Description du module GENERATEUR rev.2

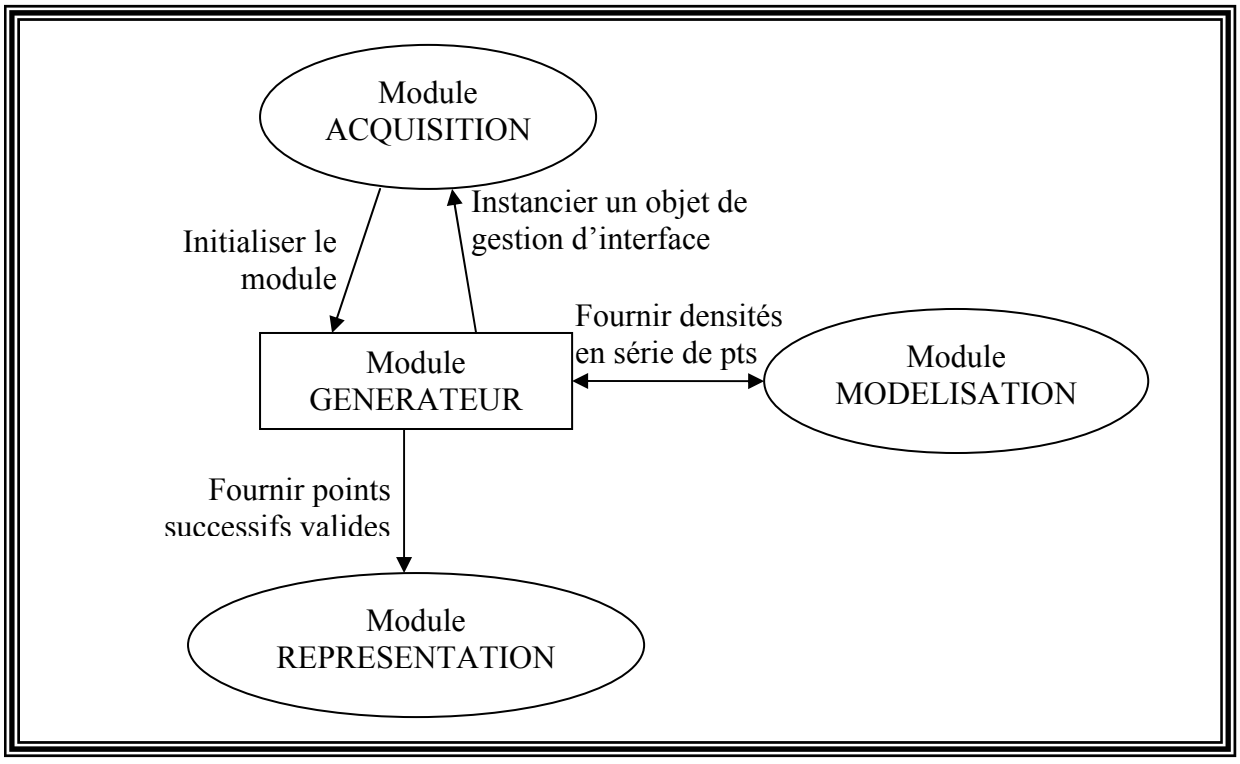
1. Rôle du module

Ce module doit implémenter un générateur de « points aléatoires » selon une répartition de densité donnée. Tout d'abord, le générateur doit être initialisé ; puis à chaque sollicitation, le module est en mesure de fournir un point valide pour le modèle en cours. Les modalités sont précisées plus loin.

Rév. 2 : Les points générés sont forcément valides et n'ont plus à être acceptés par le module MODELISATION.

2. Echanges avec les autres modules

A. Représentation visuelle



B. Paramètres en entrée

Durant une phase d'initialisation, le générateur demandera les densités en une série de points au module MODELISATION. Par ailleurs, les paramètres essentiels au fonctionnement du générateur, imposés par l'utilisateur seront fournis par l'intermédiaire d'un objet de gestion de l'interface, selon le même schéma que dans le module MODELISATION.

