Pricing d'option européennes et américaines à l'aide d'un arbre trinomial en VBA et Python

Charbit Paul-Antoine / Castellini Fabien

Université Paris-Dauphine – Master 272 IEF

Présentation

Ce document sert de support à notre projet visant à développer un modèle de pricing pour les options vanilles européennes et américaines, avec ou sans dividende discret. Nous utilisons un arbre trinomial dans une approche de programmation orientée objet en Python et VBA. Le modèle trinomial repose sur la modélisation de la dynamique du sous-jacent en trois états possibles à chaque intervalle de temps: hausse, baisse ou évolution constante, avec des probabilités calculées dans un cadre risque neutre. Nous explorerons les résultats obtenus, la comparaison entre les implémentations VBA et Python, la convergence vers le modèle Black & Scholes pour les options européennes, ainsi que le calcul des grecques principaux: Delta, Gamma et Vega.

1 Modèle Trinomial

Le modèle trinomial, une approche d'évaluation d'actifs contingents, est particulièrement adapté aux options américaines sur des sous-jacents unidimensionnels. Il se distingue par sa modélisation de la dynamique du sous-jacent via une loi trinomiale, évitant ainsi l'utilisation d'équations différentielles stochastiques (EDS) typiques des méthodes de Monte Carlo ou des différences finies. Dans ce modèle, chaque étape peut se traduire par l'un des trois états suivants : hausse, baisse ou stabilité, avec des probabilités associées calculées dans un cadre risque-neutre.

2 Structure globale du code

2.1 Construction et Pricing

Pour réaliser le projet, nous avons d'abord choisi de nous concentrer sur le code Python avant de le transposer en VBA. Par conséquent, les structures de chaque code sont relativement proches l'une de l'autre. Le pricer est structuré en trois parties principales:

- Les classes *Market*, *Model* et *Option* pour initialiser les paramètres.
- La classe *Tree*.
- La classe *Node*, utilisée avec la classe Tree pour construire l'arbre.

La construction de l'arbre s'effectue via deux méthodes principales : *Build_Tree* et *Build_Blocks*. Le processus commence par l'instanciation de l'arbre avec des paramètres spécifiques, suivi par l'appel à *Build_Tree* pour construire l'arbre de manière séquentielle en utilisant *Build_Blocks*.

Pour pricer, nous avons recours à une méthode récursive: la méthode *Price* de la classe *Node* vient valoriser chaque noeud, en s'appelant elle même sur les noeuds "enfants". In fine une méthode *Option_Price* de *Tree* appelle Price sur la racine.

2.2 Introduction du dividende discret

L'introduction du dividende discret affecte la valeur du spot des nœuds. La fonction *Is_Close* assure que les connexions dans l'arbre restent adéquates post-dividende, une étape cruciale pour le pricing précis.

2.3 Optimisation

Les probabilités sont initialisées une seule fois dans le constructeur de *Tree* pour optimiser la performance. Le *pruning* est également utilisé pour exclure les branches ayant des probabilités extrêmement faibles, ce qui améliore la performance sans compromettre significativement la précision du modèle.

2.4 Interface VBA

L'utilisation de l'interface VBA par l'utilisateur se présente comme ceci:
- Dans le tableau "Parameters", l'utilisateur renseigne la valeurs des paramètres de l'option qu'il souhaite calculer (ce qui correspond aux cases en fond bleu). L'utilisateur n'est pas obligé de renseigner une date ou un montant de dividende.
- Avant de lancer le pricing, l'utilisateur peut choisir d'intégrer le pruning (avec un seuil de 10**(-8) ainsi que la représentation de l'arbre dans la feuille "Tree"
- Une fois le pricing effectué, le valeur de l'ooption, le temps d'exécution ainsi que le prix et les dérivées du modèle de Black&Scholes sont affichées
- Enfin, dans le tableau "Greeks (with dividend)", l'utilisateur peut calculer les dérivées delta, gamma et vega en fonction des paramètres qu'il a renseigné plus haut ainsi que de l'écart du spot et de la volatilité.

3 Difficultés rencontrées et Solutions

3.1 Python

L'intégration du dividende a présenté des défis, notamment dans la méthode de pricing, où le forward de chaque nœud devait être ajusté pour le dividende.

