Corrigé de l'exercice 3 du 24 mars

Question 1

Y étant la dérivée temporelle de X, on peut interpréter X_t comme la position au temps t d'un mobile évoluant dans le plan $\mathbb{R}^2$. L'équation (1) signifie alors que la vitesse du mobile évolue suivant un bruit brownien ; ou en d'autres termes, que le mobile est soumis à une accélération qui est un bruit blanc gaussien. D'un point de vue physiquement concret, cela signifie que le mobile est soumis à des forces aléatoires fluctuant très trapidement dans le temps, et qu'il n'y a pas de phénomène de frottement qui tende à le ramener à une vitesse nulle. Cela pourrait être une modélisation raisonnable pour, par exemple, un astéroïde dérivant dans l'espace.

Concernant la régularité de X, l'équation (1) nous dit que la vitesse Y sera un mouvement brownien. Nous savons que le mouvement brownien est $\mathcal{C}^0$ mais pas $\mathcal{C}^1$. En passant à la primitive, cela implique que la trajectoire $(X_t)_t$ est $\mathcal{C}^1$ mais pas $\mathcal{C}^2$. [∗]

Question 2

Voici un code possible :

```
function simulation (C, T)
% La fonction 'simulation' simule le système d'équations différentielles
% (1) et trace la trajectoire suivie par X dans R^2. 'C' est la matrice de
% covariance par unité de temps du bruit brownien auquel est soumise la
% vitesse, et 'T' est la durée de temps sur laquelle on effectue la
% simulation.

% 'N' est le nombre de pas de temps pour la discrétisation.
N = 1600;
% 'dt' est l'intervalle de temps entre deux instants de discrétisation.
dt = T / N;
% Allocation de mémoire pour (X_t)_t et pour (Y_t)_t. Noter que X (resp. Y)
% est une fonction à valeurs dans R^2, de sorte que ses valeurs successives
% seront stockées dans une matrice de 2 lignes appelée 'X' (resp. Y).
% Attention, la colonne i de la matrice des positions (ou des vitesses) ne
% correspondra pas au temps i*dt mais à (i-1)*dt, puisque l'indexation des
% matrices de Matlab commence à 1.
X = zeros(2,N+1);
Y = zeros(2,N+1);
% On calcule la matrice de Cholesky de C une fois pour toutes, et on
% l'appelle 'R'.
R = chol(C);

% Initialisation de X et Y au temps 0. (Ici cette étape est en fait
% superflue vu que X et Y partent de valeurs nulles).
X(:,1) = [0;0];
Y(:,1) = [0;0];
% Boucle de simulation par la méthode d'Euler stochastique. Le fait qu'on
```

[∗]. On peut en fait donner un sens plus fin à la notion de régularité d'une trajectoire, qui conduirait à dire que le mouvement brownien est essentiellement "$\mathcal{C}^{1/2}$" ; la trajectoire X est donc essentiellement de régularité $\mathcal{C}^{3/2}$ — plus exactement, $\mathcal{C}^{3/2-\varepsilon}$ pour tout $\varepsilon > 0$.

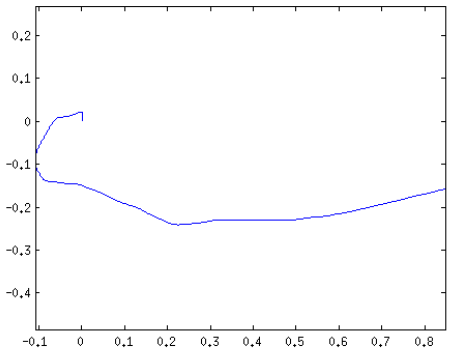
```
% ait ici un système d'équations ne change rien au principe. Attention, ici
% le bruit brownien correspond à un vecteur /colonne/ ; il faut donc
% transposer la méthode Cholesky. On prendra garde à ce que la matrice de
% Cholesky correspondant à dt*C est sqrt(dt)*R : il en effet une racine
% carrée pour dt dans la mesure où la matrice de base est quadratique en la
% matrice de Cholesky.
for i = 1:N
    X(:,i+1) = X(:,i) + Y(:,i) * dt;
    Y(:,i+1) = Y(:,i) + R' * randn(2,1) * sqrt(dt);
end

% On trace le résultat : la trajectoire suivie par X n'est autre que le
% graphe de la deuxième ligne (positions en y) de la matrice X contre sa
% première ligne (positions en x). Noter l'utilisation de « axis equal »
% pour avoir la même échelle sur les deux axes.
plot (X(1,:), X(2,:));
axis equal;

return
end
```