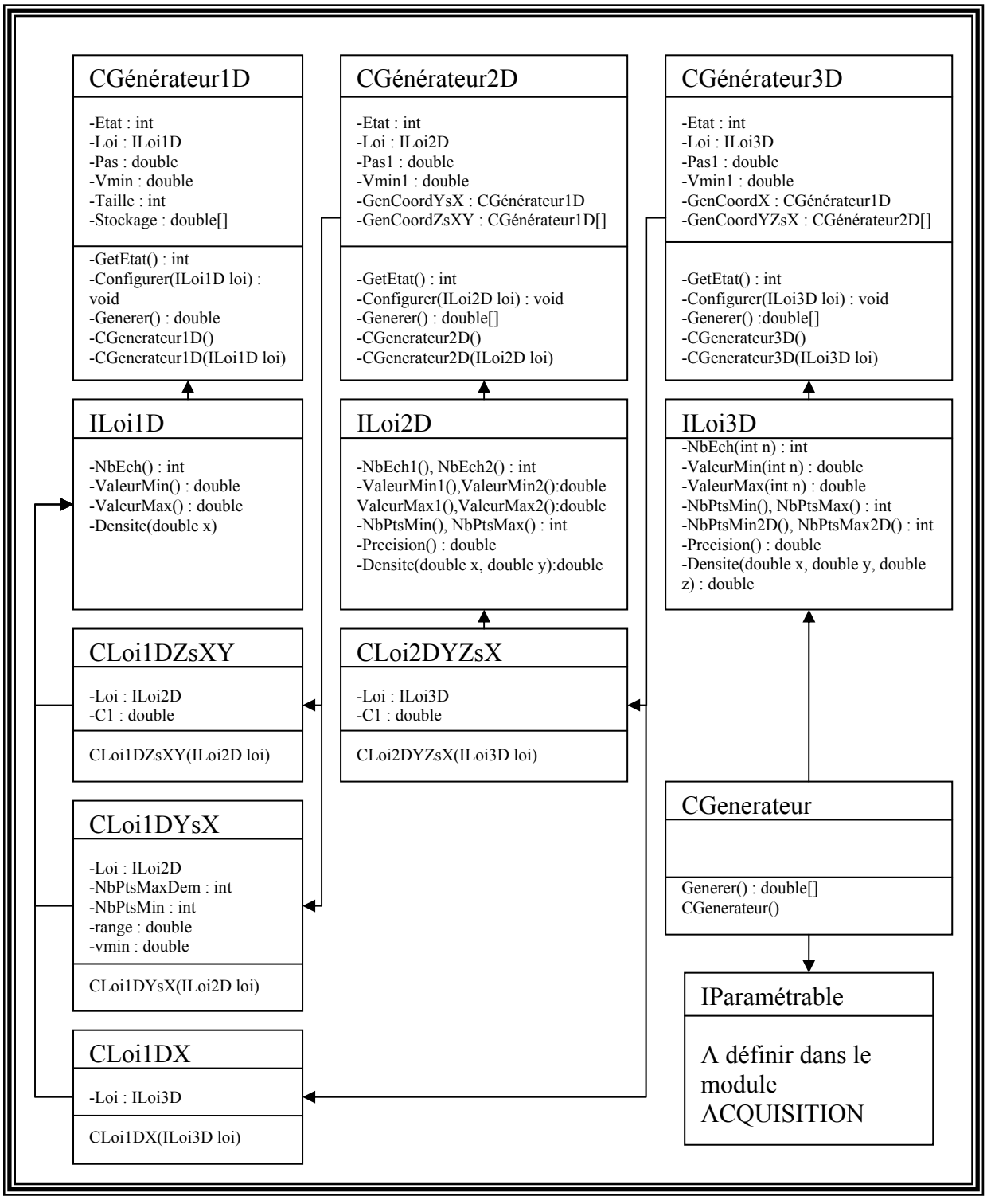
Paragraphe révisé (rev.2): Après l'initialisation, il n'est plus jamais fait appel à la modélisation.

C. Paramètres en sortie

Le module implémente l'interface IParamétrable du module ACQUISITION, pour son initialisation.

A la demande du module REPRESENTATION, le générateur lui calculera un nouveau point, c'est-à-dire un triplet de réels.

3. Diagramme de classes



4. Description des classes

Rev.2 : Le diagramme des classes a été considérablement remanié par rapport à la précédente mouture afin d'éviter la redondance de code générateur, particulièrement délicat à optimiser. Dans cette organisation, tout le réel travail de génération est fait dans CGenerateur1D, les deux autres objets principaux (CGenerateur2D et CGenerateur3D) ne faisant que préparer les données, et initialiser le CGenerateur1D avec la bonne configuration (celle-ci est mise en place en affectant un objet implémentant ILoi1D, à savoir CLoi1DX, CLoi1DYsX et CLoi1DZsXY).
Les états des objets, s'ils sont toujours présents, ont été simplifiés, et leur sens est maintenant commun aux trois types de générateur.(cf.CGenerateur1D).
Les objets sont décrits ici dans les grandes lignes, mais les références sont plus détaillées dans la documentation HTML générée par javadoc à partir des commentaires de source. S'y reporter donc.