3.2 VBA

La majorité des difficultés en VBA étaient d'ordre syntaxique, en raison de la transition du code de Python à VBA. Les erreurs communes incluaient la gestion des variables et la construction d'une fonction de pricing efficace.

Premièrement, le message d'erreur "Variable objet ou variable bloc With non définie" du soit à l'oubli du terme "Set" lorsque l'on attribue une valeur à une variable objet ou alors "New" lors de la déclaration d'une nouvelle variable objet.

Nous avons également eu du mal à construire une fonction de pricing efficace (au-delà de 15 pas, impossible de pricer une option dû à la surchauffe d'Excel). Le problème a été simplement réglé en ajoutant la ligne de code "Exit function" dans la condition initiale de la fonction vérifiant que la variable "price_val" n'est pas vide.

Une autre difficulté que nous avons rencontré dans cette fonction "price" est la nécessité de déclarer "price_val" comme Variant pour nous assurer que la condition initiale "IsEmpty" soit respectée (nous n'avons pas trouvé de solution qui nous aurait évité de déclarer un Variant).

Une différence importante par rapport au code Python est la fonction "init_t" permettant d'établir les paramètres initiaux de l'arbre qui sera ensuite appelée dans "MakeTree". Pour cette partie du code, nous nous sommes inspirés du premier cours effectué en classe sur les nombre complexes.

4 Résultats et illustrations

Nombre de pas	Temps d'exécution (s)	Prix
100	0.031942	7.985710
200	0.068132	7.973885
300	0.103148	7.969828
400	0.172276	7.975388
500	0.250822	7.977247
600	1.066291	7.977606
700	0.392260	7.977290
800	0.501976	7.976292
900	1.161464	7.975547
1000	0.661651	7.974731
1100	1.391768	7.973892
1200	0.850518	7.974255
1300	1.710650	7.975027
1400	1.132961	7.975539
1500	1.998575	7.975857
1600	2.006715	7.975848
1700	1.455640	7.975921
1800	2.422417	7.975906
1900	2.450752	7.975824
2000	2.564389	7.975691

Table 1: Temps d'exécution et prix en fonction du nombre de pas

Le temps d'exécution entre les deux codes varie énormément. Pour 100 pas de temps, le code Python s'exécute en 0.03 secondes, quand celui en VBA s'exécute entre 0.7 et 1 seconde.

Les prix obtenus dans le Tableau 1 donne une idée de la vitesse de pricing du code Python, bien qu'obtenus grâce à une boucle rendant les résultats instables et pouvant expliquer pourquoi le temps d'exécution n'est pas parfaitement croissant avec le nombre de pas.

On constate également une différence importante dans la capacité de chaque langage à pricer jusqu'à un certain nombre de pas: seulement 440 pour VBA et jusqu'à 2300 pour Python. Pour rappel notre Pricer utilise une méthode récusrive, limité par définition par la saturation de la mémoire stack.

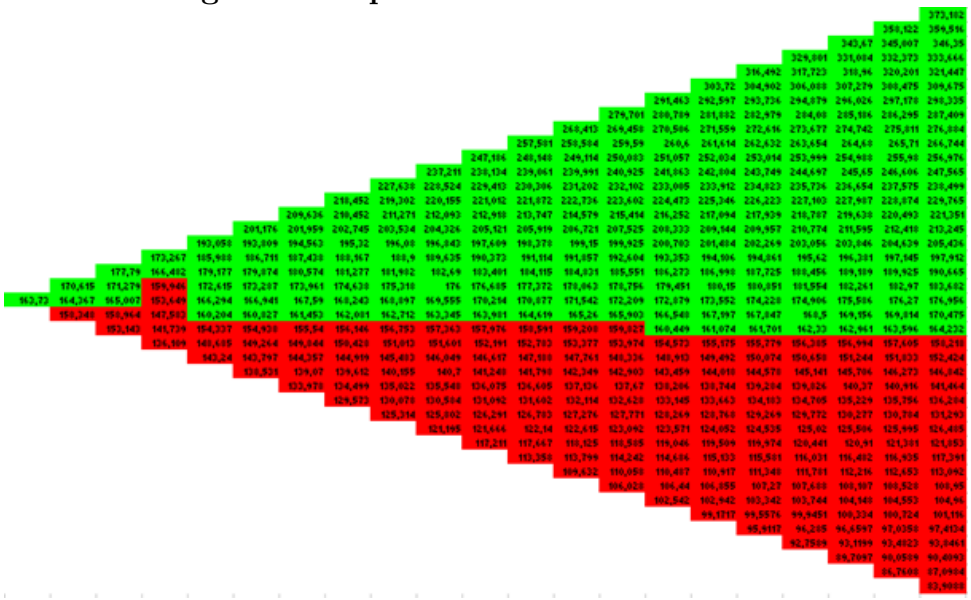
Sur python, cette limite peut-être repoussé via le package sys et sa méthode "setrecursionlimit", mais ce n'est pas une solution pour pricer sur des centaines de milliers de steps.