L'exécution m'a donné :

```
>> C = [3 1;1 2];
>> simulation(C,1)
```



Question 3

Voici un code possible :

```
function MCdistance (C, T, M)
% La fonction 'MCdistance' évalue la distance moyenne parcourue à vol
% d'oiseau au terme du temps 'T'. 'C' est la matrice de covariance par
% unité de temps du bruit gaussien, et 'M' le nombre de simulations de
% Monte-Carlo demandées.

% On calcule la matrice de Cholesky de C une fois pour toutes, qu'on
% appelle 'R'.
R = chol(C);

% Il s'agit d'un code extrêmement classique, que je ne commente donc guère.
S = 0;
```

```
S2 = 0;
for j = 1:M
    % La simulation de la distance à vol d'oiseau est donnée par la
    % sous-routine 'simulation_prMC' codée plus loin.
    distance = simulation_prMC(R,T);
    S = S + distance;
    S2 = S2 + distance * distance;
end

m = S / M;
v = S2 / M - m * m;
fprintf('Distance à vol d''oiseau moyenne : %.4f ± %.4f\n', ...
    m, 1.96*sqrt(v/M));
return


% Sous-routine 'simulation_prMC'
    function distance = simulation_prMC (R, T)
        % La fonction 'simulation_prMC' effectue le même travail que
        % 'simulation', à ceci près qu'au lieu de plotter la trajectoire de
        % X à la fin de la simulation, elle retourne la distance 'distance'
        % parcourue à vol d'oiseau. En outre, pour améliorer l'efficacité
        % lors de la boucle de Monte-Carlo, ce n'est pas la matrice C que
        % la fonction 'simulation_prMC' prend comme argument, mais
        % directement sa matrice de Cholesky 'R'.
        N = 1600;
        dt = T / N;
        X = zeros(2,N+1);
        Y = zeros(2,N+1);
        X(:,1) = [0;0];
        Y(:,1) = [0;0];
        for i = 1:N
            X(:,i+1) = X(:,i) + Y(:,i) * dt;
            Y(:,i+1) = Y(:,i) + R' * randn(2,1) * sqrt(dt);
        end
        % On calcule 'distance', qui est la norme euclidienne de X(:,N+1) -
        % X(:,1). Noter qu'on aurait aussi pu calculer cette norme en
        % utilisant la fonction 'norm' pré-programmée dans Matlab.
        distance = sqrt (sum ((X(:,N+1)-X(:,1)).^2));
        return
    end

end
```

L'exécution m'a donné :

```
>> MCdistance(C,1,600)
Distance à vol d'oiseau moyenne : 1.1768 ± 0.0513
```

Question 4

Pour appliquer la technique des variables de contrôle, nous avons besoin de calculer l'espérance des variables de contrôle. Notant $Z_2 := \int_0^T |Y_t|^2 \, dt$ la première variable de contrôle, commençons par calculer $\mathbb{E}(Y_2)$: d'après la formule de Fu-

bini [†],

$$\mathbb{E}(Z_2) \overset{\text{déf}}{=} \mathbb{E}\Big(\int_0^T |Y_t|^2\,dt\Big) = \int_0^T \mathbb{E}\big(|Y_t|^2\big)\,dt.$$

