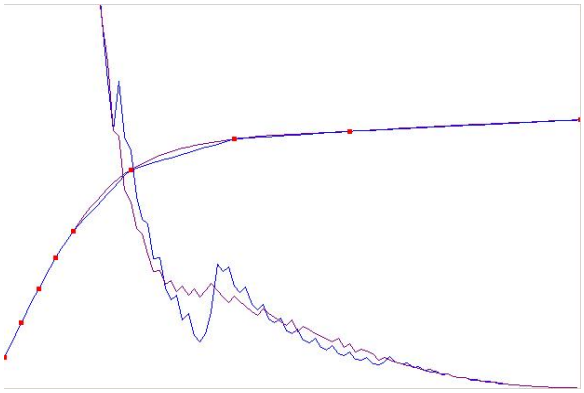


FIG. 45 – Comparaison entre l'interpolation de Steffen (violet) et l'interpolation linéaire (bleu)

corrélation. L'optimiseur ne permet donc absolument pas d'expliciter une "fonction" : on lui donne une interpolation de base, et il la modifie très légèrement. Mais en partant de l'interpolation linéaire on ne tombe pas du tout sur les autres interpolations meilleures ! La convergence est très rapide, et la complexité d'une bonne fonction d'évaluation est la clé du problème de non-convergence vers de meilleures fonctions.

Un intérêt cependant : il y a un résultat chiffré (l'évaluation du défaut, que l'on espère minimale) et l'on peut donc comparer différentes interpolations entre elles grâce à ce nombre. C'est donc un critère objectif.

Voici différents graphiques obtenus. Rappelons que la courbe plus ou moins décroissante sur les figures est la courbe de spread, que l'on cherche à rendre totalement décroissante.

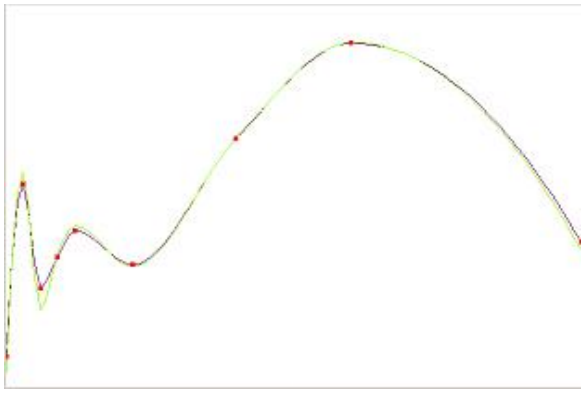


FIG. 46 – Comparaison entre l'interpolation de Steffen (violet) et l'interpolation par convolution (vert)

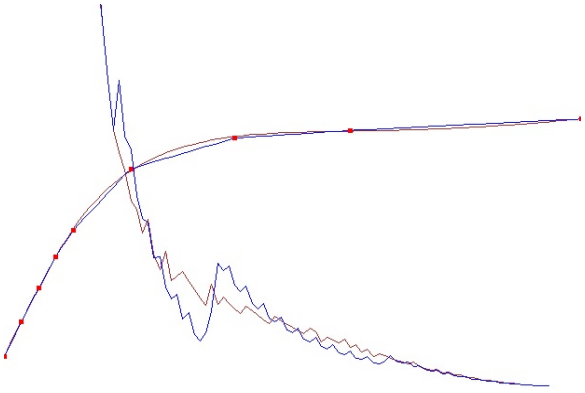


FIG. 47 – Comparaison entre l'interpolation de Bézier (marron) et l'interpolation linéaire (bleu)

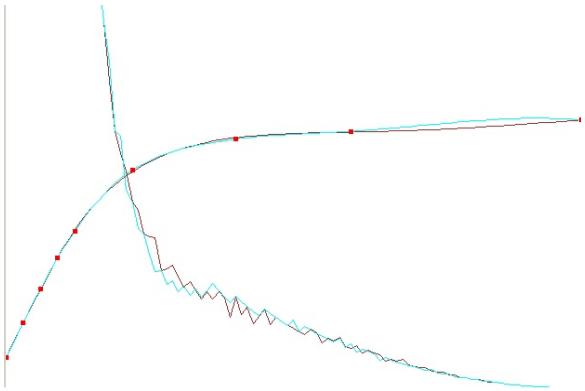


FIG. 48 – Comparaison entre l'interpolation de Steffen (cyan) et l'interpolation de Bézier (marron)

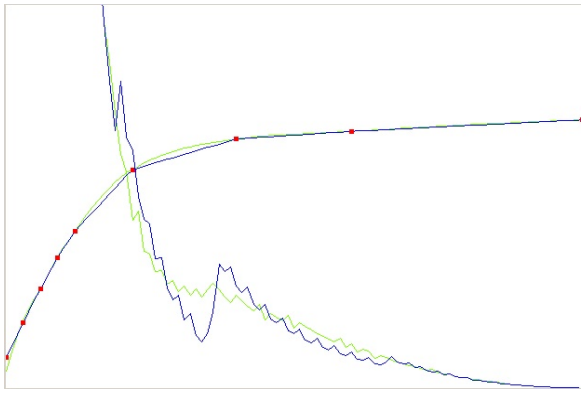


FIG. 49 – Comparaison entre l'interpolation par convolution (vert) et l'interpolation linéaire (bleu)

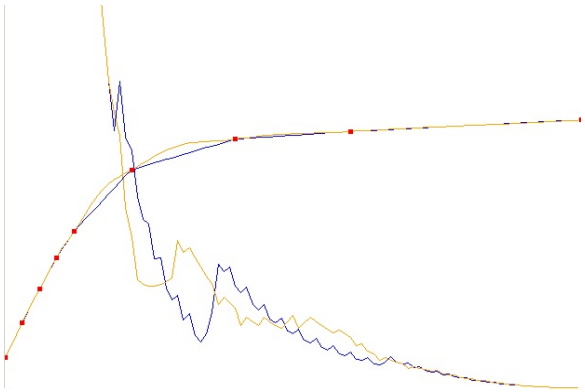


FIG. 50 – Comparaison entre l'interpolation exponentielle (orange) et l'interpolation linéaire (bleu)

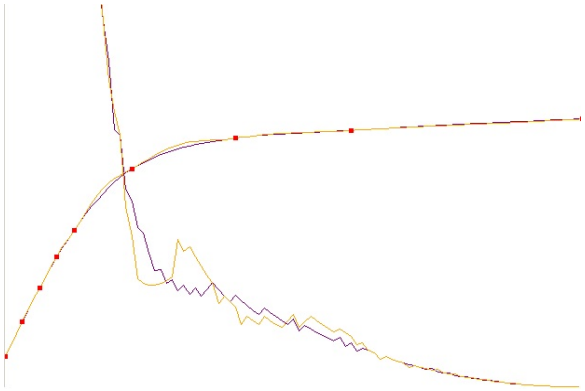


FIG. 51 – Comparaison entre l'interpolation de Steffen (violet) et l'interpolation exponentielle (orange)

5.3 Conclusion

Ces images sont l'occasion de montrer que les courbes interpolarices les plus lisses sont effectivement celles qui donnent lieu aux courbes de spread les plus harmonieuses.

A notre sens seules les interpolation de Bézier, par intégrale et de Steffen sont à retenir. Les deux premières donnent également d'excellentes courbes de tendance dans les cas où l'on opte pour une interpolation "inexacte", qui ne passe pas exactement par les points voulus.