A. La classe CGenerateur1D

C'est devenu l'objet pivot du module. Etant donné une loi de densité à 1 dimension fournie par un objet implémentant l'interface ILoi1D, et bien entendu après une initialisation qui a lieu au premier appel, l'objet CGenerateur1D est capable de fournir un nombre aléatoire selon cette loi de densité. (Il utilise une loi de transformation décrite plus loin).

1. L'état du générateur

Le champ Etat stocke un entier qui peut prendre trois valeurs distinctes :

- 0, l'objet vient d'être construit par défaut, et n'a aucune loi associée.
- 1, l'objet dispose d'une loi (ILoi1D) mais n'a encore jamais été sollicité, son initialisation n'a donc pas encore eu lieu.
- 3, l'objet est initialisé, et il n'utilise plus la loi qui a servi à cette initialisation.

L'état peut être consulté par la méthode GetEtat(), pour quelque raison que ce soit, mais cet état sert essentiellement à modifier le comportement interne de la méthode Generer().

2. La méthode Configurer() et l'interface ILoi1D

Que ce soit par le constructeur ou par la méthode configurer, le générateur 1D doit se voir fournir un objet implémentant ILoi1D afin de pouvoir fonctionner.
Cette interface fournit notamment :

- La loi de densité à une dimension (évidemment)
- L'intervalle de travail (ValeurMin et ValeurMax en sont les bornes)
- Le nombre d'échantillons à faire dans cet intervalle (NbEch). Cela définit donc avec les 2 valeurs précédentes le pas d'échantillonnage.

Cette interface est essentielle à l'utilisation du générateur 1D par les deux autres (2D et 3D).

3. La méthode Generer()

a) Comportement suivant l'état de l'objet

- Etat 0, la méthode génère une exception EtatNonValide, parce que ce cas ne doit pas survenir en cours d'utilisation normale.
- Etat 1, la méthode effectue l'initialisation, c'est-à-dire :
  - Allocation du tableau de stockage
  - Remplissage de ce tableau

- Stockage dans l'objet de Vmin, Taille, Pas afin d'éviter d'avoir à les solliciter ou les calculer plus tard dans le chemin critique(j'entends par là la portion de code qui va être bouclée un très grand nombre de fois, et doit par conséquent être particulièrement optimisée.)
- Etat 3, la méthode génère un nombre aléatoire à l'aide de l'algorithme décrit au paragraphe suivant.

b) Description de l'algorithme implémenté

CGenerateur1D est l'objet de base du module : on lui donne une taille, on échantillonne la loi de densité en stockant taille échantillons, puis on calcule à partir de là les sommes partielles de la somme de Darboux. (Cette somme tend vers la primitive de la loi de densité sommée quand le nombre d'échantillons tend vers l'infini). Dès lors, le générateur est prêt à fonctionner selon le principe suivant : on génère un nombre aléatoire n selon une répartition uniforme (entre 0 et la valeur de la somme totale), on parcourt la liste des sommes partielles s(k) dans l'ordre k croissant, et on fournit l'indice i de la première case dont le contenu est supérieur au nombre n. A partir de cet indice i, on calcule la coordonnée correspondante à l'aide du pas et de la coordonnée minimum.

On utilise le rapport (s(k) – n) / (s(k) – s(k-1)) pour effectuer une interpolation linéaire de la coordonnée. Ceci revient à avoir un générateur fournissant des points dans R^3, à partir d'une loi de densité à 3 dimensions échantillonnée avec les pas PasN (N = X, Y, Z), et maintenue constante entre les limites d'échantillonnage.

Cet échantillonnage pourrait certes être évité, en recalculant systématiquement les tableaux pour les valeurs X et Y tirées (et plus haut pour les valeurs X tirées), mais au prix d'un temps de génération par point beaucoup plus élevé.

Cependant, en l'état on pourra déjà tirer parti du fait que les pas sont différents sur les trois axes : on peut prendre un pas plus élevé le long de la ligne de visée (c'est-à-dire pour l'axe Z), tout en réduisant la précision angulaire (i.e. les pas des axes X et Y).

