NOM : Date : .
PRENOM : Groupe : .

**Finance mathématique : Feuille de réponses du TP 1**
**Les trajectoires d'un modèle CRR à $n$ étapes**

**On répondra aux questions posées aussi clairement que possible dans les espaces prévus et on remettra cette feuille de réponses en fin de TP à l'enseignant chargé du TP.**

L'objet de cet exercice est de calculer et dessiner un modèle à $n$ étapes généralisant les modèles à une et deux étapes vus en cours. Mais avant cela, nous expérimenterons dans le cas d'une marche aléatoire plus simple, la *marche de Wiener*, des lignes de code `Scilab` afin de nous familiariser avec leur syntaxe.

**Exercice 1. :** On rappelle que la marche de Wiener $W_t$ est définie par $W_0 = 0$ et $W_{t+\delta t} = W_t \mp \sqrt{\delta t}$. On désigne par $i$ le nombre de pas de temps, $t = i\delta t$ et par $j$ le nombre de "+", ou de *up*, on peut ranger les valeurs prises par $W_t$ dans une matrice triangulaire[1] $W(i,j) = WW(i+1, j+1)$, pour $i = 0, \ldots, n$ et $j = 0, \ldots, i$.

Charger Scilab puis ouvrir un éditeur (en cliquant sur `Editeur` dans le menu en haut de fenêtre) et saisir le programme suivant puis l'exécuter :

```
n=100 ;T=1 ;delta_t=T/n ;WW0=0 ;
WW=zeros(n+1,n+1) ;WW(1,1)=WW0 ; //initialisation
For i=1 :n WW(i+1,1)=WW(i,1)-sqrt(delta_t) ; //ici j=0
      For j=1 :i WW(i+1,j+1)=WW(i,j)+sqrt(delta_t) ;
      end ;
end ;
```

Calculer les valeurs suivantes de $W$ en fonction de $\delta t$ :

$$\boxed{WW(5,5) = .....\sqrt{\delta t}, \quad WW(5,1) = .....\sqrt{\delta t}, \quad WW(5,2) = .....\sqrt{\delta t}.}$$

En déduire la formule donnant $W(i,j)$ en fonction de $i$, de $j$ et de $\sqrt{\delta t}$.

$$\boxed{W(i,j) = WW(i+1, j+1) = .....\sqrt{\delta t}.}$$

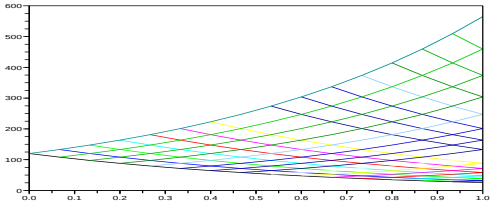
**Exercice 2. :** Considérons à présent le modèle de Cox-Ross-Rubinstein, c'est-à-dire la marche aléatoire définie par $S_t$ définie par $S_0 = S0$ et $S_{t+\delta t} = S_t U_t$, avec $U_t \in \{$`up,down`$\}$. Pour des raisons de réalisme du modèle qui apparaîtront peu à peu, il est important que les facteurs `up`$< 1 <$`down` dépendent de $n$ d'une façon convenablement choisie : on pose

`up=exp(sigma/sqrt(n))`$= e^{+\frac{\sigma}{\sqrt{n}}}$ et `down=1/up`$= e^{-\frac{\sigma}{\sqrt{n}}}$. En introduisant, comme précédemment la notation $S(i,j) = SS(i+1, j+1)$, calculer les valeurs prises par la marche $S_t$ en prenant $\sigma = 0.4$ et $S_0 = 140$. Indiquer quelques unes des valeurs de $S(i,j)$.

[1] La numérotation des lignes et des colonnes par Scilab commence à 1 et non 0 !

**Exercice 3.** **:** On veut à présent tracer l'"arbre de Cox" joignant les points $(i * \delta t, S(i,j))$ à $((i+1) * \delta t, S(i+1, j+\delta J_{i+1}))$, avec $\delta J_{i+1} \in \{0,1\}$ :

Arbre de Cox-Ross-Rubinstein :




On utilisera pour cela les deux instructions suivantes :

`xset("window",1)` qui choisit (et éventuellement crée) la fenêtre 1 comme fenêtre graphique courante (qu'on peut effacer au moyen de `clf(1)`).