L'interpolation de Steffen est sans surprises puisque sa consistance se prouve mathématiquement. Elle interpole très bien les données monotones (ce que l'on a ici) cependant son comportement n'est pas satisfaisant lorsque les données ne le sont plus. Elle reste néanmoins non divergente mais toujours trop rigide dans ce cas.

L'interpolation par intégrale semble réaliser une alternative prometteuse à Bézier et à Steffen : elle interpole très bien les données monotones et affiche de jolies courbures sur les données non monotones comparée à Steffen. Elle diverge cependant un petit peu comme pour Bézier lorsque les données à interpoler sont trop disparates.

L'interpolation exponentielle reste quant à elle à creuser.

Troisième partie

Extrapolation du smile de corrélation

Cette partie est confidentielle.

A Preuves mathématiques

A.1 Interpolation exponentielle

Afin de choisir une fonction (pour l'interpolation exponentielle) qui vérifie $f(0) = 0, f(1) = 1$ et $\forall n \geq 1, f^{(n)}(0) = f^{(n)}(1) = 0$ on a choisit de se baser sur le résultat suivant :

f : $\left\{ \begin{array}{c} \mathbb{R} \to \mathbb{R} \\ x \to 0 \text{ si } x \leq 0 \\ x \to e^{-\frac{1}{x}} \text{ sinon} \end{array} \right\}$ est C^∞. Il faut vérifier en 0 que toutes les dérivées de la fonction coté "strictement positif" vaillent 0.

Preuve Hypothèse de récurrence en n : $\exists P_n, Q_n \epsilon \mathbb{R}[X]$ tels que $f^{(n)}(x) = \frac{P_n(x)}{Q_n(x)} e^{-\frac{1}{x}}$ (quand $x > 0$)

Hypothèse vraie en 0 ($P_0 = 1 = Q_0$)

$n \to n+1$: $f^{(n)}(x) = \frac{P_n(x)}{Q_n(x)} e^{-\frac{1}{x}} => f^{(n+1)}(x) = ((\frac{P_n(x)}{Q_n(x)})' + \frac{1}{x^2}\frac{P_n(x)}{Q_n(x)})e^{-\frac{1}{x}}$ donc le résultat est prouvé et la récurrence se propage. (prendre par exemple $P_{n+1}(x) = P_n Q_n + X^2(P'_n Q_n - Q'_n P_n)$ et $Q_{n+1} = X^2 Q_n^2$)

Une fois ce résultat prouvé, il est facile de voir que $\lim_{x \to 0^+} \frac{P_n(x)}{Q_n(x)} e^{-\frac{1}{x}} = 0$, par exemple en prenant le logarithme et en voyant qu'il tend vers $-\infty$. ($\ln(x)$ est négligeable devant $\frac{1}{x}$ en 0) ∎

f est donc une fonction C^∞, où toutes les dérivées sont nulles en 0, et qui vaut 0 en 0 et 1 à l'infini. (Et toutes les dérivées tendant naturellement vers 0 à l'infini).

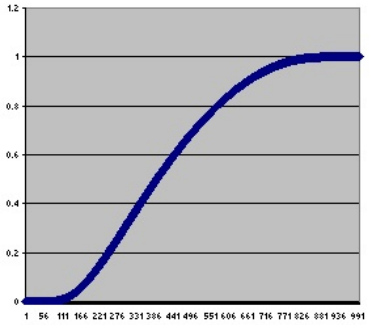
Il faut "réduire" f pour la faire "tenir" dans [0,1], tout en respectant les contraintes.

Pour cela on considère $g(x) = f(x.e^x)$. Cette fonction a vocation a tendre vers l'infini très rapidement afin d'avoir rapidement toutes les dérivées nulles vers l'infini, de sorte à ce qu'une "compression" de $\mathbb{R}^+ \to [0,1]$ ne crée pas de dérivées non nulles en 1.

En considérant la preuve par récurrence précédente, on peut noter que pour $n \geq 1, \exists\ a_n \geq 2$ tel que $f^{(n)}(x) \sim \frac{1}{x^{a_n}}$ quand $x \to +\infty$ (en effet le degré de Q_n augmente plus que celui de P_n)

Cependant $x \to x.e^x$ a toutes ses dérivées du type $P(x)e^x$ où P est un polynôme. Donc $g^{(n)}(x) = e^{-(a_n-1)x}(1 + o(e^{-(a_n-1)x}))$ en l'infini. En 0 f a toutes ses dérivées nulles tandis que $x \to x.e^x$ ne présente pas de "soucis" : toutes ses dérivées sont finies en 0. g a donc aussi toutes ses dérivées nulles en 0.

Quand $x \to 1^-$, $\tan(\frac{\pi}{2}x) = \frac{\sin(\frac{\pi}{2}x)}{\cos(\frac{\pi}{2}x)} \sim \frac{1}{\frac{\pi}{2}(1-x)}$ donc $h(x) = g(\tan(\frac{\pi}{2}x))$ a encore toutes ses dérivées nulles en 0 (puisque les dérivées en 0 de tan ne posent pas de soucis particulier et que dans g il y a encore du $e^{-\frac{1}{x}}$ en facteur) et idem en l'infini (dû au $e^{-(a_n-1)x}$ en facteur dans g tandis que tan a des dérivées équivalentes à des fractions rationnelles)



On obtient ainsi le résultat :
Afin de symétriser la fonction h obtenue, on peut poser $\widehat{h}(x) = \frac{h(x)+1-h(1-x)}{2}$.

A.2 Market Law of Losses

Soi $\overline{L_K}$ l'espérance de la perte sur la tranche [0, K%] vue par le marché, et $L(K)$ la probabilité vue par le marché pour que la perte atteigne K.
$\overline{L_K} = \int_0^K x dL(x) + K(1 - L(K))$, par définition
Ceci se réécrit $\overline{L_K} = K - \int_0^K L(x)dx$ et donc $\frac{\partial \overline{L_K}}{\partial K} = 1 - L(K)$

Soit maintenant $L(K, \rho_K)$ la probabilité que la perte soit inférieure à K étant donnée une corrélation ρ_K et calculée grâce au modèle de la copule gaussienne.
Alors $\overline{L_K} = \int_0^K x dL(x, \rho_K) + K(1 - L(K, \rho_K))$ puisque la base correlation sur la

tranche [0, K%] est faite pour que justement on ait que la perte calculée par la copule gaussienne soit égale à celle du marché.
Soit $\overline{L_K} = K - \int_0^K L(x, \rho_K)dx$
Ainsi $\frac{\partial \overline{L_K}}{\partial K} = 1 - L(K, \rho_K) - \frac{\partial \rho_K}{\partial K} \int_0^K \frac{\partial L}{\partial \rho}(x, \rho_K)dx$
ou encore $\frac{\partial \overline{L_K}}{\partial K} = 1 - L(K, \rho_K) - Skew(K, \rho_K) \int_0^K \frac{\partial L}{\partial \rho}(x, \rho_K)dx$

Mais si l'on définit $Rho(K, \rho_K) = \frac{\partial \overline{L_K}}{\partial \rho_K}$, on a
$Rho(K, \rho_K) = -\int_0^K \frac{\partial L}{\partial \rho}(x, \rho_K)dx$
En combinant les 2 expressions que l'on a, on retrouve :

$$L(K) = L(K, \rho_K) - Skew(K, \rho_K) * \rho(K, \rho_K)$$

soit $P(Loss < K) = P(Loss < K, \rho_K) - Skew(K, \rho_K) * Rho(K, \rho_K)$

A.3 Simulation d'une variable aléatoire normale centrée réduite

L'ordinateur peut fournir des nombres (pseudos-)aléatoires de manière uniforme et indépendante compris entre 0 et 1. C'est ce que l'on suppose, et en pratique le générateur pseudo-aléatoire des ordinateurs tient bien ses promesses.