Rev.2 : Le parcours des sommes partielles est maintenant effectué par dichotomie, ce qui permet d'obtenir une recherche en O(log2 n) au lieu de O(n). Du point de vue de notre générateur 3D, cela signifie qu'on peut choisir un échantillonnage serré sur l'axe Z, sans que les temps de calculs augmentent trop.

B. La classe CGenerateur2D

Du point de vue externe, ce générateur fait la même chose que CGenerateur1D, mais pour une loi de probabilité à 2 dimensions. En réalité, cette classe « travaille » très peu, puisqu'elle se contente de dispatcher le travail de calcul à des objets de classe CGenerateur1D, et que la majorité des ajustements, pour passer d'une loi 2D à une loi 1D sont faits par les classes CLoi1DZsXY et CLoi1DYsX.

1. Générer Y connaissant X : la classe CLoi1DYsX

a) Principe

Le X est fixé au moment où on fait appel au générateur 2D, et ce indépendamment de lui. Il s'agit donc de générer la première coordonnée de la paire de nombres aléatoires demandée au générateur 2D. Or, la loi de probabilité de Y (connaissant X) est obtenue en intégrant la loi de probabilité h(x0 ; y, z) sur z sur l'intervalle de travail :

$$h(x0 ; y) = \text{int}(h(x0 ;y, z), z=\text{ValeurMinz}..\text{ValeurMaxz})$$

Pour notre application, il s'agit donc simplement de calculer cette intégrale pour chaque valeur de y qui nous intéresse, c'est-à-dire à chaque pas d'échantillonnage. C'est exactement ce que fait cette classe : outre passer les paramètres définis dans l'interface ILoi1D, la loi de densité 1D qu'elle implémente réalise en fait le calcul de cette intégrale.
Cette classe est donc utilisée par le générateur2D pour initialiser le générateur1D, qui lui servira alors à générer Y.

b) Détails

En l'état, les calculs d'intégrales sont fait par une méthode Monte-Carlo : c'est-à-dire qu'on génère une grande quantité de nombre aléatoires z répartis selon une loi uniforme, et qu'on en fait la somme des densités h(x0, y0 ; z) en ces points, le tout étant divisé par la quantité de nombre générés :

$$\text{Int}(h(x0, y0 ;z), z=\text{ValeurMinz}..\text{ValeurMaxz}) = \lim((1/N)*\text{sum}(h(x0, y0 ;zn), n=0..N-1), N->\text{infinite})$$

Où les zn sont une suite de réalisations indépendantes d'une loi de densité uniforme entre ValeurMinz et ValeurMaxz, et nulle ailleurs.
Pour ces calculs, il est prévu trois paramètres : le nombre minimum de points à utiliser, le nombre de points maximum à utiliser (ce afin de majorer le temps de calcul), et l'erreur absolue tolérée sur le résultat de l'intégrale. Cependant ces paramètres ne sont pas encore exploités correctement : il faut évaluer la précision du calcul, déterminer le nombre de points nécessaires pour atteindre une précision donnée…

2. Générer Z connaissant X et Y : la classe CLoi1DZsXY

Cette classe est des plus simples, elle ne fait que passer au générateur 1D une loi de densité à 1D obtenue directement à partir de la loi 2D, où la coordonnée y est une constante définie à la création de la classe. En fait, le générateur2D va créer autant de classes CLoi1DZsXY et de CGenerateur1D qu'il y a d'échantillons sur l'axe Y. Lors de la génération d'un point, le bon générateur est appelé après avoir généré un Y à l'aide du générateur1D configuré par la classe CLoi1DYsX(cf. § précédent).

3. La méthode Configurer et l'interface ILoi2D

Comme le CGenerateur1D, le CGenerateur2D doit se voir fournir une configuration, sous la forme d'un objet implémentant l'interface ILoi2D. Dans cette interface sont définies les méthodes suivantes :

- ValeurMax1, ValeurMin1, ValeurMax2, ValeurMin2 : donnent l'intervalle de travail en Y (1) et en Z (2)
- NbEch1, NbEch2 : le nombre d'échantillons demandé en Y, et en Z
- Precision, l'erreur tolérable pour le calcul d'intégrales
- NbPtsMin, NbPtsMax : les nombres de points min et max à utiliser pour le calcul des intégrales.
- Densite, la loi de densité à 2D à utiliser pour générer les paires aléatoires.