Figure 1: Différence des prix entre Python et VBA

StartDate	Spot	Rate	Volatility	Dividend	DivDate	Maturity	Type	Exercise	Strike	NbSteps	VBAPrice	PythonPrice	Gap
23/08/2022	88,71	1,46	30,95	1,36	08/12/2022	07/04/2023	Put	European	70,4	46	1,882847	1,882848	7,530E-07
06/08/2022	117,04	3,8	44,15	10,04	29/04/2023	08/05/2023	Put	American	122,97	81	25,48267	25,482674	5,582E-06
08/09/2022	86,5	11,64	49,14	5,25	29/05/2023	13/12/2023	Put	American	69,13	70	7,878481	7,878467	1,431E-05
30/11/2022	101,49	9,3	3,91	0,61	15/10/2023	29/10/2023	Call	American	73,13	94	34,06587	34,065895	2,270E-05
12/08/2022	114,21	-4,6	45,57	6,17	11/08/2023	01/02/2024	Call	European	121,4	81	17,18969	17,189689	8,847E-07
12/01/2022	100,6	-2,07	43,71	2,21	01/02/2022	04/09/2022	Put	American	134,34	90	41,60697	41,606851	1,212E-04
18/02/2022	163,73	9,71	10,77	1,66	02/04/2022	07/12/2022	Call	European	108,88	75	61,34816	61,348146	1,38E-05

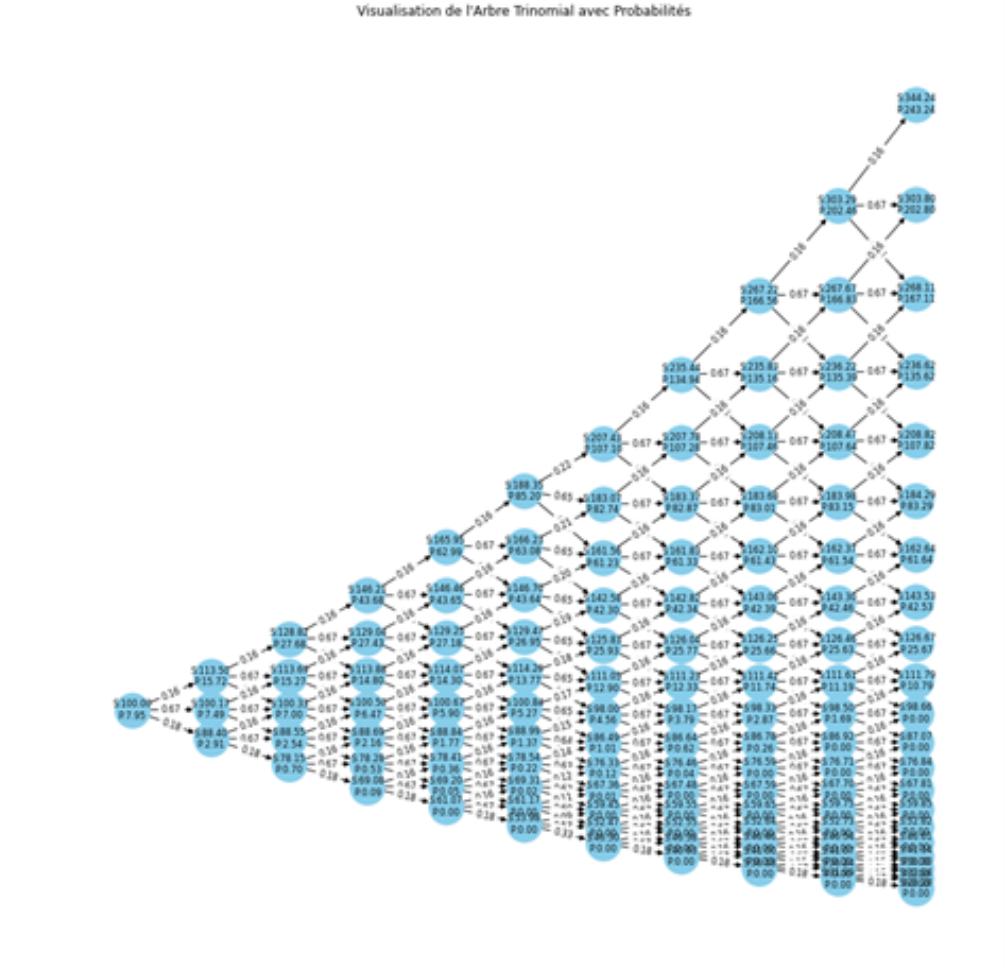
Remarque: les prix sont calculés sans pruning. Les écarts de prix entre les deux codes sont de l'ordre de 10^{-8} à 10^{-4} en fonction des paramètres renseignés.