Si U_1 et U_2 sont 2 variables aléatoires indépendantes et uniformes sur [0, 1] alors si $X = \sqrt{-2ln(U_1)}\cos(2\pi U_2)$, $X \sim N(0,1)$.

La preuve se fait en 2 temps.

Lemme III.1 *Si $X, Y \sim N(0,1)$ et sont indépendantes, si R et θ sont tels que $X = R\cos\theta, Y = R\sin\theta$ avec R positif, alors θ est uniforme sur $[0, 2\pi]$ et R^2 suit une loi exponentielle de paramètre $\frac{1}{2}$.*

Preuve On rappelle que si $X \sim N(0,1)$ alors $E(e^{itX}) = e^{-\frac{t^2}{2}}$.

Fixons α et intéressons nous au couple $(X\cos\alpha + Y\sin\alpha, X\sin\alpha - Y\cos\alpha)$.

Sa fonction caractéristique vaut $E(e^{it(X\cos\alpha+Y\sin\alpha)+iu(X\sin\alpha-Y\cos\alpha)})$

$= E(e^{i(t\cos\alpha+u\sin\alpha)X+i(t\sin\alpha-u\cos\alpha)})$

$= e^{-\frac{(t\cos\alpha+u\sin\alpha)^2}{2}} e^{-\frac{(t\sin\alpha-u\cos\alpha)^2}{2}}$, par indépendance de X et de Y.

C'est à dire après développement $e^{-\frac{t^2+u^2}{2}}$ qui est la fonction caractéristique de (X, Y). Ainsi par rotation d'un angle α la loi du couple ne change pas. C'est donc que θ suit une loi uniforme sur $[0, 2\pi]$.

Quant à R^2, $E(e^{itR^2}) = E(e^{it(X^2+Y^2)})$

$= \frac{1}{2\pi} \iint_{\mathbb{R}^2} e^{it(x^2+y^2)} e^{-\frac{x^2}{2}} e^{-\frac{y^2}{2}} dxdy$

On effectue un changement de variable polaire. Il vient :

$E(e^{itR^2}) = \frac{1}{2\pi} \int_0^{2\pi} \int_0^{+\infty} e^{itr^2} e^{-\frac{r^2}{2}} r dr d\theta$

$= \int_0^{+\infty} e^{itr^2} e^{-\frac{r^2}{2}} r dr = \frac{1}{1-2it}$ (pas de difficultés)

Or si A suit une loi exponentielle de paramètre λ, $f(x) = \lambda e^{-\lambda x}$ et sa fonction caractéristique est $E(e^{itA}) = \int_0^{+\infty} e^{itx}\lambda e^{-\lambda x} dx = \frac{\lambda}{\lambda - it}$.

D'où le résultat. ∎

Passons maintenant au résultat final. On prouve en fait que $\sqrt{-2\ln U_1}\cos(2\pi U_2)$ et $\sqrt{-2\ln U_1}\sin(2\pi U_2)$ sont normales centrées réduites et indépendantes.

Preuve

Rappelons que si F désigne la fonction de répartition d'une variable aléatoire réelle X, et si U est uniforme sur [0, 1] alors $F^{-1}(U) \sim X$ (en effet, $P(X < a) = F(a) = P(U <= F(a)) = P(F^{-1}(U) < a)$)

Si X est une variable aléatoire qui suit une loi exponentielle de paramètre $\frac{1}{2}$, $f(x) = \frac{1}{2}e^{-\frac{1}{2}x}$ et $F(x) = \int_0^x f(t)dt = 1 - e^{-\frac{1}{2}x}$ ainsi si U est uniforme sur [0, 1], $-2\ln(1-U) \sim \exp_{\frac{1}{2}}$ et donc comme U a même loi que $1-U$, $-2\ln U \sim \exp_{\frac{1}{2}}$.

Ainsi $(\sqrt{-2\ln U_1}\cos(2\pi U_2), \sqrt{-2\ln U_1}\sin(2\pi U_2))$ a même loi qu'un couple de 2 variables normales centrées réduites indépendantes. ∎

On aurait pu prouver directement par changement de variable le résultat, mais il semble plus compréhensible de détailler de cette manière.

A.4 Loi normale

Nous utilisons souvent dans nos pricers la fonction $N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{x^2}{2}}dx$. Seulement il faut pouvoir implémenter cette fonction, et que ceci se fasse rapidement vu le nombre de fois où nous allons l'utiliser.

Une solution est par exemple donnée dans le cours de Gobet-El Karoui :

$N(x) \approx 1 - \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}(ay + by^2 + cy^3)$

où $y = \frac{1}{1+.33267x}$, $a = 0.4361836$, $b = -0.1201676$ et $c = 0.937298$

Nous utilisons cette formule pour $x \geq 0$. Pour $x < 0$, $N(x) = 1 - N(-x)$

Pour calculer N^{-1}, nous utilisons la dichotomie en majorant le résultat obtenu à $x = 6$ puisqu'il faut bien prendre une limite.

A.5 Loi normale bi-dimensionnelle

Si X et Y sont deux variables aléatoires normales centrées réduites, de corrélation ρ, on note :

$N(a, b, \rho) = P(X < a, Y < b) = \int_{-\infty}^{a}\int_{-\infty}^{b} \frac{1}{2\pi\sqrt{1-\rho^2}}\exp(-\frac{1}{2}\frac{x^2-2\rho xy+y^2}{1-\rho^2})dxdy$

Une manière d'approximer cette intégrale est décrite dans Hull, 1993, chapitre 10.

```
Public Function N2(ByVal a As Double, ByVal b As Double, ByVal rho As Double) As Double
Dim aa As Variant, bb As Variant
Dim aprim As Double, bprim As Double, summ As Double
Dim i As Integer, j As Integer, delta As Double, rho1 As Double, rho2 As Double, denum As Double

If a <= 0 And b >= 0 And rho <= 0 Then
aprim = a / Math.Sqr(2# * (1# - rho * rho))
bprim = b / Math.Sqr(2# * (1# - rho * rho))
aa = Array(0.325303, 0.4211071, 0.1334425, 0.006374323)
bb = Array(0.1337764, 0.6243247, 1.3425378, 2.2626645)
summ = 0#
For i = 0 To 3
```

```
For j = 0 To 3
summ = summ + aa(i) * aa(j) * f(bb(i), bb(j), aprim, bprim, rho)
Next j
Next i
summ = summ * Math.Sqr(1# - rho * rho) / pi
N2 = summ
Exit Function

ElseIf (a * b * rho) <= 0 Then
If a <= 0 And b >= 0 And rho >= 0 Then
N2 = N1(a) - N2(a, -b, -rho)
Exit Function
ElseIf a >= 0 And b <= 0 And rho >= 0 Then
N2 = N1(b) - N2(-a, b, -rho)
Exit Function
ElseIf a >= 0 And b >= 0 And rho <= 0 Then
N2 = N1(a) + N1(b) - 1# + N2(-a, -b, rho)
Exit Function
End If

Else
denum = Math.Sqr(a * a - 2# * rho * a * b + b * b)
rho1 = (rho * a - b) * sign(a) / denum
rho2 = (rho * b - a) * sign(b) / denum
delta = (1# - sign(a) * sign(b)) / 4#
N2 = N2(a, 0#, rho1) + N2(b, 0#, rho2) - delta
Exit Function
End If

End Function
```

B Implémentation des fonctions

B.1 Pricer de CDS

Comme indiqué dans la partie Pricer de CDS, nous créons une fonction qui aux lambdas calcule le spread résultant, et nous calculons les lambdas successivement par dichotomie. Le code est en VBA, mais il se généralise très facilement dans n'importe quel langage, sauf peut être la première partie où les paramètres du marché sont lus directement depuis la feuille Excel. On suppose donnés le spread d'un CDS sur toutes les maturités de 1 à 10.

