

Projet 601

Optimisation de la production d'un barrage hydraulique

Aurélien Pardo, Simon Marsan et Julien Ducrey

Mars 2022

Table des matières

1	Introduction	3
1.1	Fonctionnement d'un barrage hydraulique	4
1.2	Modélisation mathématique du problème	5
1.2.1	Puissance électrique et facteurs de productions contrôlables	5
1.2.2	Équation de programmation dynamique de Bellman	5
1.2.3	Application de l'équation de Bellman, au problème d'optimisation de la production du barrage	6
1.3	Objectifs du projet	7
2	Problème de la pyramide des nombres	8
2.1	Présentation du problème	8
2.2	Présentation de l'algorithme de remonté	8
2.3	Présentation de la résolution sur Matlab	9
3	Résolution pour un barrage avec 1 retenue	11
3.1	Résolution avec entrée d'eau constante	11
3.1.1	Présentation du code, de la résolution numérique en Matlab	11
3.1.2	Quelques exemples de tests du code	13
3.2	Résolution avec entrée d'eau périodique	17
3.2.1	Présentation du code et de la résolution numérique en Matlab	17
3.2.2	Quelques exemples de tests du code	18
3.3	Résolution avec une crue dans l'entrée d'eau	22
3.3.1	Présentation du code et de la résolution numérique en Matlab	22
3.3.2	Quelques exemples de tests du code	23
3.4	Résolution ajoutant la prise en compte des tarifs horaires de l'électricité	26
3.4.1	Présentation du code et de la résolution numérique en Matlab	26
3.4.2	Quelques exemples de tests du code	28
3.5	Résolution probabiliste, en générant une chaîne de Markov	33
3.5.1	Génération aléatoire d'une Chaîne de Markov	33
3.5.2	Application aux problèmes d'optimisation de la production électrique du barrage hydraulique	35
4	Résolution pour un barrage à 2 retenues	48
4.1	Résolution avec entrées d'eaux constantes	48
4.1.1	Présentation du code et de la résolution numérique en Matlab	48
4.1.2	Quelques exemples de tests du code	50
4.2	Résolution avec ajouts des entrées d'eaux non constantes	55
4.2.1	Présentation du code et de la résolution numérique en Matlab	55
4.2.2	Quelques exemples de tests du code	57
4.3	Résolution avec ajouts des tarifs horaires de l'électricité	60
4.3.1	Présentation du code et de la résolution numérique en Matlab	60
4.3.2	Quelques exemples de tests du code	62
4.4	Résolution en simulant les entrées d'eaux des intempéries par des chaînes de Markov	65

4.4.1	Présentation du code et de la résolution numérique en Matlab	65
4.4.2	Quelques exemples de tests du code	68
5	Résolution pour un barrage à 3 retenues	73
5.1	Résolution avec des entrées d'eaux constantes	73
5.1.1	Présentation du code et de la résolution numérique en Matlab	73
5.1.2	Quelques exemples de tests du code	77
5.2	Résolution matricielle	79
5.2.1	Présentation du code et de la résolution numérique en Matlab	80
5.2.2	Quelques exemples de tests du code	82
5.3	Résolution avec ajout des entrées d'eaux non constantes et des tarifs horaires de l'électricité	84
5.3.1	Présentation du code et de la résolution numérique en Matlab	84
5.3.2	Quelques exemples de tests du code	88
5.3.3	Remarques	89
6	Bibliographie	91

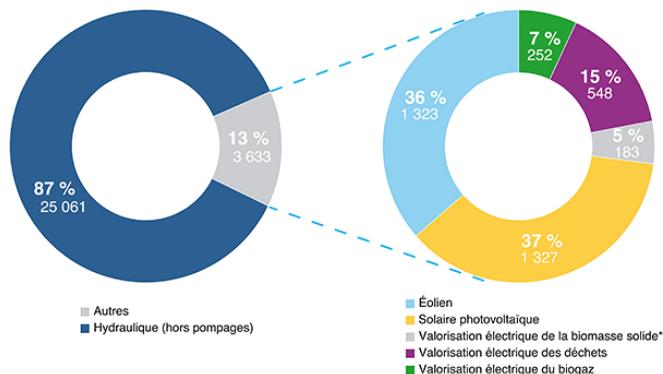
1 Introduction

Depuis le Moyen-Âge, l'énergie hydraulique fournie par les cours d'eaux est exploitée au moyen des moulins à eau. Depuis le milieu du *XIX^{ème}* siècle, elle est également utilisée pour produire de l'électricité, grâce aux premiers barrages hydrauliques. Aujourd'hui, l'énergie hydroélectrique représente la principale source d'énergie électrique renouvelable de la planète.

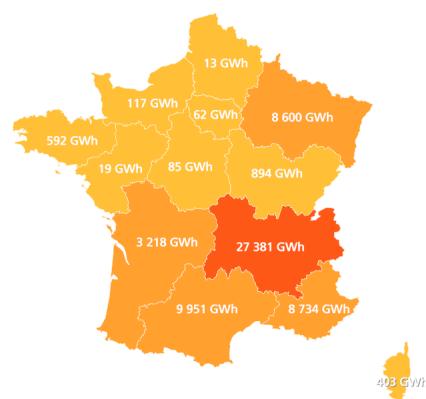
Le plus grand barrage du monde est actuellement celui des Trois Gorges, il est situé en Chine et exploite les eaux du Yang-Tsé-Kiang ou Fleuve Bleu.



Les reliefs naturels et les nombreux lacs, torrents et rivières à forts dénivellés font des régions montagneuses un endroit parfaitement propice aux installations de ces barrages hydrauliques.



En 2019, avec ses 25 061 GWh, l'hydro-électricité représentait près de 87,3%[3], de la production électrique renouvelable de la région Auvergne Rhône-Alpes, faisant de celle-ci la première région française en termes de production hydraulique.



Hydraulique : production par région en 2019

Source RTE - Bilan électrique 2019

© EDF

1.1 Fonctionnement d'un barrage hydraulique

Pour fonctionner, un barrage hydraulique exploite l'énergie potentielle de l'eau au sein du réseau hydrographique et se sert de son mouvement pour produire de l'électricité.

Un site de production d'énergie hydraulique se compose principalement en trois parties. Premièrement, il possède un barrage, qui permet de former par accumulation une importante retenue d'eau. Deuxièmement, il comporte aussi une centrale hydraulique, lieu de production de l'électricité et enfin un transformateur, équipé de lignes à haute tension.