Cf. §4B1b

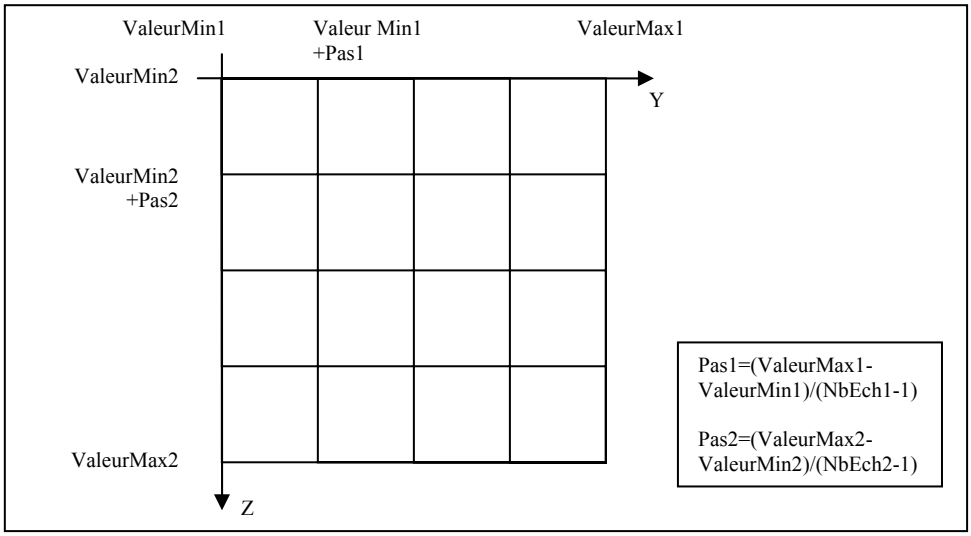


Figure 1 : Signification de ValeurMinN, ValeurMaxN, NbEchN

4. La méthode Generer

Comme le CGenerateur1D, son comportement dépend de l'état de l'objet :

- Etat 0, on génère une exception EtatNonValide
- Etat 1, on crée NbEch1+1 CGenerateur1D, dont NbEch1 sont configurés avec une classe CLoi1DZsXY (leurs références sont stockées dans le tableau GenCoordZsXY), et le dernier est configuré avec un objet CLoi1DYsX(dont la référence est stocké dans le champ GenCoordYsX)
- Etat 3, on génère un Y à l'aide de GenCoordYsX, puis en fonction du résultat, on appelle un des GenCoordZsXY[i] pour générer un X, où Y appartient à l'intervalle Yi ;Y{i+1}. (on pose Y0 = ValeurMin1 et Y{NbEch}=ValeurMax1, tous les intervalles étant de largeur égale au pas d'échantillonnage)

C. La classe CGenerateur3D

Fonctionnellement, cette classe utilise le même artifice pour passer d'une loi 2D à une loi 3D, que le générateur 2D pour passer d'une loi1D à une loi 2D.

1. La classe CLoi1DX

Le générateur utilise un générateur 1D, configuré avec un objet CLoi1DX, pour générer un X (la loi de densité h(x) pour tout y et tout z est également calculée par intégration, mais sur une surface cette fois-ci : le rectangle {X}x[ValeurMin2..ValeurMax2]x[ValeurMin3..ValeurMax3].) Ce calcul d'intégrale est effectué dans l'objet CLoi1DX, pour des valeurs X fournies par le générateur (calculées en fonction de l'échantillonnage choisi, i.e de ValeurMin1, ValeurMax1 et NbEch1).

2. La classe CLoi2DYZsX

Le générateur utilise également NbEch1 générateurs 2D, pour générer Y et Z, connaissant X (déterminé par le générateur 1D) ;en effet le CGenerateur2D approprié est appelé en fonction du X tiré. Pour ce faire, le CGenerateur2D est configuré avec un objet CLoi2DYZsX, qui implémente l'interface ILoi2D.

3. L'interface ILoi3D

Comme les autres générateurs, le générateur 3D doit être configuré par un objet implémentant une interface spécifique : ILoi3D. Les méthodes sont les suivantes :

- NbEch fournit le nombre d'échantillons sur chaque axe
- ValeurMin, ValeurMax, les bornes de l'intervalle de travail sur chaque axe : cela définit le cube d'espace dans lequel le générateur travaillera. Plus précisément, le générateur tirera des points dans toutes les zones de ce cube où la densité est non nulle.