Le tableau des lambdas (intensités de défaut par unité de temps) ainsi que d'autres paramètres (recovery, taux d'intérêt, ...) sont ici des variables globales.

```
Option Explicit
```

```
Dim r As Double, recovery As Double, rbpv As Double
Dim spreads(1 To 10) As Double, lambdas(1 To 10) As Double
' initialisation des données utilisateur

' puis calcul des lambdas (intensités de défaut par unité de temps)
Private Sub btncompute_Click()
Dim k As Integer
r = CDbl(Me.Range("MarketData").Cells(2, 2))
recovery = CDbl(Me.Range("MarketData").Cells(3, 2))
For k = 1 To 10
spreads(k) = CDbl(Me.Range("MarketData").Cells(6 + k, 2))
Next k
For k = 1 To 10
lambdas(k) = impliciter_lambda(k)
Me.Range("MarketData").Cells(6 + k, 3) = lambdas(k)
Next k
End Sub

'k indique sur quelle plage impliciter lambda (0 à 1 an, 1 à 2 ans, ...)
Private Function impliciter_lambda(ByVal k As Integer) As Double
Dim a As Double, b As Double, s As Double
a = 0
b = 1
Do While (b - a) > 0.00000001
lambdas(k) = (a + b) / 2
s = calcule_spread(CDbl(k))
If s < spreads(k) Then
a = (a + b) / 2
Else
b = (a + b) / 2
End If
Loop
impliciter_lambda = (a + b) / 2
End Function

Private Function calcule_spread(ByVal tfinal As Double) As Double
Dim s As Double, k As Integer, n As Integer
Dim integr_lambda As Double
Dim temp As Double, l As Double, aa As Double, bb As Double
Dim floatleg As Double
'CALCUL DE LA FLOAT LEG
s = 0#
n = Int(tfinal)
For k = 1 To n
l = lambdas(k)
s = s + (l / (l + r)) * (Math.Exp(-(r + l) * (k - 1))) * (1 - Math.Exp(-(r + l)))
```

```
    Next k
    If (tfinal < 10) And (tfinal > n) Then
    l = lambdas(n + 1)
    s = s + (l / (l + r)) * (Math.Exp(-(r + l) * n)) * (1 - Math.Exp(-(r + l) * (tfinal
- n)))
    End If
    floatleg = s * (1 - recovery)
    'CALCUL DE LA FIXED LEG
    integr_lambda = 1 ' = exp (integrale de lambda de 0 à ti)
    n = Int(4 * tfinal)
    s = 0#
    For k = 1 To n
    l = lambdas(1 + Int((k - 1) / 4))
    temp = integr_lambda * l * Math.Exp(-r * (k - 1) * 4)
    aa = l + r
    temp = temp * ((1 - Math.Exp(-0.25 * aa)) / (aa * aa) - 0.25 * Math.Exp(-0.25
* aa) / aa)
    integr_lambda = integr_lambda * Math.Exp(-0.25 * l)
    s = s + 0.25 * Math.Exp(-r * k / 4) * integr_lambda + temp
    Next k
    If (tfinal < 10) And (tfinal > (n / 4)) Then
    l = lambdas(Int(tfinal) + 1)
    temp = integr_lambda * l * Math.Exp(-r * n * 4)
    aa = l + r
    bb = tfinal - n / 4
    temp = temp * ((1 - Math.Exp(-bb * aa) / (aa * aa)) - bb * Math.Exp(-bb * aa)
/ aa)
    integr_lambda = integr_lambda * Math.Exp(-bb * l)
    s = s + bb * Math.Exp(-r * tfinal) * integr_lambda + temp
    End If
    rbpv = s
    calcule_spread = floatleg / rbpv
    End Function
```

B.2 Pricer de CDO

On a implémenté ici les 3 différentes manières d'écrire un pricer, comme on l'a décrit plus tôt, en VBA toujours. On réutilise le code de pricing de CDS afin de déduire les intensités de défaut lambdas.

Nous utilisons quelques fonctions d'Excel (fonction Beta et Gamma) grâce au snippet : Application.WorksheetFunction.GammaLn(x) (on peut remplacer GammaLn par n'importe quelle fonction d'Excel).

Dans l'ordre de ce code : calcul par récurrence, par loi beta puis par monteCarlo. Chacune de ces fonctions utilise beaucoup de petites fonctions.

Il faut faire attention à quelque chose d'autre : lorsque l'on dit que l'on achète la tranche $[K_1, K_2]$, cela veut dire que lorsque les pertes (en prenant en compte le taux de recouvrement) atteignent $K_1\%$ du nominal on commence à payer.

Ainsi en considérant un taux de recouvrement de 40% par exemple, la tranche $[60\%, 100\%]$ est de fait "fictive" puisque les pertes n'atteindront au pire que 60%.