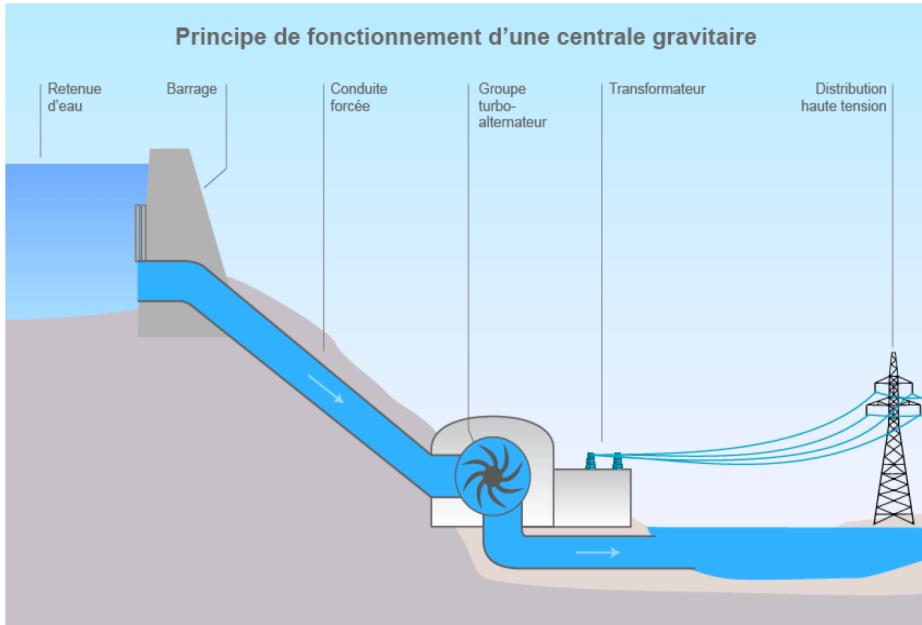


Schéma centrale hydraulique.

Le principe de fonctionnement est le suivant : le barrage retient l'écoulement naturel de l'eau, ce qui alimente le lac de sa retenue, conservant ainsi à disposition toute l'énergie potentielle de l'eau.

Une fois que le niveau de la retenue est suffisant, ou en fonction des besoins, des vannes sont ouvertes, envoyant ainsi l'eau à travers de gigantesques tuyaux métalliques appelés conduites forcées.

Ces conduites guident le flux d'eau jusqu'à la centrale hydraulique, située en aval. Le dénivelé entre la retenue et la centrale permet aux mouvements de l'eau de prendre de la vitesse, transformant ainsi progressivement son énergie potentielle en énergie cinétique.

À son arrivée dans la centrale, la force motrice accumulée de l'eau est employée pour faire tourner une turbine, qui va ensuite entraîner un alternateur. C'est ce mouvement de l'alternateur qui est à l'origine de la production d'un courant électrique alternatif.

À l'intérieur de l'alternateur, le mouvement rotatoire engendré par celui qui est issu de la turbine provoque des interactions entre les électroaimants du rotor et les bobines de fils de cuivre du stator.

Ces interactions créent un déséquilibre des charges électriques, des atomes de cuivre, générant ainsi un champ magnétique.

Les électrons situés dans les fils de cuivre vont alors se déplacer de manière à rétablir l'équilibre des charges, et le mouvement rotatoire du rotor va permettre d'entretenir et de déplacer ce déséquilibre, faisant alors alterner le mouvement de ces électrons, ce qui correspond à la génération d'un courant électrique alternatif.

Enfin, le transformateur modifie la tension de ce courant électrique en la rehaussant entre 225 et 400 mille volts, pour pouvoir le transporter sur les lignes à haute tension et ainsi l'injecter dans le réseau de distribution d'EDF.

Quand à l'eau turbinée, elle est évacuée vers l'aval de la centrale, dans le lit naturel du cours d'eau, par le canal de fuite.

L'ouverture des vannes est bien sûr conditionnée par des variables tels que la météo, le niveau de remplissage du barrage, le besoin en électricité, mais aussi la plage du tarif horaire de l'électricité.

1.2 Modélisation mathématique du problème

1.2.1 Puissance électrique et facteurs de productions contrôlables

On souhaite optimiser la production d'électricité d'un barrage hydraulique. Or, en se basant sur la présentation du fonctionnement d'un de ces barrages dans la section précédente, on peut déduire que la quantité d'électricité produite instantanément dépend directement de deux principaux facteurs.

Le premier facteur est la hauteur de chute de l'eau dans les conduites forcées car on exploite son énergie potentielle, qui est également fonction de cette hauteur de chute.

Le second facteur est le débit moyen du flux d'eau lorsqu'il atteint la turbine à la sortie des conduites forcées. Le débit de ce flux dépend lui-même de deux autres facteurs.

Le premier de ces facteurs est le niveau de la hauteur d'eau dans la retenue du barrage, car elle contribue à faire augmenter la pression à l'entrée des conduites forcées. Ainsi, elle accélère la vitesse de chute du flux d'eau et, par suite, sa vitesse d'arrivée sur la turbine.

Le second facteur est la quantité d'eau qui est injectée dans les conduites forcées, c'est à dire le contrôle appliqué sur le volume contenu dans la retenue du barrage.

En effet, l'augmentation de la quantité d'eau libérée dans les conduites forcées contribue également à faire augmenter la masse de ce flux d'eau et donc l'intensité de la force de pesanteur, qui va s'appliquer dessus au cours de sa chute.

Ce phénomène favorise également un accroissement rapide de la vitesse de chute du flux d'eau et ainsi, toujours par suite, sa vitesse d'arrivée sur la turbine.

À partir de ces constats, on dispose donc de l'expression de la puissance électrique suivante :

$$P = H * \rho * g * \mu * Q$$

Avec P :Puissance électrique en Watt, W

H :Hauteur en mètres, m

ρ :Masse volumique de l'eau en Kg/m^3 , $\rho = 1000Kg/m^3$

g : Constante d'accélération de la pesanteur, $g=9.81 m/s^2$

μ :rendement de la centrale, $\mu = 0.75$ environ

Q :Débit moyen m^3/s

À l'aide de ces constats et de cette expression, on comprend que les deux facteurs, que l'on a la possibilité de réguler, sont le niveau de la hauteur d'eau H contenue dans la retenue du barrage ainsi que la quantité d'eau Q envoyée dans les conduites forcées à un instant n .

La résolution du problème consiste donc à trouver la succession de contrôles optimaux sur le volume contenu dans la retenue du barrage, qui permet de maximiser la production électrique de la centrale hydraulique. Il s'agit de trouver un équilibre parmi toutes les combinaisons de contrôles physiquement possibles, entre une grande hauteur d'eau et un grand débit, tout en prenant en compte la mise à jour du volume de la retenue du barrage.

1.2.2 Équation de programmation dynamique de Bellman

Afin de résoudre notre problème d'optimisation, nous avons utilisé un algorithme de programmation dynamique.

Le principe de la programmation dynamique[6] a été découvert dans les années 1940 par Richard Bellman[2], un mathématicien américain. Il consiste en une équation, appelée équation de Bellman ou équation de programmation dynamique, qui permet de représenter la résolution d'un problème d'optimisation sous forme récursive.

On considère un système dynamique discret dont l'état à chaque instant est donné par :

$$\forall n \in \llbracket 0; N - 1 \rrbracket, x_{n+1} = f_n(x_n, u_n) \text{ et } x_0 = \bar{x}$$

L'instant final $N \in \mathbb{N}^*$, s'appelle *l'horizon* du problème. À chaque instant $n \in \llbracket 0; N - 1 \rrbracket$, x_n est *l'état* du système et u_n est le *contrôle*, c'est à dire la décision prise à l'instant n . f_n est la *dynamique* du problème

à l'instant n .