Or, la variable aléatoire Y_t étant la réalisation au temps t d'un mouvement brownien de covariance C par unité de temps, elle suit la loi $\mathcal{N}(tC)$, de sorte que (notant par des indices 'x' et 'y' les coordonnées respectives des vecteurs et matrices de $\mathbb{R}^2$),

$$\mathbb{E}(|Y_t|^2) = \mathbb{E}({Y_{t,x}}^2 + {Y_{t,y}}^2) = \mathbb{E}({Y_{t,x}}^2) + \mathbb{E}({Y_{t,y}}^2) = (tC)_{xx} + (tC)_{yy} = 2t\sigma^2.\ [‡]$$

Après intégration, il en sort que

$$\mathbb{E}(Z_2) = 2\int_0^T t\sigma^2\,dt = T^2\sigma^2.$$

Notons maintenant $Z_1 := \int_0^T |Y_t|\,dt$. De même que précédemment, pour calculer $\mathbb{E}(Z_1)$ il nous faut calculer $\mathbb{E}(|Y_t|)$. Or la variable aléatoire Y_t suit la loi $\mathcal{N}(\sigma^2 t\mathbf{I}_2)$, càd. que $(Y_{t,x}, Y_{t,y})$ suit la loi

$$d\mathbb{P}\big((Y_{t,x}, Y_{t,y}) = (x,y)\big) = \frac{1}{2\pi\sigma^2 t}\exp\!\Big(-\frac{x^2+y^2}{2\sigma^2 t}\Big)dxdy.$$

On remarque que cette densité dépend uniquement de $|(x,y)|$. Par conséquent, la probabilité que $|Y_t|$ soit compris entre r et $r+dr$, autrement dit la probabilité que $(Y_{t,x}, Y_{t,y})$ soit dans la couronne comprise entre les cercles centrés sur l'origine de rayons respectifs r et $r+dr$, est égale à

$$d\mathbb{P}\big(|Y_t| = r\big) = \frac{1}{2\pi\sigma^2 t}\exp\!\Big(-\frac{r^2}{2\sigma^2 t}\Big) \times 2\pi r dr,$$

où le facteur « $2\pi r dr$ » correspond à l'aire de la couronne précédemment évoquée. [§]

Il s'ensuit que

$$\mathbb{E}(|Y_t|) = \int_0^\infty r\,d\mathbb{P}(|Y_t| = r) = \frac{1}{\sigma^2 t}\int_0^\infty r^2 e^{-r^2/2\sigma^2 t}\,dr$$

Le changement de variables $r \leftarrow r\ /\ \sigma t^{1/2}$ transforme l'intégrale précédente en

$$\sigma t^{1/2}\int_0^\infty r^2 e^{-r^2/2}\,dr,$$

qui vaut par intégration par parties : (en écrivant « $r^2 e^{-r^2/2} = r \times r e^{-r^2/2}$ »),

$$\sigma t^{1/2}\int_0^\infty e^{-r^2/2}\,dr.$$

[†]. Qui est bien licite ici.

[‡]. Rappelons que pour un vecteur aléatoire *centré* Ξ de matrice de covariance Q, $\mathbb{E}(\Xi_i\Xi_j) = \mathbb{E}(\Xi_i)\,\mathbb{E}(\Xi_j) + \mathrm{Cov}(X_i, X_j) = 0 + Q_{ij} = Q_{ij}$.

[§]. Ici j'en profite pour faire remarquer une identité remarquable : si Y est un vecteur gaussien bidimensionnel centré réduit (càd. $Y \sim \mathcal{N}(\mathbf{I}_2)$), alors $|Y|^2$ suit une loi $\mathcal{E}xpon(1)$. [¶]

[¶]. De manière générale, pour $Y \sim \mathcal{N}(\mathbf{I}_d)$, la loi de $|Y|^2$ est appelée « loi $\chi^2(d)$ ». Ce que je dis est qu'il se trouve que la loi $\chi^2(2)$ coïncide avec la loi $\mathcal{E}xpon(1)$).