Par contre la loi Beta est "réglée" sur [0, 1]. Il faut donc effectuer un "changement d'échelle", c'est à dire réajuster la perte afin qu'elle concorde avec les autres algorithmes.

```
Option Explicit
Public recovery(1 To 10) As Double, correl(1 To 10) As Double, leverage(1 To 10) As Double
Public lambdas(1 To 10, 1 To 10) As Double
Public k1 As Double, k2 As Double
Public floatleg As Double, rbpv As Double
Dim pi(1 To 10, 0 To 10) As Double ' pour calculer la loi inverse de TAUi

Public Function calculer_spread_recurrence(tfin As Double) As Double
Dim integr As Double, n As Integer, k As Integer, t As Double, i As Integer, j As Integer, nt As Integer, ligne As Integer
Const nb_integr = 1000, nb_integr2 = 100, nb_integr_t = 20, nb_integr_z = 200
Dim u As Double, grandL As Double, grandN As Integer, z As Double, grandT As Integer, nz As Integer, NTFinal As Integer
Dim levera(1 To 10) As Integer 'levera * u = leverage
Dim table() As Double, temp() As Double, p As Double, nn As Integer
u = leppcm()
grandL = 0#

For i = 1 To 10
grandL = grandL + leverage(i) * (1 - recovery(i))
Next i
k1 = k1 * grandL
k2 = k2 * grandL
grandN = Int((grandL / u) + 0.0001)
grandT = Int(tfin * nb_integr_t)
For i = 1 To 10
levera(i) = Int((leverage(i) * (1 - recovery(i))) / u)
Next i
'création de la table des probabilités
ReDim table(0 To grandT, 0 To grandN) As Double 'contient P(Lt = n*u)
ReDim temp(0 To 1, 0 To grandN) As Double 'contient P(Lt,i = n*u)
initialise_TAUiMoins1
For nt = 0 To grandT
t = tfin * (CDbl(nt) / CDbl(grandT))
For j = 0 To grandN
```

```
    table(nt, j) = 0#
    Next j
    For nz = 1 To nb_integr_z
    z = -6# + 12# * CDbl(nz) / CDbl(nb_integr_z)
    effacer temp, grandN
    temp(0, 0) = 1#
    For i = 1 To 10
    ligne = i Mod 2
    For j = 0 To grandN
    p = proba_condition(i, t, z)
    If j >= levera(i) Then
    temp(ligne, j) = temp(1 - ligne, j) * (1 - p) + temp(1 - ligne, j - levera(i)) * p
    Else
    temp(ligne, j) = temp(1 - ligne, j) * (1 - p)
    End If
    Next j
    Next i
    For j = 0 To grandN
    table(nt, j) = table(nt, j) + temp(0, j) * Exp(-z * z * 0.5) * 12# / nb_integr_z
    Next j
    Next nz
    For j = 0 To grandN
    table(nt, j) = table(nt, j) / Math.Sqr(2 * Tout.pi)
    Next j
    Next nt
    'fin de création de la table

    'floatleg
    floatleg = Math.Exp(-Tout.r * tfin) * espe_recur(grandT, table, grandN, u) 'espe
= E(Lt[k1-k2])
    integr = 0#
    For nt = 1 To grandT
    t = tfin * (CDbl(nt) / CDbl(grandT))
    integr = integr + Math.Exp(-Tout.r * t) * espe_recur(nt, table, grandN, u) * tfin
/ CDbl(grandT)
    Next nt
    floatleg = floatleg + integr * Tout.r

    'fixedleg
    rbpv = 0#
    n = Int(tfin * 4)
    NTFinal = Int(0.25 * CDbl(grandT) / tfin)
    For k = 1 To n
    integr = 0#
    For nt = 0 To (NTFinal - 1)
    t = k / 4# + 0.25 * CDbl(nt) / CDbl(NTFinal)
```

```
    nn = nt + Int(CDbl(k - 1) / (4# * tfin) * CDbl(grandT))
    integr = integr + (k2 - k1 - espe_recur(nn, table, grandN, u)) * 0.25 / NTFinal
    Next nt
    rbpv = rbpv + Math.Exp(-Tout.r * CDbl(k) / 4#) * integr
    Next k
    calculer_spread_recurrence = floatleg / rbpv
    End Function

    Private Function espe_recur(ByVal t As Integer, ByRef table() As Double, ByVal
n As Double, ByVal u As Double) As Double
    Dim l As Double, i As Integer
    l = 0#
    For i = 0 To n
        l = l + table(t, i) * (plus(u * CDbl(i) - k1) - plus(CDbl(i) * u - k2))
    Next i
    espe_recur = l
    End Function

    Public Function calculer_spread_betalaw(tfin As Double) As Double
    Dim integr As Double, n As Integer, k As Integer, t As Double, i As Integer
    Const nb_integr = 1000, nb_integr2 = 100
    initialise_TAUiMoins1
    'floatleg
    floatleg = Math.Exp(-Tout.r * tfin) * espe(tfin) 'espe = E(Lt[k1-k2])
    integr = 0#
    For k = 1 To nb_integr
    t = (tfin * k) / nb_integr
    integr = integr + Tout.r * Math.Exp(-Tout.r * t) * espe(t) * tfin / nb_integr
    Next k
    floatleg = floatleg + integr
    'fixedleg
    rbpv = 0#
    n = Int(tfin * 4)
    For k = 1 To n
    integr = 0#
    For i = 1 To nb_integr2
    t = (k - 1) / 4 + 0.25 * i / nb_integr2
    integr = integr + (k2 - k1 - espe(t)) * 0.25 / nb_integr2
    Next i
    rbpv = rbpv + Math.Exp(-Tout.r * k / 4) * integr
    Next k
    calculer_spread_betalaw = floatleg / rbpv
    'redimensionnement du floatleg
    t = 0#
    For i = 1 To 10
    t = t + leverage(i) * (1 - recovery(i))
```

```
    Next i
    floatleg = floatleg * t
    End Function

    Private Function espe(t As Double) As Double
    espe = espe_plus(k1, t) - espe_plus(k2, t)
    End Function

    Private Function espe_plus(ByVal k As Double, ByVal t As Double) As Double ' = E(Lt-k)+
    Dim i As Integer, j As Integer, l As Double, lbis As Double, mu As Double, sigma2 As Double, chi As Double, s As Double
    Dim a As Double, b As Double
    'moyenne de Lt
    l = 0#
    lbis = 0#
    For i = 1 To 10
    l = l + leverage(i) * (1 - recovery(i))
    lbis = lbis + leverage(i) * (1 - recovery(i)) * proba(i, t) ' = proba(Taui <= t)
    Next i
    mu = lbis / l
    'variance de Lt
    s = 0#
    For i = 1 To 10
    For j = 1 To 10
    If i <> j Then
    a = Tout.N1moins1(proba(i, t))
    b = Tout.N1moins1(proba(j, t))
    s = s + leverage(i) * (1 - recovery(i)) * leverage(j) * (1 - recovery(j)) * Tout.N2(a, b, correl(i) * correl(j))
    Else
    s = s + (leverage(i) * (1 - recovery(i))) ^2 * proba(i, t)
    End If
    Next j
    Next i
    sigma2 = s / (l * l) - mu * mu
    chi = mu * (1 - mu) / sigma2 - 1
    a = mu * chi
    b = (1 - mu) * chi