On suppose que l'état du système vit dans un ensemble X fixé, c'est à dire que $\forall n \in \llbracket 0; N-1 \rrbracket, x_n \in X$. Le contrôle u_n vit dans un ensemble \mathbb{U} et $f_n : X \times \mathbb{U} \rightarrow X$. La condition initiale $\bar{x} \in X$ est fixée.

Dans un problème de contrôle optimal discret (ou d'horizon fini), étant donné une condition initiale \bar{x} , on définit pour tout contrôle $u \in \mathbb{U}$, une fonction de gain :

$$G(\bar{x}, u) = \sum_{n=0}^{N-1} L_n(x_n, u_n) + g(x_N)$$

La quantité $L_n(x_n, u_n)$ est appelée le *gain courant* à l'instant n , g étant le *gain terminal*. Ce sont des fonctions $L_n : X \times \mathbb{U}^N \rightarrow \mathbb{R}$ et $g : X \rightarrow \mathbb{R}$.

L'objectif est de maximiser le gain parmi tous les contrôles possibles $u \in \mathbb{U}^N$, c'est à dire de calculer :

$$\max_{u \in \mathbb{U}^N} \sum_{n=0}^{N-1} L_n(x_n, u_n) + g(x_N)$$

On obtient ainsi cette équation de programmation dynamique en horizon fini sur l'ensemble des suites à N éléments de \mathbb{U} .

On cherche donc la combinaison, ou bien la suite des N contrôles optimaux qui permettent, à partir de la condition initiale \bar{x} , de maximiser la fonction gain G .

Comme il est impossible de tester une par une, en un temps de calcul raisonnable, toutes les combinaisons possibles des contrôles et de conserver la suite optimale, on utilise un algorithme de programmation dynamique récursif afin de trouver cette solution optimale.

1.2.3 Application de l'équation de Bellman, au problème d'optimisation de la production du barrage

On présente maintenant l'algorithme de programmation dynamique avant de l'appliquer au problème du barrage.

$\forall x \in X$ et $\forall p \in \llbracket 0; N-1 \rrbracket$, on définit la fonction valeur du problème comme :

$$V(p, x) = \max_{u_p \in \mathbb{U}^p} \sum_{n=p}^{N-1} L_n(x_n, u_n) + g(x_N)$$

où l'état $x_p = (x_n)_{n=p}^N$, est défini par récurrence :

$$\begin{cases} x_p &= \bar{x} \\ x_{n+1} &= f_n(x_n, u_n), \forall n \in \llbracket p; N-1 \rrbracket \end{cases}$$

Notons que la quantité que nous cherchons à maximiser est $V(0, \bar{x})$.

Le principe de la programmation dynamique, repose sur l'équation de Bellman suivante :

Théorème 1.2 (Programmation dynamique). Pour tout $x \in X$, on a :

$$\begin{cases} V(p, x) &= \max_{u_p \in \mathbb{U}^p} \sum_{n=p}^{N-1} L_n(x_n, u_n) + g(x_N), \forall p \in \llbracket 0; N-1 \rrbracket \\ V(N, x) &= g(x) \end{cases}$$

On applique maintenant l'équation de Bellman au problème du barrage. Pour cela, on considère les éléments suivants :

- N : horizon du problème.
- X_n : état du système à l'instant n , représente le volume d'eau dans le barrage, à la date n .
- U_n : contrôle appliqué au système, à l'instant n . Correspond au débit sortant.
- W_n : débit entrant dans le barrage, à la date n .

Évolution du système sur $\llbracket n; n + 1 \rrbracket$:

$$X_{n+1} = X_n + W_n - U_n = f_n(X_n, U_n)$$

où $f_n(x, u) = x + W_n - u$

Soit $V(n, x)$ la production d'électricité entre n et N lorsque $X_n = x$. Nous cherchons à maximiser $V(0, x)$, c'est à dire à trouver les contrôles optimaux sur les débits sortants pour maximiser la production d'électricité.

La production d'électricité sur $\llbracket n; n + 1 \rrbracket$ est :

$$L(X_n, U_n) = X_n * \rho * g * \mu * U_n$$

On en déduit enfin l'algorithme suivant :

$$\begin{cases} V(n, x) &= \max_{u \in \mathbb{U}^N} (L(x, u) + V(n + 1, f_n(x, u))), \forall n \in \llbracket 0; N - 1 \rrbracket. \\ V(N, x) &= 0. \end{cases}$$

Cet algorithme est récursif car il calcule la valeur de V en (n, x) en faisant appel à la valeur de V en $(n + 1, x)$, la fonction V est donc construite récursivement avec des appels à elle-même à chaque itération de l'algorithme.

On peut observer que cet algorithme de programmation dynamique fonctionne sur un principe de remontée. Pour chaque état du système on part d'une valeur de production d'électricité donnée à l'horizon du problème et on remonte. Cela correspond à une subdivision du problème initial en N sous-problèmes, résolus successivement.

Comme à chaque étape n on cherche le meilleur contrôle appliquée à l'instant précédent, parmi tous les contrôles potentiels pour maximiser la fonction valeur de production V , on supprime ainsi plus rapidement un nombre de plus en plus grand de possibilités, ce qui explique que cet algorithme peut-être compilé en un temps très raisonnable.

1.3 Objectifs du projet

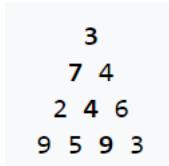
- Implémenter l'algorithme de programmation dynamique et tracer les graphes correspondants à l'évolution de la production d'électricité dans les cas où :
 1. le débit entrant est constant.
 2. le débit entrant évolue de façon sinusoïdale.
 3. le débit entrant contient une crue.
- Modifier l'algorithme précédent en prenant en compte l'évolution du tarif de l'électricité suivant l'heure de la journée et optimiser le gain financier.
- Supposer que le débit entrant est aléatoire et évolue comme une chaîne de Markov.
Implémenter l'équation de programmation dynamique stochastique associée.
- Supposer que le barrage possède maintenant deux retenues et modifier l'équation de programmation dynamique en conséquence.
Implémenter l'algorithme correspondant, pour tous les cas déjà traités, avec une seule retenue.

2 Problème de la pyramide des nombres

Pour mieux comprendre le principe de la programmation dynamique en optimisation nous avons commencé par résoudre un problème plus simple que celui du barrage.

2.1 Présentation du problème

Il s'agit du problème de la pyramide des nombres.^[1] On considère une pyramide semblable à celle représentée ci-dessous et l'on souhaite descendre la pyramide en passant par une seule case de chaque étage, on est obligé de descendre à chaque déplacement et l'on doit choisir les cases traversées de manière à maximiser la somme des nombres présents sur les cases traversées.



Le chemin optimale est ici représenté en gras.