Remarque 1 : Pour une densité très proche de zéro, la possibilité de tirer un point à cet endroit existe, mais si la densité est exactement nulle, il est assuré que le générateur ne tirera **jamais** de point dans cette zone.
Remarque 2 : Le générateur ne doit en aucun cas se voir fournir une loi de densité pouvant prendre des valeurs négatives, ce qui de toute façon n'a aucun sens. Le résultat d'une telle erreur est indéterminé.

- Precision, NbPtsMin, NbPtsMax, même sens que pour ILoi2D.
- NbPtsMin2D, NbPtsMax2D, l'équivalent pour les intégrales à 2D exigées par le CLoi1DX.

5. Discussions diverses

A. Loi de densité échantillonnée

Il est impératif d'échantillonner, toute autre solution serait trop coûteuse en temps de calcul. Cependant, nous avions deux possibilités.

- **Echelons** (solution retenue) : Un cube (plus exactement un pavé) est choisi à l'aide de la loi de densité discrétisée, et dans le « cube » sélectionné un point est choisi avec une loi uniforme. (Du moins c'est à ça que revient l'algorithme si on interpole)

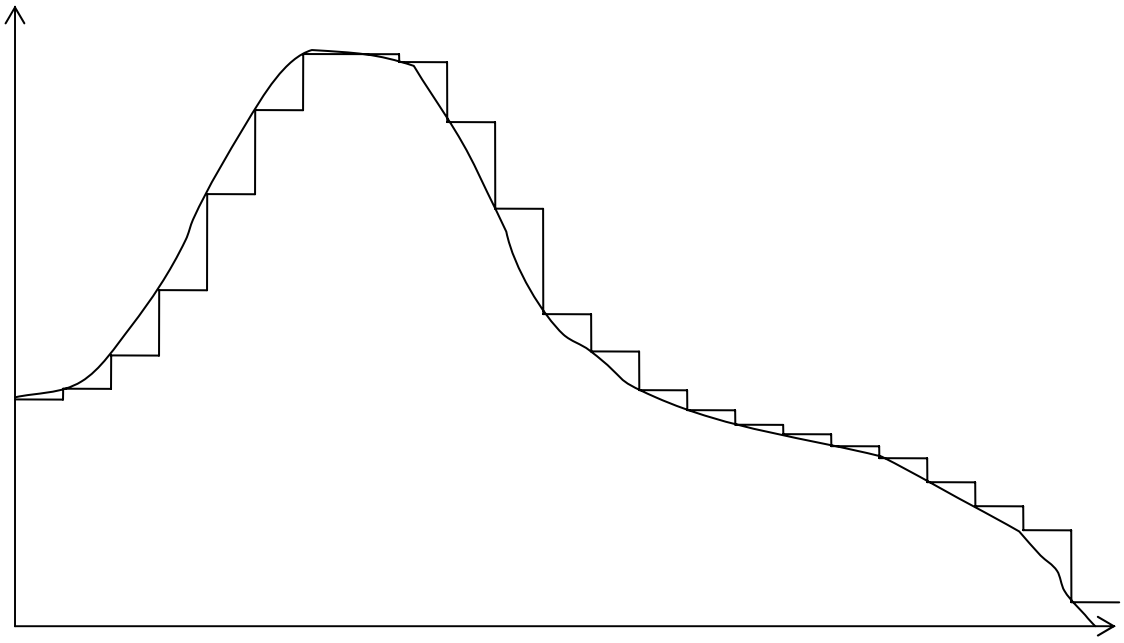


Figure 2 : La coordonnée est générée selon une loi de probabilité de la forme ci-dessus (montrée en comparaison avec la loi originale)

- **Discrétisation** (autre solution possible) : Une grille 3D pouvait être définie, et les seuls points générables auraient été les intersections des droites de la grille. Cela serait revenu à échantillonner la loi de densité sous forme de distribution de Dirac (c'est ce qui a lieu si on n'interpole pas.)