    espe_plus = -k * (1 - grandF(a, b, k)) + (1 - grandF(a + 1, b, k)) * a / (a + b)
    End Function

    Public Function calculer_spread_montecarlo(tfin As Double) As Double
    Dim i As Integer, k As Long, n As Integer, l As Double
    Dim xi As Double, ii As Double
    Dim correlbis(1 To 10) As Double
```

```
    Dim taui(1 To 2, 1 To 10) As Double
    Dim z As Double, s As Double, t As Double
    Dim dejacalcule As Boolean, dejafloat As Double, dejarbpv As Double, ok As
Boolean
    Const nb_simulations = 100000
    initialise_TAUiMoins1
    floatleg = 0#
    rbpv = 0#
    dejacalcule = False
    l = 0#
    For i = 1 To 10
    correlbis(i) = Math.Sqr(1 - correl(i) * correl(i))
    l = l + leverage(i) * (1 - recovery(i))
    Next i
    'CALIBRAGE des DONNEES ; choix d'une échelle une bonne fois pour toutes
    k1 = k1 * l
    k2 = k2 * l
    For k = 1 To nb_simulations
    z = Tout.gaussienne_reduite()
    ok = False
    For i = 1 To 10

    'calcul des temps de défaut
    xi = z * correl(i) + correlbis(i) * Tout.gaussienne_reduite()
    taui(2, i) = TAUiMoins1(i, Tout.N1(xi))
    taui(1, i) = i
    If taui(2, i) <= tfin Then ok = True
    Next i
    If Not ok And dejacalcule Then GoTo dejacalcul
    trier taui 'par ordre croissant
    'floatleg
    s = 0#
    For i = 1 To 10
    If (taui(2, i) <= tfin) Then
    If i = 1 Then
    s = s + Math.Exp(-Tout.r * taui(2, i)) * rembourser(i, taui)
    Else
    s = s + Math.Exp(-Tout.r * taui(2, i)) * (rembourser(i, taui) - rembourser(i - 1,
taui))
    End If
    Else
    Exit For
    End If
    Next i
    If Not ok Then dejafloat = s
    'fixedleg
```

```
    t = 0#
    n = Int(tfin * 4)
    For i = 1 To n 'i designe cette fois le ieme intervalle de temps
    ii = CDbl(i)
    t = t + Math.Exp(-Tout.r * ii / 4) * ((k2 - k1) / 4 - evaluer_loss(taui, (ii - 1#)
/ 4#, ii / 4#))
    Next i
    t = t + Math.Exp(-Tout.r * tfin) * ((k2 - k1) * (tfin - CDbl(n) / 4#) - eva-
luer_loss(taui, CDbl(n) / 4#, tfin))
    If Not ok Then
    dejarbpv = t
    dejacalcule = True
    End If
    dejacalcul :
    floatleg = floatleg + IIf(ok, s, dejafloat)
    rbpv = rbpv + IIf(ok, t, dejarbpv)
    Next k
    floatleg = floatleg / nb_simulations
    rbpv = rbpv / nb_simulations
    calculer_spread_montecarlo = floatleg / rbpv
    End Function
    Private Function rembourser(i As Integer, taui() As Double) As Double
    Dim k As Integer, l As Double
    For k = 1 To i
    l = l + leverage(taui(1, k)) * (1 - recovery(taui(1, k)))
    Next k
    rembourser = plus(l - k1) - plus(l - k2)
    End Function

    Private Function evaluer_loss(ByRef taui() As Double, deb As Double, fin As
Double) As Double
    Dim i As Integer, l As Double, ltranche As Double, lasti As Integer
    lasti = 0
    l = 0#
    ltranche = 0#
    For i = 1 To 10
    If taui(2, i) < deb Then
    l = l + leverage(taui(1, i)) * (1 - recovery(taui(1, i)))
    lasti = i
    ElseIf taui(2, i) >= deb And taui(2, i) < fin Then
    lasti = i
    If i > 1 Then 'sinon il y a l = 0 !
    ltranche = ltranche + (taui(2, i) - maxi(taui(2, i - 1), deb)) * (plus(l - k1) - plus(l
- k2))
    End If
    l = l + leverage(taui(1, i)) * (1 - recovery(taui(1, i)))
```

```
    ElseIf taui(2, i) >= fin Then
    Exit For
    End If
    Next i
    If lasti >= 1 Then 'sinon il y a l = 0 !
    ltranche = ltranche + (fin - maxi(taui(2, lasti), deb)) * (plus(l - k1) - plus(l - k2))
    End If
    evaluer_loss = ltranche
    End Function

    Public Sub initialise_TAUiMoins1()
    Dim i As Integer, k As Integer
    For i = 1 To 10
    pi(i, 0) = 1
    For k = 1 To 10
    pi(i, k) = pi(i, k - 1) * Math.Exp(-lambdas(i, k))
    Next k
    Next i
    End Sub
    Private Function TAUiMoins1(ByVal i As Integer, ByVal x As Double) 'trouver
T tel que P(Tau <= T) = x
    Dim k As Integer
    For k = 1 To 10
    If x < (1 - pi(i, k)) Then
    TAUiMoins1 = k - 1 - 1 / lambdas(i, k) * Math.Log((1 - x) / pi(i, k - 1))
    Exit Function
    End If
    Next k
    TAUiMoins1 = 11
    End Function
    Public Function proba(ByVal i As Integer, ByVal t As Double) As Double
    Dim j As Integer
    j = Int(t)
    If j < 10 Then
    proba = 1 - pi(i, j) * Math.Exp(-lambdas(i, j + 1) * (t - j))
    Else
    proba = 1 - pi(i, j)
    End If
    End Function

    Public Function proba_condition(ByVal i As Integer, ByVal t As Double, ByVal
z As Double) As Double
    Dim rho As Double
    rho = correl(i)
    proba_condition = Tout.N1((Tout.N1moins1(proba(i, t)) - rho * z) / Math.Sqr(1
- rho * rho))
```

```
End Function
Private Sub effacer(ByRef temp() As Double, ByVal grandN As Integer)
Dim i As Integer
For i = 0 To grandN
temp(0, i) = 0#
temp(1, i) = 0#
Next i
End Sub
Private Function leppcm() As Double
Dim retour As Double, i As Integer
retour = leverage(1) * (1 - recovery(1))
For i = 2 To 10
retour = ppcm(retour, leverage(i) * (1 - recovery(i)))
Next i
leppcm = retour
End Function

Public Function grandF(a As Double, b As Double, k As Double) As Double
'renvoie integr(x^a-1.(1-x)^b-1) ,sur [0-k]/integr(0-1)
If k = 1 Then
grandF = 1
ElseIf k = 0 Then
grandF = 0
ElseIf a > 15 Then
grandF = 0.001
ElseIf b > 15 Then
grandF = 1
Else
grandF = Application.WorksheetFunction.BetaDist(k, a, b)
End If
End Function
Public Function betta(a As Double, b As Double) As Double
'renvoie integr(x^a-1.(1-x)^b-1) sur [0-1]
If a > 10 Or b > 10 Then
betta = 0#
Else
betta = gamma(a) * gamma(b) / gamma(a + b)
End If
End Function

Public Function gamma(x As Double) As Double
gamma = Math.Exp(Application.WorksheetFunction.GammaLn(x))
End Function
```