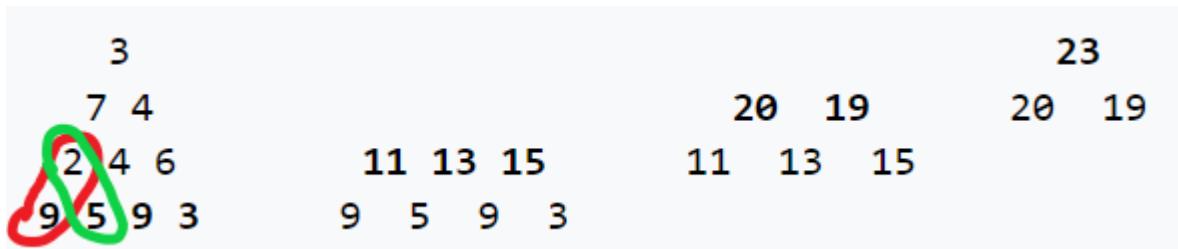
Pour résoudre ce problème, on pourrait éventuellement essayer de tester tous les chemins possibles et de simplement choisir celui qui donne la somme maximale comme chemin optimal. Mais étant donné le fait que le nombre de possibilités est de 2^N , où N représente le nombre d'étages de la pyramide, on en déduit rapidement que le temps de résolution, lorsque n devient grand, tendrait rapidement vers l'infini.

On utilise donc la programmation dynamique.

2.2 Présentation de l'algorithme de remonté

L'algorithme repose sur le principe suivant. On part de l'étage le plus bas de la pyramide et on considère les deux premières valeurs adjacentes.

On compare les valeurs de chacune de leurs sommes, avec la valeur située en amont de celles-ci, sur l'étage supérieur de la pyramide. Comme coloré en rouge et en vert sur l'image ci-dessous.



On choisit le maximum, que l'on place dans une copie de la pyramide.

On répète ensuite l'opération en se décalant d'un cran vers la droite, pour toutes les paires de valeurs de ce même étage de la pyramide.

On fait ensuite de même avec l'étage supérieur, dans la copie de la pyramide et ainsi de suite, jusqu'à atteindre le même étage que celui du sommet de la pyramide de départ.

Cet algorithme consistant en une remontée, on choisit à la fin quel était, à l'étage précédent, le choix optimal. On supprime ainsi un grand nombre de choix non optimaux et on ne garde que les optimaux entre l'avant-dernier étage et le dernier.

On se retrouve ainsi avec un sous-problème, qui est en fait le même que celui de départ, mais pour une pyramide plus petite. C'est à dire la même pyramide sans le dernier étage, car on connaît déjà les choix optimaux une fois parvenu au dernier étage de la sous-pyramide.

Ainsi, en répétant $N-1$ fois le même procédé, on reconstruit une pyramide de même format que celle de départ, dont les coefficients à l'étage $i \in \llbracket 1, N \rrbracket$ sont les valeurs des sommes optimales qu'il est possible d'obtenir en sommant les meilleurs choix de valeurs, pour tous les chemins, entre l'étage i et dernier de la

pyramide.

La valeur située au sommet de la copie de la pyramide est alors la valeur de somme maximale, qu'il est possible d'obtenir par une des possibilités de chemins descendant la pyramide.
Pour retrouver ce chemin, on choisit à chaque étage de la copie de la pyramide la valeur maximale.

Il est important de noter que le problème peut parfois avoir plusieurs solutions, par exemple dans le cas où la somme maximale peut-être obtenue grâce à plusieurs chemins de descentes différents.

2.3 Présentation de la résolution sur Matlab

Afin de résoudre notre problème de manière numérique sur Matlab, nous avons choisi d'utiliser une structure de , pour représenter notre pyramide, car c'est la structure de données la plus pratique et la plus simple d'utilisation sur Matlab.

Nous avons donc commencé par écrire notre pyramide sous cette forme :

```
pyramide1=[[3,0,0,0];
            [7,4,0,0];
            [2,4,6,0];
            [9,5,9,3]];
```

Nous avons ensuite écrit la fonction suivante :

```
function [solution,chemin]=chemin_max(pyramide)
n=size(pyramide);
intermediaire=zeros(n);intermediaire(1,:)=pyramide(n(1),:);
chemin=zeros(1,n(1));chemin(1,1)=1;l=1;
for i=n(1)-1:-1:1
    courant=[pyramide(i,:)];
    for j=1:i
        courant(j)=max(intermediaire(l,j)+courant(j),intermediaire(l,j+1)+courant(j));
    end
    l=l+1;
    intermediaire(1,:)=courant;
end
solution=intermediaire(n(1),1);
for i=2:n(1)
    sous=intermediaire(n(1)+1-i,chemin(i-1));
    sur=intermediaire(n(1)+1-i,chemin(i-1)+1);
    if max(sous,sur)==sous
        chemin(i)=chemin(i-1);
    else
        chemin(i)=chemin(i-1)+1;
    end
end
end
```

Elle prend une pyramide sous forme matricielle en paramètre et retourne la valeur de somme optimale ainsi que le chemin correspondant.

La fonction se divise en deux principales parties. La première est la construction de la copie de la pyramide décrite précédemment et la seconde est la récupération du chemin optimal à partir de cette copie de la pyramide.

Dans la première partie du code, on initialise les données :

n : la taille de la pyramide.

intermediaire : une matrice de zéros, de même taille que celle de la pyramide. On affecte à sa dernière ligne la dernière ligne de la pyramide, comme point de départ de l'algorithme de remonté.

chemin : un vecteur rempli de zéros, de même longueur, que le nombre d'étage de la pyramide. Son premier coefficient est toujours 1 puisque l'on passe toujours par le sommet.

La première boucle for est indexée sur les étages de la pyramide et tourne en sens inverse car elle remonte.

On récupère ensuite la ligne de la pyramide correspondante, au tour de boucle.

La seconde boucle for réécrit cette ligne, comme expliqué précédemment. On arrête cette boucle à i, car il n'y a que des zéros au delà de la diagonale puisque les pyramides sont représentées par des matrices triangulaires inférieures.

On incrémente la variable *l*, afin de passer à la ligne suivante dans la matrice copie *intermediaire*, car l'ordre des lignes est inversé entre *intermediaire* et *pyramide*.

Voir ci-dessous.

```
intermediaire =
9   5   9   3
11  13  15  0
20  19  0   0
23  0   0   0
```

On stocke ensuite la nouvelle ligne obtenue dans *intermediaire* et on passe à la ligne suivante.

Dans la seconde partie du code, on récupère la somme maximale, qui est alors le coefficient, tout en bas à gauche, de la matrice *intermediaire*.

Ensuite, pour reconstruire le chemin, on utilise une boucle for indexée sur chaque étage.

On prend à l'étage supérieur les deux coefficients de *intermediaire*, situés au dessus de l'emplacement du coefficient optimal récupéré au tour de boucle précédent. On fait un test avec un if, pour savoir lequel est le plus grand et on récupère sa coordonnée de colonne qui indique le chemin optimal.

On teste notre fonction, à l'aide de l'appel suivant :

```
[sol1,chem1]=chemin_max(pyramide1)
```

On obtient alors les résultats suivants :

```
>> pyramide
```

```
sol1 =
```

```
23
```

```
chem1 =
```

```
1   1   2   3
```

On observe que l'on obtient bien la valeur de la somme optimale, ainsi que le chemin qui permet de l'obtenir, représenté sous forme d'un tableau d'entier.