Figure 2: Représentation de l'arbre sur Excel



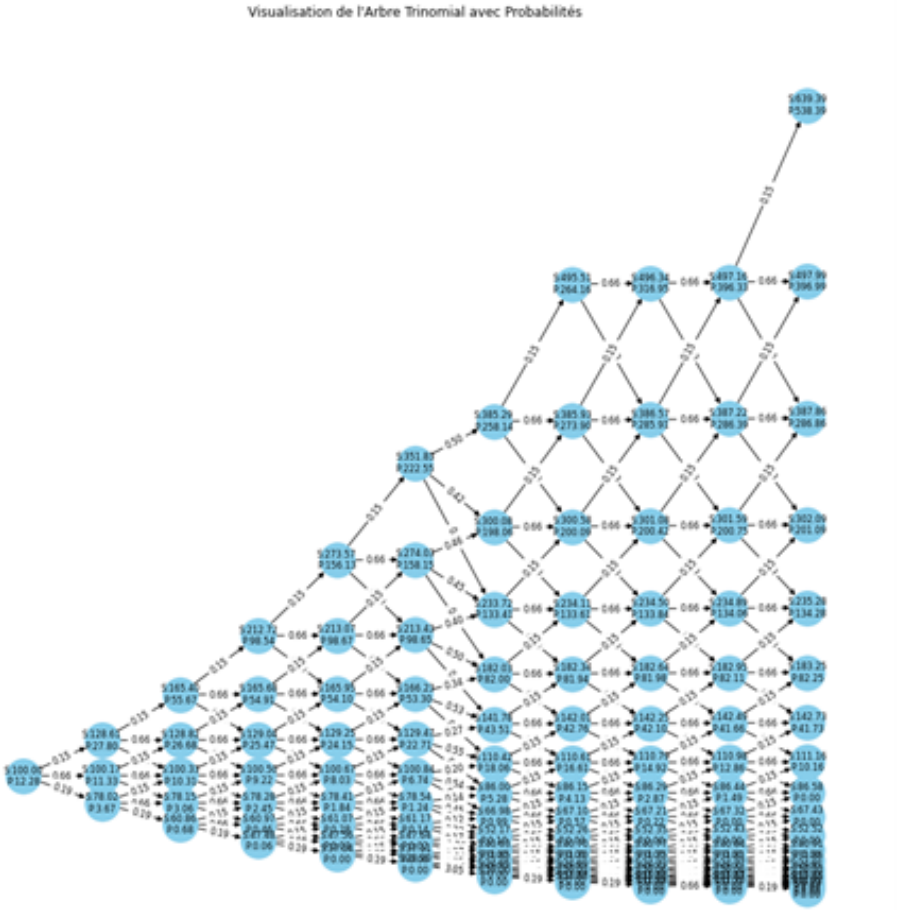
Remarque: illustration d'un call européen à 20 pas de temps, avec dividende et sans pruning. L'impact du dividende est visible sur la frontière d'exercice au quatrième pas de temps.