`plot2d(Abs,Ord)` qui trace les lignes polygonales (ouvertes) dont les abscisse et les ordonnées des sommets figurent dans les colonnes homologues des matrices notées ici `Abs` et `Ord`.

Le code suivant utilise un algorithme qui parcourt tous les cotés de l'arbre de Cox une fois et une seule et qui se prète donc bien à la syntaxe de `plot2d`.

```
   Abs=zeros(n+1,n+1); // le dessin comportera n+1 lignes polygonales
Ord=zeros(n+1,n+1); // ayant chacune n cotes, donc n+1 sommets
for k=0 :n // k numerote les lignes polygonales
    for l=0 :n-k // premieres moitie : depart en (t,S(t,k)) avec t=k*delta_t
        Abs(l+1,k+1)=(k+l)*delta_t ; // les abscisses croissent
        Ord(l+1,k+1)=SS(k+l+1,k+1) ; // et les ordonnees decroissent : j=k=Cste
    end ; // on arrive en (T,S(T,k)) avec T=n*delta_t
    for l=1 :k // on repart de ce point
        Abs(n-k+l+1,k+1)=(n-l)*delta_t ; // avec les abscisses qui diminuent
        Ord(n-k+l+1,k+1)=SS(n-l+1,k-l+1) ; // et les ordonnees aussi
    end ; // la lignes n^o k abouti en (t,S(t,0)) avec t=(n-k)*delta_t
end ;
```

A noter que `xs2eps(1,'ArbreCox.eps')` et `xs2gif(1,ArbreCox.gif)` permettent de sauvegarder votre figure dans la norme `eps` et `gif` respectivement.

Utilisez les commentaires du code pour comprendre l'algorithme sur le dessin, pour une petite valeur de $n$, `n=10` par exemple. Combien de lignes polygonales sont-elles calculées ? Combien y-a-t-il de sommets dans chaque ligne polygonale ?

**Exercice 4.** **:** Tracer l'arbre de Cox pour diverses valeurs de $n$ et comparer avec l'arbre de la marche de Wiener. La marche CRR est considérée comme un meilleur modèle pour des cours d'actions que la marche de Wiener. Pour quelle raison, à votre avis ?

**Exercice 5. :** On souhaite à présent simuler un nombre $M$ de trajectoires de la marche CRR en utilisant la commande `rand(n,M,'uniform')` qui retourne une matrice `nxM` de nombres aléatoires uniformément distribués sur $[0,1]$.

Que retourne `int(p+rand(n,M))`, à votre avis (si $p \in ]0,1[$) ?

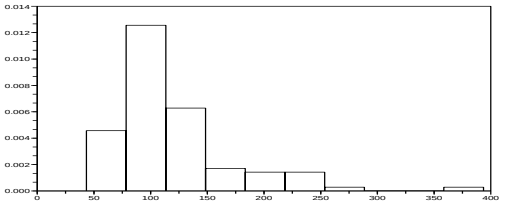
Et, si l'on pose `deltaJ=int(p+rand(n,M))`, que retourne `cumsum(deltaJ,"r") ?`

On suppose $n = 100$. Simuler $M = 40$ trajectoires en choisissant $p = 0.5$. En reproduisant la simulation avec d'autres valeurs de la volatilité $\sigma$, étudier et expliquer l'influence de ce paramètre sur la forme des trajectoires.

**Exercice 6. :** Simuler de même $M$ trajectoires de la marche de Wiener et comparer avec les trajectoires de CRR.

**Exercice 7. :** A l'aide de l'instruction `histplot`, représenter un histogramme des valeurs finales (on prendra par exemple $n = 200$) des trajectoires aléatoires de CRR ; pour cela, on simulera $M = 40$ trajectoires au moins et on pourra par exemple prendre `int(sqrt(M))` pour nombre de classes.

Histogramme des valeurs finales :




Quelle est la loi de probabilité des valeurs finales d'une marche de Wiener ? Expliquer pourquoi ?