Dans ce tableau, l'indice de la case désigne l'étage de la pyramide et la valeur de la case désigne le rang de l'élément de l'étage, en partant de la gauche, qu'il faut choisir pour construire le chemin optimal.

Il est important de noter que si le problème possède plusieurs solutions optimales, l'algorithme présenté ci-dessus n'en donnera qu'une seule, qui correspondra au chemin situé le plus en haut à gauche dans la pyramide.

Au cours des séances de TD nous avons également vu d'autres problèmes d'optimisation faisant appel à la programmation dynamique pour leur résolution, comme par exemple le problème du voyageur de commerce.

3 Résolution pour un barrage avec 1 retenue

On se concentre maintenant sur notre problème d'optimisation de la production électrique, dans le cas le plus simple, c'est à dire pour un barrage n'ayant qu'une seule retenue d'eau.

3.1 Résolution avec entrée d'eau constante

Pour cette première résolution, on se donne une entrée d'eau journalière W , constante sur toute la durée de production N .

3.1.1 Présentation du code, de la résolution numérique en Matlab

Pour commencer, voici les paramètres, de notre problème :

```
L = taille maximale de la retenue du barrage.
o = taille maximale de l'ouverture des conduites forcées.
N = durée de la période de production d'électricité.
W = entrée d'eau journalière, dans le barrage.
vol_depart = valeur arbitraire du volume présent dans le barrage au
départ, pour pouvoir construire le graphe.
```

La saisie de ces différents paramètres par l'utilisateur est essentielle afin de pouvoir utiliser l'algorithme de programmation dynamique, présenté précédemment dans ce document.

Cette saisie permet également d'adapter la résolution numérique et sa représentation, pour approximer au mieux la situation réelle que l'on veut simuler.

Voici le code de la fonction, qui effectue la résolution numérique et qui renvoie tous les résultats :

```
function [prod_max,programme]=Optimise(L,o,N,W,vol_depart)
% Données
rho=1000;
g=9.80665;
mu=0.75;
% Définition des matrices
V=zeros(N+1,L+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1); % Matrices des valeurs des contrôles, liées aux valeurs de production, coefficient par coefficient.
Volume=[0:L];
% Vecteur des valeurs des niveaux de remplissages du barrage.
Controle=[0:o];
% Construction des Matrices U et V, par remontée.
for i=N:-1:1 % Boucle pour le temps
    for j=1:L+1 % Boucle pour le volume
        possible=zeros(1,o+1);
        for k=1:o+1 % Boucle pour les contrôles possibles.
            if(Volume(j)+W-Controle(k)>=0) % Pour être sûr que le contrôle est physiquement possible, il reste de l'eau dans le barrage.
                possible(1,k) = Volume(j)*rho*g*mu*Controle(k) + V(i+1,max(min(L,Volume(j)+W-Controle(k)),1));
            end
        end
        [V(i,j),indice]=max(possible);
        U(i,j)=Controle(indice); % Décalage, pour avoir le contrôle u=0, à cause de l'indexation des tableaux dans Matlab.
    end
end
[prod_max,rang]=max(V(1,:));
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX Volume_opt, grâce aux matrices U et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+W-Controle_opt(i-1)),0);
    Controle_opt(i)=U(i,vol_courant(i)+1);
end
% Récupération des résultats
programme=Controle_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
plot(0:N,vol_courant);
end
```

Cette fonction prend les paramètres précédents en entrée, et retourne la valeur du maximum de production atteignable parmi toutes les valeurs possibles de volume de départ et pour tous les contrôles physiques.

ment applicables.

La fonction retourne également la suite des contrôles à appliquer, pour optimiser la production et ainsi obtenir la valeur de production maximale pour le volume de départ donné en paramètre.

La fonction se découpe en plusieurs étapes, qui sont clairement délimitées dans le code par les commentaires colorés en vert.

Dans la première partie on initialise les 3 variables ρ , g et μ , qui vont représenter les constantes des grandeurs physiques qui interviennent dans notre problème. On leur affecte une valeur approchée de leur valeur réelle.

Dans la seconde partie du code, on initialise les matrices et les vecteurs qui vont servir dans l'algorithme de programmation dynamique.

La matrice V contient les valeurs maximales de production électrique sur tous les contrôles possibles. Dans cette matrice, les lignes représentent le temps n et les colonnes représentent les différentes valeurs de volume possibles de la retenue. Le coefficient de V , situé à la ligne i et à la colonne j , représente la valeur maximale de production électrique entre le temps i et le temps final N pour une même valeur de volume $Volume(j)$.

La matrice U contient les valeurs des contrôles optimaux à appliquer pour optimiser la production électrique, c'est à dire les quantités d'eaux à libérer dans les conduites forcées à l'instant n pour un certain volume x présent dans la retenue du barrage au même instant. Les lignes discrétisent également le temps n , et les colonnes les différentes valeurs de volume possibles pour la retenue.

Le vecteur $Volume$ contient, dans l'ordre croissant, les valeurs discrétisées de tous les différents volumes d'eaux possiblement présents dans la retenue du barrage à un instant n .

Enfin, le vecteur $Controle$ contient, quant à lui, toujours dans l'ordre croissant, les différentes valeurs discrétisées de tous les contrôles physiquement applicables, au volume présent dans la retenue du barrage à un instant n .

Étant donné que les matrices Matlab utilisent des entiers naturels pour indexer leurs coefficients, on discrétise également les valeurs des volumes et des contrôles possibles sur des entiers naturels.

Dans la troisième partie du code, on construit les matrices U et V par remontée, à l'aide de l'algorithme de programmation dynamique.

Une première boucle for, dont l'ordre d'exécution est inversé permet de remonter dans le temps, instant par instant.

Une seconde boucle for, imbriquée dans la première et dont l'ordre d'exécution est direct, permet d'itérer sur toutes les valeurs de volumes possibles.

Ensuite, on crée un vecteur *possible*, qui va contenir les valeurs de production en un même volume, pour toutes les valeurs de contrôles physiquement applicables à ce volume.

Puis, dans une troisième boucle for, imbriquée dans la seconde boucle et dont l'ordre d'itération est également direct, on itère sur toutes les valeurs de ces contrôles applicables.

On teste avec une structure conditionnelle *if* si le contrôle itéré est physiquement applicable (retenue non vide ou suffisamment remplie).

Si c'est le cas, on calcule la valeur de production pour ce volume, pour ce contrôle et en cet instant et on stocke cette valeur dans le vecteur *possible*. Sinon, on laisse 0 car le contrôle n'est pas physiquement possible, ainsi la valeur de production que l'on obtiendrait par la formule de l'équation de programmation dynamique n'est pas celle du maximum que l'on recherche.

Une fois que la troisième boucle for a fini toutes ses itérations, on récupère le max des valeurs contenues dans le vecteur *possible*, ainsi que le contrôle correspondant, que l'on range respectivement dans les matrices V et U . Puis on passe au tour suivant de la seconde boucle for et ainsi de suite.