Or il est bien connu que

$$\int_{-\infty}^{+\infty} e^{-r^2/2}\, dr = \sqrt{2\pi}$$

(cela vient par exemple de ce que la densité de la loi normale centrée réduite est d'intégrale 1), donc au final,

$$\mathbb{E}(|Y_t|) = \sigma t^{1/2}\sqrt{2\pi}/2,$$

ce qui conduit après intégration à :

$$\mathbb{E}(Z_1) = \sigma T^{3/2}\sqrt{2\pi}/3.$$

On peut alors implémenter le code de réduction de la variance par variables de contrôles (multiples). Voici ce qu'une telle implémentation peut donner :

```
function MCdistance_ctrl (sigma, T, M)
% La fonction 'MCdistance_ctrl' fait le même travail que 'MCdistance', mais
% en utilisant les variables de contrôle suggérées par l'énoncé. Notez que
% puisqu'ici la matrice de covariance par unité de temps C doit être
% isotrope, on la passe en argument sous la forme du scalaire 'sigma' tel
% que C = sigma^2*I_2.

% Outre les sommes habituelles 'S' et 'S2', ici il va falloir tenir à jour
% la somme SZ des variables de contrôle (Z_2,Z_1) ainsi que la somme SZ2 de
% leurs ‘‘carrés’’ (incluant les produits croisés), et aussi la somme SP des
% produits entre les variables aléatoires distance et (Z_2,Z_1).
S = 0;
S2 = 0;
SZ = zeros(1,2);
SZ2 = zeros(2,2);
SP = zeros(1,2);

for j = 1:M
    % Dans cette version la sous-routine de simulation, que j'appelle ici
    % 'simulation_prMC_c', renvoie non seulement la distance parcourue à
    % vol d'oiseau mais aussi la valeur des variables de contrôle.
    [distance,Z2,Z1] = simulation_prMC_c(sigma,T);
    % On maintient S et S2, mais aussi SZ, SZ2 et SP.
    S = S + distance;
    S2 = S2 + distance * distance;
    SZ = SZ + [Z2,Z1];
    SZ2 = SZ2 + [Z2,Z1]' * [Z2,Z1];
    SP = SP + distance * [Z2,Z1];
end

% Dans un premier temps, on en déduit (une estimation de) la covariance
% 'CovCtrl' entre distance et (Z_2,Z_1), ainsi que de la matrice de
% covariance 'VarCtrl' de (Z2,Z1). Notez que cela impose au passage de
% calculer les moyennes empiriques respectives 'm' et 'mCtrl' de distance
% et (Z_2,Z_1).
m = S / M;
mCtrl = SZ / M;
CovCtrl = SP / M - m * mCtrl;
VarCtrl = SZ2 / M - mCtrl' * mCtrl;
```

```
% On en déduit (une estimation de) le vecteur optimal lambda tel que
% distance-lambda*(Z_2,Z_1)' ait une variance minimale, à l'aide de la
% formule du cours. Rappelons que sous Matlab, si 'V' est un vecteur-ligne
% et 'M' une matrice carrée de même dimension, « V / M » signifie « VM^{-1}
% » (*1).
lambda = CovCtrl / VarCtrl;

% Maintenant, il faut savoir ce qu'on aurait trouvé comme somme et somme
% des carrés si on avait utilisé la ‘‘bonne’’ variable, à savoir
% distance-lambda*(Z_2,Z_1)'. J'appelle 'Sbon' et 'S2bon' les quantités en
% question. Ce qui est remarquable avec la méthode des variables de
% contrôle, c'est qu'il n'y a rien à recalculer: on peut déduire Sbon et
% S2bon des sommes maintenues dans notre boucle!
Sbon = S - lambda * SZ';
S2bon = S2 - 2 * lambda * SP' + lambda * SZ2 * lambda';
% On en déduit l'intervalle de confiance sur la ‘‘bonne’’ variable.
mbon = Sbon / M;
vbon = S2bon / M - mbon * mbon;
margebon = 1.96 * sqrt (vbon / M);
% Comme on connait exactement l'espérance 'EZ' de (Z_2,Z_1), on en déduit
% enfin l'intervalle de confiance sur la variable distance.
EZ = [T^2*sigma^2, sigma*T^(3/2)*sqrt(2*pi)/3];
estimateurvrai = mbon + lambda * EZ';
margevrai = margebon;
fprintf('Distance à vol d''oiseau moyenne %.4f ± %.4f\n', ...
    estimateurvrai, margevrai);
return



% Sous-routine 'simulation_prMC_c'