Figure 3: Représentation de l'arbre sur Python (pruning 10e-10)



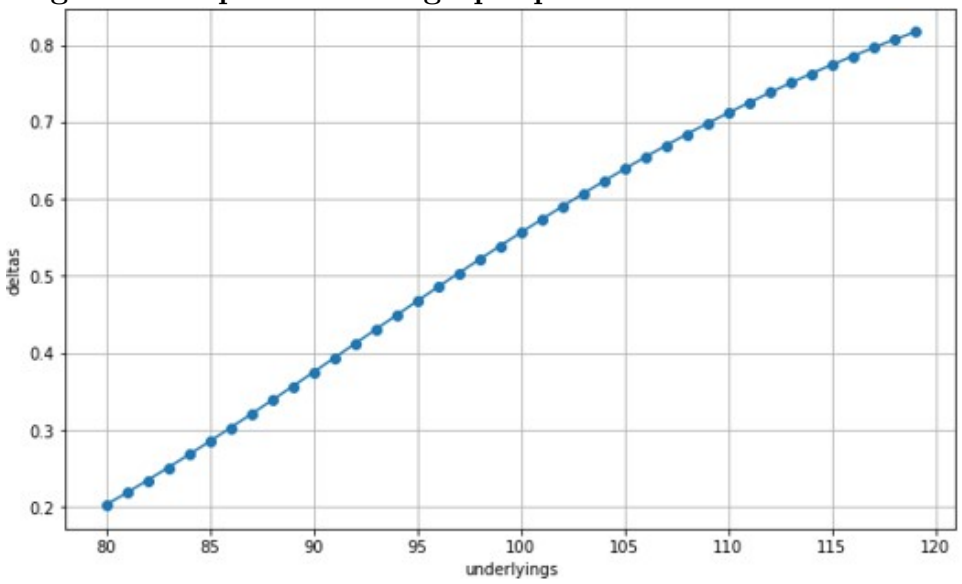
Remarque : Volatilité = 0.25, Dividende = 3 et seuil de pruning à 10e-9.
Ici, les effets du pruning ne sont pas visibles dû à un nombre de pas de temps trop faible. En revanche, on observe bien un étirement de l'arbre vers le haut, et un écrasement vers le bas, ceci étant dû à la vol: les noeuds étant multipliés/divisés d'un facteur $alpha^i$ à partir du centre de chaque colonne. Une volatilité plus élevée aura pour effet d'exacerber ce phénomène.

Figure 4: Représentation de l'arbre sur Python (pruning 10e-7)



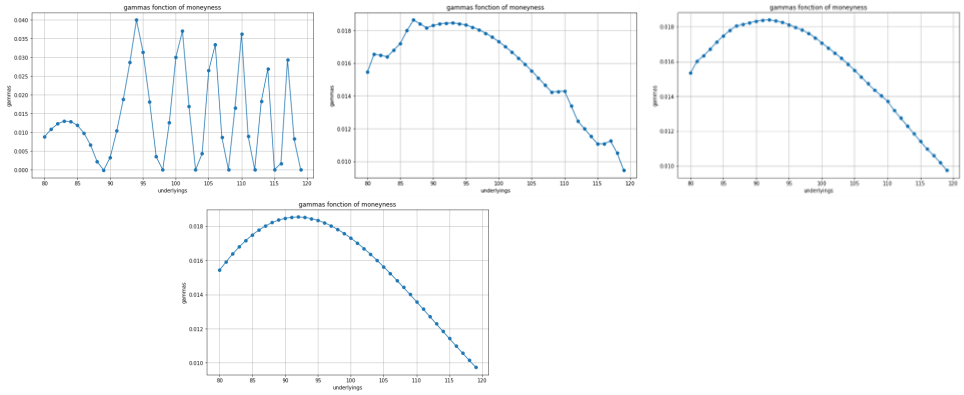
Remarque : Volatilité = 0.5, dividende = 15 et seuil de pruning à 10e-7, ce qui rend visible l'élagation de l'arbre. De plus nous avons volontairement augmenter le dividende par rapport à l'exemple précédent pour illustrer son effet sur l'arbre: pour la colonne correspondant à la date de dividende, tous les noeuds baissent.