Les matrices V et U sont donc remplies en allant de gauche à droite et du bas vers le haut. On doit procéder dans ce sens là pour les construire car l'équation de programmation dynamique (présentée dans l'introduction) intervenant dans l'algorithme fait des appels récursifs aux coefficients de V situés sur la ligne inférieure de la matrice, d'où la remontée.

Dans la quatrième partie du code, on réutilise les matrices U et V nouvellement obtenues pour construire le vecteur des valeurs, des contrôles optimaux, ainsi que celui des valeurs, du volume de la retenue du barrage. Tout cela à partir de la valeur donnée en paramètre vol_depart , qui représente le volume d'eau contenu dans la retenue au début de la période de production N .

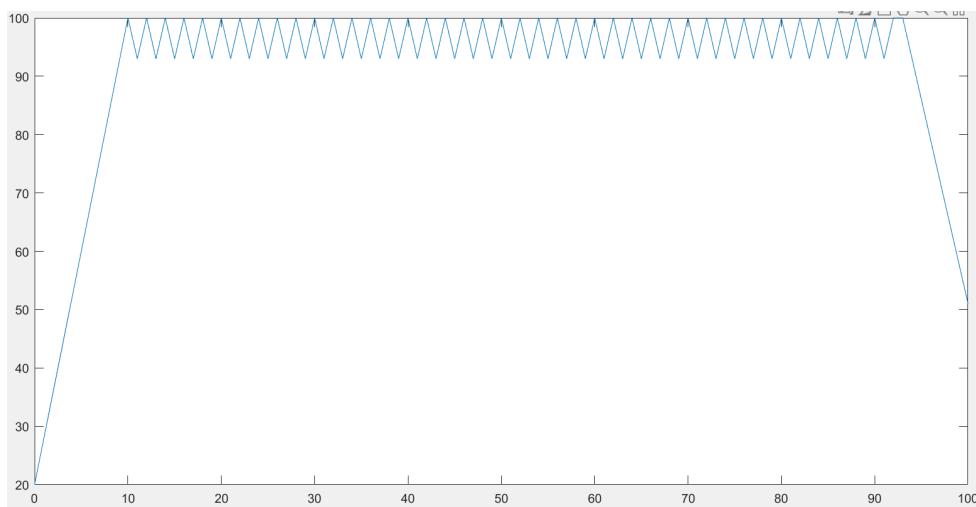
Enfin, on termine en récupérant la suite des contrôles $Controle_opt$, à appliquer pour optimiser la production électrique et en traçant le graphe du volume de la retenue du barrage en fonction du temps n , grâce à la commande *plot* et au vecteur *vol_courant*.

3.1.2 Quelques exemples de tests du code

On effectue quelques simulations avec différentes données en paramètres et on observe principalement le graphe du volume de la retenue, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélités à la réalité physique.

On commence par ce premier test :

```
[production,controles]=Optimise(100,15,100,8,20)
```



On observe tout d'abord que le niveau d'eau ne déborde jamais de la retenue. Le code est pourtant construit de manière à ce que cela puisse arriver. Mais on observe cela, que dans le cas où l'entrée d'eau journalière W est supérieur à la valeur maximale des contrôles applicables, en une seule fois. Car dans ce cas là, la retenue se remplit plus vite, qu'elle ne peut physiquement se vider, donc elle déborde.

Ici, l'entrée d'eau journalière W est de 8 et le contrôle maximal applicable, en une seule fois au volume de la retenue est 15, donc la retenue ne déborde pas.

Ce choix de l'algorithme s'explique par le fait que, si l'eau déborde, elle est perdue et ne passera donc pas par les turbines, ce qui représente une perte de gain potentiel.

On ne peut logiquement pas atteindre le maximum de production, en perdant de l'eau, alors qu'il existe des combinaisons de contrôles applicables qui peuvent permettre de faire passer toute l'eau par les turbines.

On observe également que l'algorithme fait en sorte de toujours maintenir le volume de la retenue à un niveau de remplissage, aussi grand que possible, et cela sur toute la durée de la période de production N . Il commence d'ailleurs par remplir entièrement la retenue, avant de commencer à appliquer des contrôles non nuls.

Ceci s'explique, par le fait que la hauteur d'eau h , présente dans la retenue du barrage, est en facteur

dans l'expression de la fonction de production instantanée, comme expliqué dans la partie 1.2.1, de ce rapport. En effet, plus la hauteur est grande et plus la pression exercée sur les entrées des conduites forcées sera importante.

On peut également voir que, puisque le volume d'eau du contrôle appliqué est aussi en facteur dans l'expression de la fonction de production instantanée (cf Partie 1.2.1), il faut quand même envoyer un débit important dans les conduites forcées pour produire le plus possible d'électricité. C'est ce qui est fait ici.

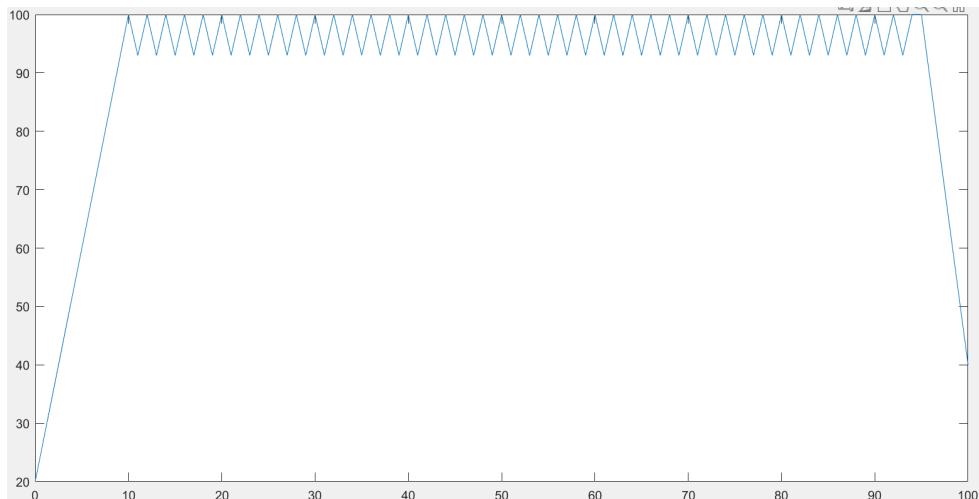
On remarque d'ailleurs que, lorsque l'on arrive vers la fin de la période de production N , la retenue se vide de manière très importante. La retenue ne se vide pas entièrement parce que la hauteur d'eau serait alors trop faible à la fin, et car il faudrait également arrêter les oscillations optimales du milieu un peu plus tôt, ce qui donnerait au final une production d'électricité plus faible.

L'algorithme trouve le juste équilibre entre vider beaucoup la retenue trop tôt et le faire trop tard, en s'adaptant à ce qui influe le plus sur la production instantanée.

L'algorithme cherche le meilleur équilibre entre produire beaucoup instantanément, tout en conservant un volume suffisant dans la retenue, tout au long de la période de production N . Tout ceci en prenant en compte l'entrée d'eau journalière W .