    function [distance, Z2, Z1] = simulation_prMC_c (sigma, T)
        % La fonction 'simulation_prMC_c' effectue le même travail que
        % 'simulation_prMC', à ceci près qu'elle retourne aussi les valeurs
        % 'Z2' et 'Z1' des variables de contrôle respectives Z_2 et Z_1 ;
        % et qu'elle prend en argument sigma et non plus R pour le bruit
        % brownien, vu qu'ici ce bruit est isotrope.
        N = 1600;
        dt = T / N;
        X = zeros(2,N+1);
        Y = zeros(2,N+1);
        X(:,1) = [0;0];
        Y(:,1) = [0;0];
        for i = 1:N
            X(:,i+1) = X(:,i) + Y(:,i) * dt;
            % Pour simuler l'incrément d'un mouvement brownien isotrope,
            % notez que « randn(2,1) » simule un vecteur isotrope de
            % covariance I_2, de sorte qu'il n'y a plus qu'à multiplier par
            % le scalaire sigma.
            Y(:,i+1) = Y(:,i) + sigma * randn(2,1) * sqrt(dt);
        end
        distance = sqrt( sum((X(:,N+1)-X(:,1)).^2));

        % On calcule mainteant Z2 et Z1. Bien entendu, on n'a pas accès à
        % la valeur exacte de ces intégrales, mais seulement à une version
```

```
        % discrétisée. J'utilise ici la méthode d'intégration des trapèzes,
        % ce qui explique que les points extrêmaux de l'intervalle de temps
        % soient affectés chacun d'un coefficient 1/2. On commence pour
        % cela par calculer les vecteurs "vitesses1" et "vitesses2" des
        % normes des vitesses et de leurs carrés. Notez que la fonction
        % 'sum' de Matlab, quand on l'applique à un matrice non-ligne,
        % calcule le vecteur-ligne des sommes de chaque colonne. (*2)
        vitesses2 = sum(Y.^2);
        vitesses1 = sqrt(vitesses2);
        Z2 = dt * (vitesses2(1) / 2 + sum(vitesses2(2:N)) + ...
            vitesses2(N+1) / 2);
        Z1 = dt * (vitesses1(1) / 2 + sum(vitesses1(2:N)) + ...
            vitesses1(N+1) / 2);
        return
    end

end


% (*1) Le sens de « V / M » est en fait un tout petit peu plus général, car
% y a aussi une signification dans certains cas où M n'est pas inversible.

% (*2) Pour forcer ce comportement également dans le cas d'une
% matrice-ligne 'L', il faut préciser « sum(L,1) » (le « 1 » signifie qu'on
% somme L selon la première direction, à savoir colonne par colonne).
```

L'exécution m'a donné :

```
>> tic; MCdistance_ctrl(1,1,1200); toc;
Distance à vol d'oiseau moyenne 0.7265 ± 0.0041
Elapsed time is 5.665063 seconds.
```

à comparer à la version sans variable de contrôle :

```
>> tic; MCdistance(eye(2),1,1200); toc;
Distance à vol d'oiseau moyenne : 0.7499 ± 0.0215
Elapsed time is 7.830656 seconds.
```

La réduction de la variance est donc extrêmement marquée : on gagne un facteur 27 en efficacité par simulation ![|||]

[|||]. Le fait que la simulation soit plus *rapide* pour `MCdistance_ctrl` que pour `MCdistance` ne provient pas de la méthode des variables de contrôle, mais simplement du fait que dans le programme avec variable de contrôle nous avons utilisé que la matrice de covariance du bruit brownien était obligatoirement scalaire, ce qui accélère les calculs (alors que dans le cas général on traitait des matrices non scalaires). En utilisant la même astuce dans `MCdistance`, je trouve un temps d'exécution de l'ordre de 5,4s, donc un peu plus rapide, comme on pouvait s'y attendre dans la mesure où il y a moins de calculs à effectuer quand on n'utilise pas les variables de contrôle.