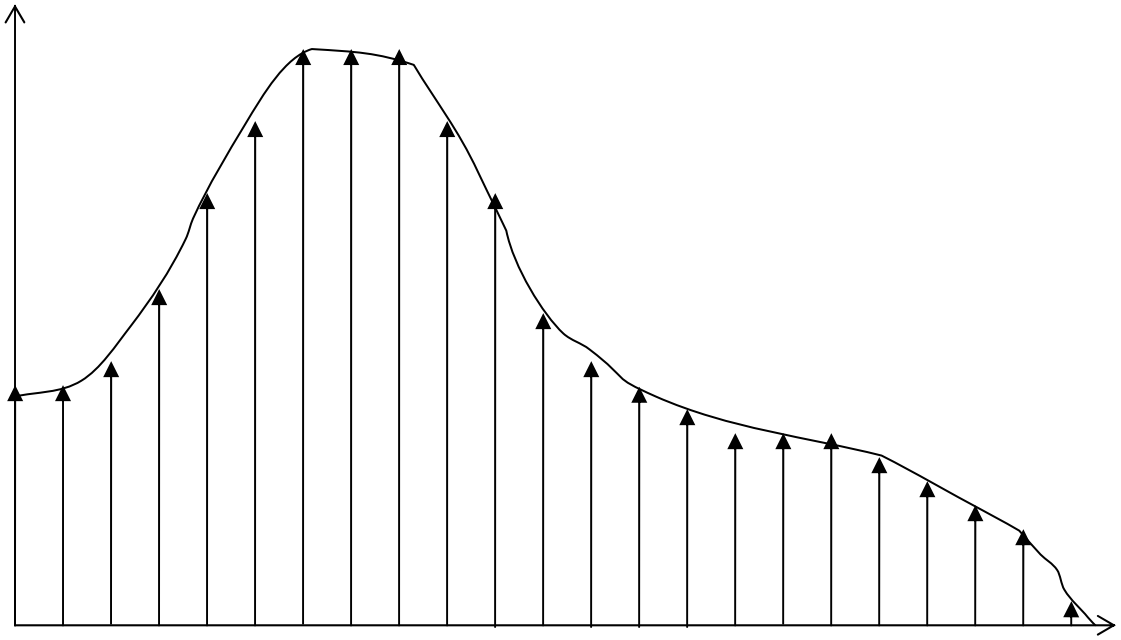


Figure 3 : En comparaison avec la loi originale, la distribution équivalente au générateur dans la deuxième solution.

B. Initialisation progressive / Initialisation partagée

L'initialisation, si elle n'est pas faite de façon judicieuse, peut être longue. Deux méthodes se sont présentées jusqu'à présent.
L'initialisation progressive consiste à ne remplir les différents sous-tableaux qu'à leur première utilisation : ceci ne complique en rien la programmation, il n'y a que quelques ajouts dans la classe CGénérateur1D. Au bout d'un certain, la majorité des tableaux auront été initialisés, mais progressivement, sans de grosses périodes de calcul dépourvues d'effet visuels concrets. L'intérêt est que l'on pourra tester rapidement des valeurs de simulation et redémarrer rapidement la simulation sans perte de temps excessive.