On effectue un nouveau test, en augmentant juste un peu le contrôle maximal applicable en une seule fois :

`[production,contrôles]=Optimise(100,20,100,8,20)`



On observe que l'algorithme effectue une simulation assez similaire à la précédente, il commence toujours par remplir la retenue au maximum. Puis, il produit les mêmes oscillations de production optimales, et cela en conservant exactement les mêmes valeurs de contrôles, alors qu'il pourrait pourtant appliquer des contrôles plus importants que dans le cas précédent.

Cela s'explique par l'importance du niveau de remplissage dans la production instantanée, car appliquer des contrôles plus importants viderait trop la retenue.

L'algorithme ne change pas sa suite de contrôle sur cette partie de la période de production, car c'est celle-ci qui permet de maximiser la production électrique et cela indépendamment de la fin de la période de production.

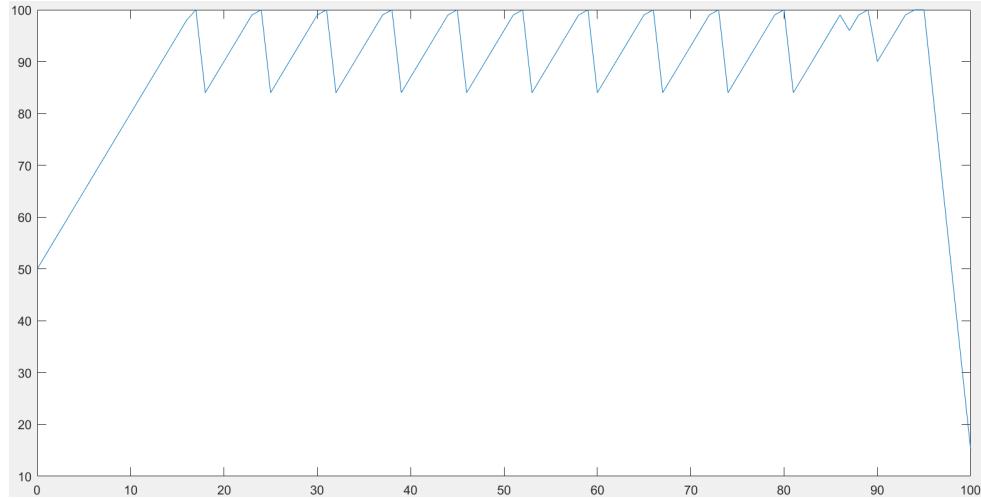
On peut quand même observer que, puisque la valeur maximale du contrôle applicable en une seule fois est passée à 20, l'algorithme peut donc vider la retenue plus rapidement. C'est pourquoi, dans cette simulation, il la vide un peu plus tard que dans l'autre. Il effectue des contrôles plus importants en moins de temps, donc il produit plus instantanément sur la fin et, en plus, il peut faire durer les oscillations de production optimale plus longtemps.

On observe que l'algorithme trouve bien le meilleur équilibre discrétréisé possible, entre tous les paramètres du problème qui interviennent dans les choix de contrôles, pour l'optimisation de la production.

On remarque qu'il s'adapte légèrement lorsqu'un paramètre varie aussi légèrement, ce qui suppose une certaine stabilité de la solution.

On effectue un nouveau test, cette fois-ci en diminuant la valeur de l'entrée d'eau journalière :

`[production,contrôles]=Optimise(100,20,100,3,50)`



On augmente alors un peu le volume de départ dans la retenue, pour que la simulation soit intéressante à observer, car le remplissage de la retenue au début de la période de production N est alors moins rapide.

On observe que l'algorithme remplit bien la retenue au maximum, avant d'appliquer des contrôles non nuls et qu'à la fin de la période de production N , il vide également la retenue au même moment que dans la simulation précédente.

En revanche, on observe cette fois que les oscillations optimales du milieu de la période de production ont une amplitude plus importante qu'avant, pour un nombre de répétitions de la période plus faible. Les contrôles appliqués en une seule fois sont plus importants, sans pour autant être maximaux.

Ceci s'explique par le fait que l'entrée d'eau journalière étant plus faible qu'avant, la retenue se re-remplit moins rapidement.

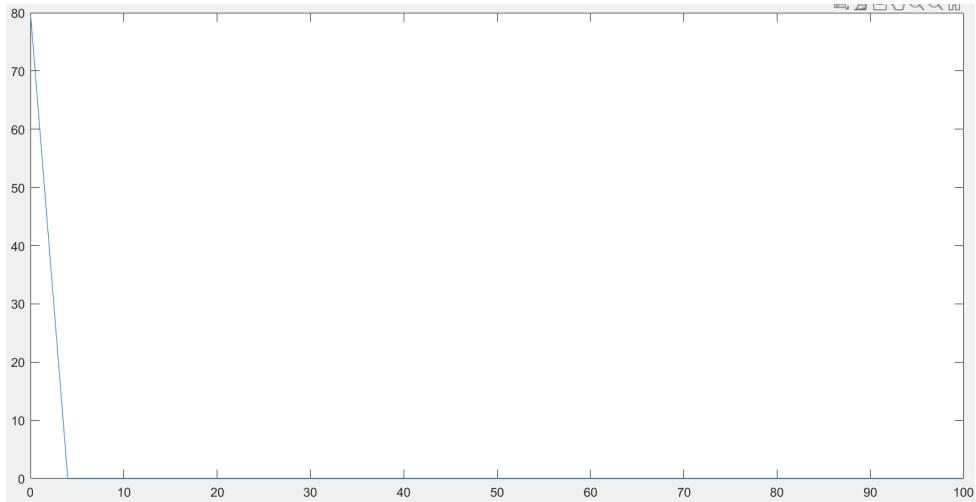
L'équilibre du maximum de production, trouvé par l'algorithme, tend alors plus à favoriser des contrôles assez importants avec une retenue pleine et à en faire moins souvent plutôt qu'à faire plus de contrôles avec une retenue pleine, mais pour des volumes plus faibles dans les conduites forcées.

Pour cette première résolution, les simulations donnent des schémas de contrôles assez répétitifs et assez similaires, suivant les paramètres de la situation car l'entrée d'eau est constante.

L'algorithme n'a donc pas à beaucoup adapter ses contrôles en fonction de W , il doit principalement se préoccuper de la hauteur d'eau dans la retenue, et de l'importance des contrôles envoyé, dans les conduites forcées en fonction de la vitesse de remplissage.