B.3 Interpolations

Notre logiciel a été écrit en C# et c'est donc dans ce langage que l'on écrit toutes nos interpolations. Encore une fois, le code est essentiellement fait de boucles et de petites conditions. Il est donc aisément transposable dans n'importe quel langage.

Toutes nos fonction d'interpolation ont la même signature : elles prennent en entrée deux tableaux de nombres réels, les abscisses et ordonnées des points à interpoler. Le résultat est un tableau de 101 réels, décrivant les ordonnées de la fonction d'interpolation dont les abscisses sont distribuées régulièrement entre la première et la dernière abscisse.

Ce choix de 101 n'est pas anodin : nous donnons 0 et 1 en première et dernière abscisses et les abscisses d'interpolation sont donc tous les 0.01.

Enfin, la convolution, l'intégrale et Bézier utilisent des fonctions communes : distance() qui mesure la distance (somme des carrés des différences) entre les points d'entrée par lesquels on doit passer, et les points de l'interpolation ; il y a aussi la fonction de modification locale de points clés créant ces différentes interpolation.

B.3.1 Stefen

```
public static double[] interpol_steffen2(double[] abs_depart, double[] ord_depart)
// aux extrémités : choix d'une 2nd degré C1
{
double[] fonction = new double[101] ;
int n = abs_depart.Length ;
double[,] coeffs = new double[n - 1, 4] ; // ax^3+bx^2++cx+d pour 1 à n-3, sinon
ax^2+bx+c pour 0 et n-2
double[] pentes = new double[n] ; // pente en xi. 0 et n-1 ne sont pas remplis
// calcul des pentes en chaque point
for (int i = 1 ; i < (n - 1) ; i++)
{
double si = (ord_depart[i+1]-ord_depart[i])/(abs_depart[i+1]-abs_depart[i]) ;
double simoins1 = (ord_depart[i]-ord_depart[i-1])/(abs_depart[i]-abs_depart[i-
1]) ;
double hi = abs_depart[i + 1] - abs_depart[i] ;
double himoins1 = abs_depart[i] - abs_depart[i - 1] ;
double pi = (simoins1 * hi + si * himoins1) / (himoins1 + hi) ;
if (si * simoins1 <= 0) // signe opposé
pentes[i] = 0.0 ;
else if (Math.Abs(pi) > (2 * Math.Min(Math.Abs(simoins1), Math.Abs(si))))
pentes[i] = 2 * Math.Min(Math.Abs(simoins1), Math.Abs(si)) * ((si > 0) ? 1.0 :
-1.0) ;
else
pentes[i] = pi ;
}
// calcul des coeffs
```

```
    // en 0
    double deltax = abs_depart[0] - abs_depart[1] ;
    coeffs[0, 0] = (ord_depart[0] - ord_depart[1] - deltax * pentes[1]) / (deltax *
deltax) ;
    coeffs[0, 1] = 2.0 * coeffs[0, 0] * deltax + pentes[1] ;
    coeffs[0, 2] = ord_depart[0] ;
    for (int i = 1 ; i < (n - 2) ; i++)
    {
    deltax = abs_depart[i + 1] - abs_depart[i] ;
    double pent = (ord_depart[i + 1] - ord_depart[i]) / deltax ;
    coeffs[i, 0] = (pentes[i] + pentes[i + 1] - 2.0 * pent) / (deltax * deltax) ;
    coeffs[i, 1] = (3.0 * pent - 2.0 * pentes[i] - pentes[i + 1]) / deltax ;
    coeffs[i, 2] = pentes[i] ;
    coeffs[i, 3] = ord_depart[i] ;
    }
    // en n-2
    deltax = abs_depart[n - 1] - abs_depart[n - 2] ;
    coeffs[n - 2, 0] = (ord_depart[n - 1] - ord_depart[n - 2] - deltax * pentes[n - 2]) /
(deltax * deltax) ;
    coeffs[n - 2, 1] = pentes[n - 2] ;
    coeffs[n - 2, 2] = ord_depart[n - 2] ;
    // calcul final : utilisation des coeffs calculés plus tôt
    int ind = 0 ;
    for (int i = 0 ; i < 101 ; i++)
    {
    double x = .01 * (double)(i) ;
    if (x > abs_depart[ind + 1])
    ind++ ;
    double xx = x - abs_depart[ind] ;
    if (ind == 0 || ind == (n - 2))
    fonction[i] = coeffs[ind, 0] * xx * xx + coeffs[ind, 1] * xx + coeffs[ind, 2] ;
    else
    fonction[i] = coeffs[ind, 0] * xx * xx * xx + coeffs[ind, 1] * xx * xx + coeffs[ind,
2] * xx + coeffs[ind, 3] ;
    }
    return fonction ;
    }
```

B.3.2 Exponentielle

```
    // utilise le calcul des pentes de steffen en les points d'entrée
    public static double[] interpol_expon(double[] abs_depart, double[] ord_depart)
    {
    double[] fonction = new double[101] ;
    int n = abs_depart.Length ;
```

```
    double[] pentes = new double[n] ; // pente en xi. 0 et n-1 sont remplis après
    double[,] coeffs = new double[n - 1, 4] ; // ax^3+bx^2++cx+d pour 1 à n-3, sinon
ax^2+bx+c pour 0 et n-2
    // calcul des pentes en chaque point
    for (int i = 1 ; i < (n - 1) ; i++)
    {
    double si = (ord_depart[i+1]-ord_depart[i])/(abs_depart[i+1]-abs_depart[i]) ;
    double simoins1 = (ord_depart[i]-ord_depart[i-1])/(abs_depart[i]-abs_depart[i -
1]) ;
    double hi = abs_depart[i + 1] - abs_depart[i] ;
    double himoins1 = abs_depart[i] - abs_depart[i - 1] ;
    double pi = (simoins1 * hi + si * himoins1) / (himoins1 + hi) ;
    if (si * simoins1 <= 0) // signe opposé
    pentes[i] = 0.0 ;
    else if (Math.Abs(pi) > (2 * Math.Min(Math.Abs(simoins1), Math.Abs(si))))
    pentes[i] = 2 * Math.Min(Math.Abs(simoins1), Math.Abs(si)) * ((si > 0) ? 1.0 :
-1.0) ;
    else
    pentes[i] = pi ;
    }
    double deltax ;
    // calcul des coeffs
    for (int i = 1 ; i < (n - 2) ; i++)
    {
    deltax = abs_depart[i + 1] - abs_depart[i] ;
    double pent = (ord_depart[i + 1] - ord_depart[i]) / deltax ;
    coeffs[i, 0] = (pentes[i] + pentes[i + 1] - 2.0 * pent) / (deltax * deltax) ;
    coeffs[i, 1] = (3.0 * pent - 2.0 * pentes[i] - pentes[i + 1]) / deltax ;
    coeffs[i, 2] = pentes[i] ;
    coeffs[i, 3] = ord_depart[i] ;
    }
    // calcul des pentes en 0 et en n-1
    // en 0
    deltax = abs_depart[0] - abs_depart[1] ;
    double a = (ord_depart[0] - ord_depart[1] - deltax * pentes[1]) / (deltax * deltax) ;
    pentes[0] = 2.0 * a * deltax + pentes[1] ;
    // en n-1
    deltax = abs_depart[n - 1] - abs_depart[n - 2] ;
    a = (ord_depart[n - 1] - ord_depart[n - 2] - deltax * pentes[n - 2]) / (deltax *
deltax) ;
    pentes[n - 1] = pentes[n - 2] + 2 * a * deltax ;

    // calcul final : utilisation des coeffs calculés plus tôt
    int ind = 0 ;
    for (int i = 0 ; i < 100 ; i++)
    {
```

```
double x = .01 * (double)(i);
if (x > abs_depart[ind + 1])
ind++;
double xx = x - abs_depart[ind];
if (ind != 0 && ind != (n - 2))
{
double[] tt = { ord_depart[ind], pentes[ind], coeffs[ind, 1] / 2 };
double[] qq ={ ord_depart[ind + 1], pentes[ind + 1], coeffs[ind+1, 1] / 2};
fonction[i] = f(abs_depart[ind], abs_depart[ind + 1], x, tt, qq);
}
else if (ind == 0)
{
double[] tt = { ord_depart[ind], pentes[ind] };
double[] qq ={ ord_depart[ind + 1], pentes[ind + 1], coeffs[ind + 1, 1] / 2 };
fonction[i] = f(abs_depart[ind], abs_depart[ind + 1], x, tt, qq);
}
else
{
double[] tt = { ord_depart[ind], pentes[ind], coeffs[ind, 1] / 2 };
double[] qq ={ ord_depart[ind + 1], pentes[ind] };
fonction[i] = f(abs_depart[ind], abs_depart[ind + 1], x, tt, qq);
}
}
fonction[100] = ord_depart[n - 1];
return fonction;
}
```

Fonctions annexes à l'interpolation exponentielle

```
private static double ff(double x) // en 0 : vaut 0 et toutes les dérivées nulles.
idem en 1 pour les dérivées, et vaut 1
{
double y = Math.Tan(.5 * Math.PI * x);
if (y == 0.0)
return 0.0;
else
return Math.Exp(-Math.Exp(-y) / y);
}
private static double f(double x) // pour symétriser ff
{
return (.5 + .5 * (ff(x) - ff(1 - x)));
}
public static double f(double a, double b, double x, double[] derivees_en_a, double[]
derivees_en_b)
{
double y = (x - a);
```

```
double s1= 0.0;
double power = 1.0;
double factorielle = 1.0;
for (int i = 0; i < derivees_en_a.Length; i++)
{
s1 += power * derivees_en_a[i] / factorielle;
power *= y;
factorielle *= (double)(i+1);
}
double s2 = 0.0;
double yy = (x - b);
power = 1.0; factorielle = 1.0;
for (int i = 0; i < derivees_en_b.Length; i++)
{
s2 += power * derivees_en_b[i] / factorielle;
power *= yy;
factorielle *= (double)(i + 1);
}
double ff = f((x-a)/(b-a));
return s1 * (1 - ff) + s2 * ff;
}
```