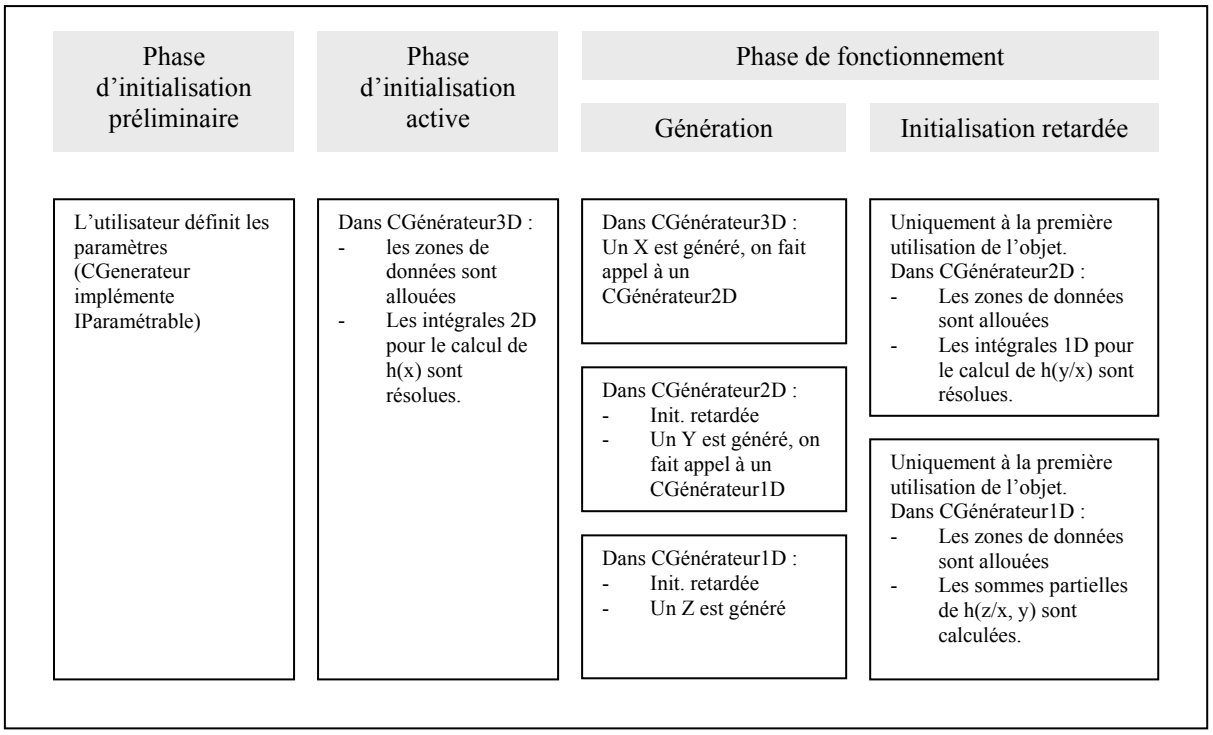


Figure 4 : Description de l'initialisation progressive du module

L'initialisation partagée consistait à calculer les différentes intégrales, dont les domaines d'intégration se recouvraient, avec les mêmes points. Mais après discussion avec M.Leroy, il semble que ceci pourrait avoir des effets négatifs sur la qualité des résultats, car on risquerait de créer des corrélations entre les différents résultats (étant donné que le générateur de nombre uniforme n'est pas parfait – générateur à congruence linéaire-). De plus cela nuisait à la précision des calculs. Cette dernière solution a donc été abandonnée, d'autant qu'elle n'aurait permis de réduire que de moitié la phase préliminaire des initialisations.

6. Les tests

A. Les générateurs à loi de probabilité arbitraire

Les différents générateurs ont été testés au cours de leur développement, à l'aide de classes spécialement conçue pour cela. A chaque générateur a été associé une applet graphique permettant de vérifier son fonctionnement sur le modèle suivant :

- L'applet TestGenerateurND initialise le CGenerateurND avec une classe CTestLoiND, implémentant l'interface ILoiND.
- Le générateur ainsi initialisé est utilisé pour tirer un grand nombre de points.

- Chaque point est comptabilisé dans un histogramme à N dimensions.
- Cet histogramme est ensuite présenté sur l'espace de travail de l'applet sous la forme d'une ou plusieurs courbes.

Les courbes qui se forment à l'écran doivent être homothétiques à la courbe de la densité de probabilité utilisée pour initialiser le générateur (en effet l'aire sous la courbe n'est pas rapportée à l'unité, comme c'est le cas pour une densité de probabilité pour simplifier l'affichage). Ceci à condition que le pas d'échantillonnage soit suffisamment faible, car dans le cas contraire, l'échelonnement apparaît : la courbe affichée est en escalier, et le nombre de palier correspond au nombre d'échantillons demandés. Ceci fait apparaître clairement la différence qui existe entre la densité de probabilité des nombres tirés par le générateur, et la densité espérée. Cette différence est d'autant plus faible que le nombre d'échantillons augmente. (Cf.figure 2).

B. Le générateur uniforme de Java

Une applet TestRandom tire un grand nombre de paires de nombres aléatoires à l'aide de la méthode Random de la classe Math (méthode utilisée à la base des trois générateurs précédents), et sont affichés sur la surface de l'applet à l'écran. L'apparition de motifs dénoterait un mauvais générateur, mais le rectangle est très rapidement noirci : le générateur passe donc ce test avec brio.
Aucun autre test n'a encore été effectué ; nous nous réfèrerons à la littérature ci-dessous.
La documentation de l'API Java indique que le générateur de nombres pseudo-aléatoire implanté dans l'API java est un générateur à congruence linéaire utilisant une « graine » (le nombre qui initie la congruence) sur 48 bits, décrit dans Donald Knuth, *The Art of Computer Programming, Volume 2*, Section 3.2.1.

7. Les performances actuelles

A compléter.