Figure 5: Représentation graphique du delta en fonction du spot



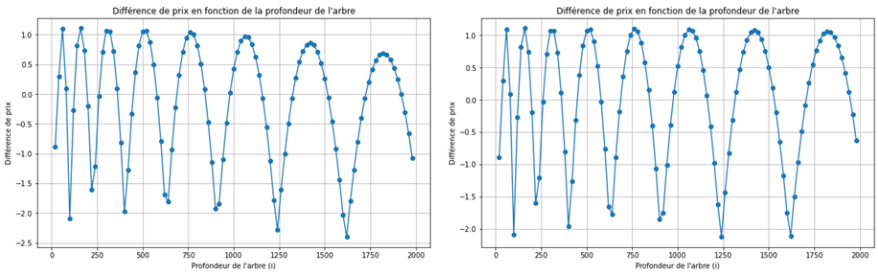
Le delta n'est pas vraiment sensible à la discrétion du spot (première dérivée par rapport au spot (ds)) et l'on obtient un résultat très proche de celui du modèle de Black & Scholes quelque soit la taille du ds.

Figures 6, 7 & 8: Représentations graphiques du gamma en fonction du spot



Remarque: Le gamma converge vers le résultat de BS (pour les cas comparables) pour un dS autour de 3.5. Les graphes représentent les gammas pour un dS de respectivement 0.5, 2, 3.5 et enfin le graphe du bas réprsente le résulat de BS.

Figures 9 & 10: Convergence vers Black & Scholes en fonction du nombre de pas

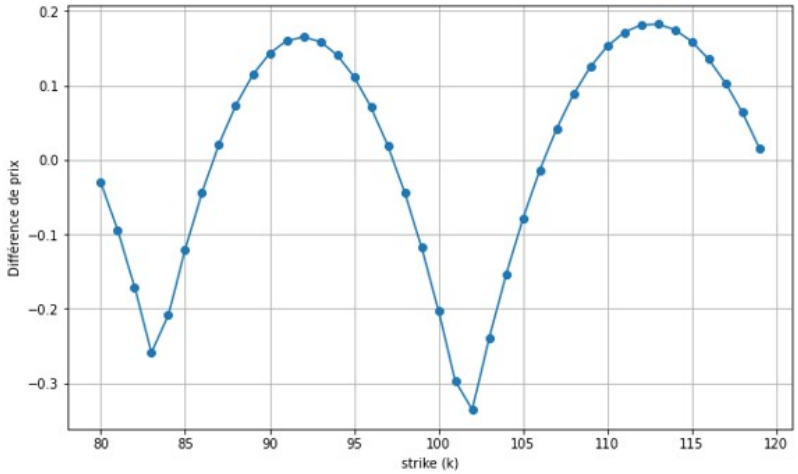


Remarques : la seule différence entre la Figure 9 et la Figure 10 est le seuil de pruning : 10^{-7} vs 10^{-10}. Nous comprenons donc que le pruning a un impact significatif sur le taux de convergence vers le modèle de Black & Scholes. L'axe des ordonnées est

$$(\text{Trinomial\_price} - \text{B\&S\_price}) \times \text{Nb\_steps}.$$

Un corridor entre les bornes supérieures et inférieures implique un taux de convergence égal au nombre de pas de temps. Autrement, le taux de convergence est inférieur au nombre de pas de temps. Le pruning peut être considéré comme un arbitrage entre le nombre de pas de temps nécessaire pour obtenir une convergence acceptable (ou un prix suffisamment précis) et la vitesse d'exécution du code pour le pricing de l'option.

Figure 11: Convergence vers Black & Scholes en fonction du Strike



Remarque : Nombre de pas de temps égal à 10
On observe une certaine periodicité, avec des écarts minimaux lorsque le strike est d'environ: 87,97,107,117.

Améliorations possibles

Dans une optique d'un besoin récurrent d'un pricer, le code pourrait être optimisé grandement en attribuant une méthode de pricing à la classe de paramètres option et non à la classe Tree, ainsi il serait simple de valoriser plusieurs options partageant le même sous jaçent, sans avoir besoin de reconstruire un arbre à chaque fois.

Il pourrait également être intéressant de rajouter un destructeur, afin de libérer la mémoire vive occupé par l'arbre une fois que l'utilisateur n'en a plus besoin. Ce point est relativement mineur en Python où des IDE tels que spyder ou Pycharm permettent de supprimer les variables directement, mais il serait d'un grand interêt sur VBA.

Enfin, en ce qui concerne le code VBA, nous pouvons évoquer l'absence de prise en compte du pruning dans le graphe représentant l'arbre.