B.3.3 Convolution

```
public static double[] interpol_convolution(double[] abs_depart, double[] ord_depart)
{
int n = abs_depart.Length;
double[] ord_bis = new double[n];
Array.Copy(ord_depart, ord_bis, n);
double[] fonction = interpol_steffen2(abs_depart, ord_depart);
double[] result = convoler(fonction);
double erreur = distance(result, abs_depart, ord_depart);
Random r = new Random();
for (int i = 0; i < 5000; i++)
{
int indice = r.Next(n);
double last_val = ord_bis[indice];
modifier(ord_bis, indice, r);
fonction = interpol_steffen2(abs_depart, ord_bis);
result = convoler(fonction);
if (erreur < distance(result, abs_depart, ord_depart))
ord_bis[indice] = last_val;
else
erreur = distance(result, abs_depart, ord_depart);
```

```
}
return interpol_steffen2(abs_depart, ord_bis);
}
private static double[] convoler(double[] fonction)
{
double[] bis = new double[101];
for (int i = 0; i < 101; i++)
{
double s = 0.0;
double s_poids = 0.0;
for (int j = -3; j <= 3; j++)
{
double poids = f_convol(Math.Abs(j));
if ((i + j) >= 0 && (i + j) < 101)
{
s_poids += poids;
s += poids * fonction[i + j];
}
}
bis[i] = s / s_poids;
}
return bis;
}
private static double f_convol(int x)
{
if (x == 0)
return 1.0;
else if (x == 1)
return .7;
else if (x == 2)
return .3;
else if (x == 3)
return .1;
else
return 0.0;
}
```

B.3.4 Bézier

Nous développons ici l'algorithme des courbes de tendance, non pas celui par inversion de matrice. L'interpolation de Bézier nécessite dans un premier temps de calculer tous les temps t_i tels que $x(t_i) = x_i$.

public static double[] interpol_bezier(double[] abs_depart, double[] ord_depart)

```
    {
    int n = abs_depart.Length;
    double[] ord_bis = new double[n];
    Array.Copy(ord_depart, ord_bis, n);
    if (les_t == null)
    les_t = calculer_t(abs_depart);
    double[] fonction = bezier(abs_depart, ord_depart, les_t);
    double erreur = distance(fonction, abs_depart, ord_depart);
    Random r = new Random();
    for (int i = 0; i < 2500; i++)
    {
    int indice = r.Next(n);
    double last_val = ord_bis[indice];
    modifier(ord_bis, indice, r);
    fonction = bezier(abs_depart, ord_bis, les_t);
    if (erreur < distance(fonction, abs_depart, ord_depart))
    {
    ord_bis[indice] = 2 * last_val - ord_bis[indice];
    fonction = bezier(abs_depart, ord_bis, les_t);
    if (erreur < distance(fonction, abs_depart, ord_depart))
    ord_bis[indice] = last_val;
    else
    erreur = distance(fonction, abs_depart, ord_depart);
    }
    else
    erreur = distance(fonction, abs_depart, ord_depart);
    }
    return fonction;
    }
    private static double[] bezier(double[] abs_depart, double[] ord_depart, double[]
les_t)
    {
    double[] result = new double[101];
    int n = abs_depart.Length;
    result[0] = ord_depart[0];
    result[100] = ord_depart[n - 1];
    for (int i = 1; i < 100; i++)
    {
    double s = 0.0;
    for (int k = 0; k < n; k++)
    s += MesMaths.ckn[n - 1][k] * ord_depart[k] * Math.Exp(k * Math.Log(les_t[i])
+ (n - 1 - k) * Math.Log(1 - les_t[i]));
    result[i] = s;
    }
    return result;
    }
```

```
    private static double[] calculer_t(double[] abs_depart) // calcule les ti tels que
abscisse courbe_bezier(t) = 0.01 * i
    {
    double[] les_t_finaux = new double[101];
    les_t_finaux[0] = 0.0;
    les_t_finaux[100] = 1.0;
    double t = 0.0;
    int i = 0;
    while (i < 99)
    {
    double abscis = calcul_abs_bezier(t, abs_depart);
    if (abscis > (((double)(i+1)) * .01))
    {
    i++;
    les_t_finaux[i] = t;
    }
    t += 0.001;
    }
    return les_t_finaux;
    }
    private static double calcul_abs_bezier(double t, double[] abs_depart)
    {
    int n = abs_depart.Length;
    double s = 0.0;
    for (int k = 0; k < n; k++)
    s += MesMaths.ckn[n - 1][k] * abs_depart[k] * Math.Exp(k * Math.Log(t) + (n
- 1 - k) * Math.Log(1 - t));
    return s;
    }
```

B.3.5 Intégrale

```
    public static double[] interpol_integrale(double[] abs_depart, double[] ord_depart)
    {
    int n = abs_depart.Length;
    double[] abs_derivees = new double[n + 1];
    double[] derivees = new double[n + 1];
    abs_derivees[0] = 0.0; abs_derivees[n] = 1.0;
    for (int i = 0; i < n - 1; i++)
    {
    abs_derivees[i + 1] = 0.5 * (abs_depart[i] + abs_depart[i + 1]);
    derivees[i + 1] = (ord_depart[i + 1] - ord_depart[i]) / (abs_depart[i + 1] -
abs_depart[i]);
    }
    derivees[0] = derivees[1];
```

```
    derivees[n] = derivees[n - 1] ;
    double[] fonction = interpo_integr(ord_depart, abs_derivees, derivees) ;
    double erreur = distance(fonction, abs_depart, ord_depart) ;
    Random r = new Random() ;
    for (int i = 0 ; i < 2000 ; i++)
    {
    int indice = r.Next(n+1) ;
    double last_val = derivees[indice] ;
    modifier(derivees, indice, r) ;
    fonction = interpo_integr(ord_depart, abs_derivees, derivees) ;
    if (erreur < distance(fonction, abs_depart, ord_depart))
    {
    derivees[indice] = 2 * last_val - derivees[indice] ;
    fonction = interpo_integr(ord_depart, abs_derivees, derivees) ;
    if (erreur < distance(fonction, abs_depart, ord_depart))
    derivees[indice] = last_val ;
    else
    erreur = distance(fonction, abs_depart, ord_depart) ;
    }
    else
    erreur = distance(fonction, abs_depart, ord_depart) ;
    }
    return fonction ;
    }
    private static double[] interpo_integr(double[] ord_depart, double[] abs_derivees,
double[] derivees)
    {
    double[] fonc = interpol_steffen2(abs_derivees, derivees) ;
    double[] fonction = new double[101] ;
    fonction[0] = ord_depart[0] ;
    for (int i = 1 ; i < 101 ; i++)
    fonction[i] = fonction[i - 1] + .01 * fonc[i] ;
    return fonction ;
    }
```

Table des figures

Bibliographie

[1] SG Credit Research Team. Pricing CDOs with a smile. Février 2005.

[2] SG Credit Research Team. Bespoke CDO : latest Developments. Juillet 2006.

[3] SG Credit Research Team. Pricing and hedging correlation products. Juillet 2004.

[4] James Wood and Ed Parcell (Derivative Fitch). Wiping the smile off your base. Juin 2007.

[5] SG Credit Research Team. Modélisation et évaluation des produits dérivés de crédit. Décembre 2007.

[6] M.Steffen (Astronomy and Astrophysics). A simple method for monotonic interpolation in one dimension. Août 1990.