Pour cet avant-dernier test, on regarde ce que choisit de faire l'algorithme lorsque l'entrée d'eau journalière W est nulle :

`[production,contrôles]=Optimise(100,20,100,0,80)`

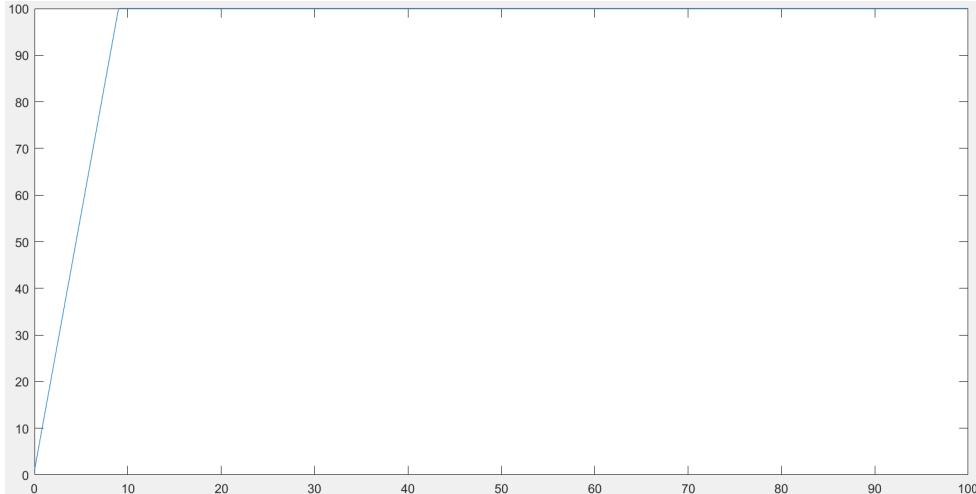


On observe que l'algorithme choisit de vider immédiatement la retenue, et cela le plus rapidement possible, avec des contrôles maximums parmi tous les contrôles applicables.

Cela est parfaitement logique, il n'y aura pas d'entrée d'eau sur toute la durée de la production, donc la meilleure chose à faire est de tout vider d'un coup, pour au moins maximiser la production instantanée, qui dépend beaucoup du volume instantanément envoyé dans les conduites forcées. (cf Partie 1.2.1)

Pour ce dernier test, voici un exemple lorsque l'entrée d'eau W , est plus importante, que la valeur maximale de contrôle applicable o :

`[production,controles]=Optimise(100,10,100,11,1)`



Comme attendu, la retenue se remplit jusqu'à déborder.

On observe quand même que l'algorithme l'aide au début, en ne la vidant pas avant son remplissage total. Ensuite, les contrôles appliqués sont tous égaux à leur valeur maximale applicable o car l'algorithme n'a pas de meilleur choix possible :

```
controles =
Columns 1 through 23
    0    0    0    0    0    0    0    0    0    10   10   10   10   10   10   10   10   10   10   10   10   10   10   10
Columns 24 through 46
    10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10
Columns 47 through 69
    10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10
Columns 70 through 92
    10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10   10
Columns 93 through 100
    10   10   10   10   10   10   10   10
```

3.2 Résolution avec entrée d'eau périodique

Pour cette nouvelle résolution, on simule l'entrée d'eau journalière W par une fonction périodique, sur toute la durée de production N .

3.2.1 Présentation du code et de la résolution numérique en Matlab

Pour commencer, voici les paramètres de notre problème pour cette nouvelle situation :

```
% L = taille maximale de la retenue du barrage.
% o = taille maximale de l'ouverture des conduites forcées.
% N = durée de la période de production d'électricité.
% vol_depart = valeur arbitraire du volume présent dans le barrage au
% départ, pour pouvoir construire le graphe.
```

On a plus besoin de donner une entrée d'eau journalière W , car elle n'est ici plus constante.

Voici le code, qui effectue la résolution numérique, de la nouvelle situation et qui renvoie également, tous les résultats voulus et qui tracent les graphes correspondants :

```
function [prod_max,programme,U]=Optimise(L,o,N,vol_depart)
    % Données
    rho=1000;
    g=9.80665;
    mu=0.75;
    % Définition des matrices
    V=zeros(N+1,L+1);           % Matrices des valeurs de production d'électricité entre les instants n et N.
    U=zeros(N+1,L+1);           % Matrices des valeurs des contrôles, liées aux valeurs de production, coefficient par coefficient.
    Volume=[0:L];
    % Vecteur des valeurs des niveaux de remplissages du barrage.
    Controle=[0:o];
    % Vecteur des tailles d'ouvertures de la conduite forcée.
    % Construction du vecteur des entrées d'eau:
    W=[];
    % Vecteur des entrées journalières.
    for i=0:N
        W(i+1)=(L/8)+(L/8)*cos((N/2)*i);
    end
    % Construction des Matrices U et V, par remontée.
    for i=N:-1:1
        % Boucle pour le temps
        for j=1:L+1
            % Boucle pour le volume
            possible=zeros(1,o+1);
            for k=1:o+1
                % Boucle pour les contrôles possibles.
                if(Volume(j)+W(i)-Controle(k)>=0) % Pour être sûr que le contrôle est physiquement possible, il reste de l'eau dans la retenue
                    possible(1,k) = Volume(j)*rho*g*mu*Controle(k) + V(i+1,round(max(min(L,Volume(j)+W(i)-Controle(k)),1)));
                end
            end
            [V(i,j),indice]=max(possible);
            U(i,j)=Controle(indice); % Décalage, pour avoir le contrôle u=0, à cause de l'indexation des tableaux dans Matlab.
        end
    end
    [prod_max,rang]=max(V(1,:));
    % Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX Volume_opt, grâce aux matrices U et V.
    Controle_opt=zeros(1,N+1);
    Controle_opt(1)=U(1,vol_depart);
    vol_courant=zeros(1,N+1);
    vol_courant(1)=vol_depart;
    for i=2:N+1
        vol_courant(i)=max(min(L,vol_courant(i-1)+W(i)-Controle_opt(i-1)),0);
        Controle_opt(i)=U(i,round(vol_courant(i))+1);
    end
    % Récupération des résultats
    programme=Controle_opt(1:N);
    % Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
    x=[0:N];
    % Vecteur abscisses du temps.
    clf
    hold on
    subplot(2,1,1);
    plot(x,W,'blue')          % Groupe de la sinusoïde.
    hold on
    plot(x,vol_courant,'green') % Groupe du volume d'eau, du barrage.
    hold off
    title('Volume et entrée d'eau, en fonction du temps')
    subplot(2,1,2);
    plot(x,Controle_opt,'red')
    title('Contrôles appliqués, en fonction du temps')
    hold off
end
```

La fonction Matlab *Optimise* de cette résolution est essentiellement identique à celle de la première résolution, pour le cas d'une entrée d'eau constante.

Cependant, elle en diffère quand même, en deux points principaux. Le premier point est que l'on ne prend plus l'entrée d'eau journalière W , en paramètre de notre fonction Matlab *Optimise*. À la place, on construit un vecteur W qui va contenir la suite de toutes les entrées d'eaux en fonction du temps. On le construit rapidement, juste avant d'utiliser l'algorithme de programmation dynamique, car on a besoin des différentes valeurs des entrées d'eau afin de pouvoir l'appliquer. On calcule toutes les valeurs que vont prendre ces entrées d'eau, à l'aide d'une boucle for itérant sur le temps n , ainsi qu'en utilisant l'expression d'une fonction périodique dont on pourra adapter la durée de la période ainsi que l'amplitude des oscillations.

Voici l'expression analytique du type de fonction périodique utilisée :

$$W(n) = A + B \cos(Cn)$$

La constante A permet de régler la hauteur des oscillations.

La constante B permet de régler l'amplitude des oscillations.

La constante C permet de régler la durée de la période d'une oscillation, donc elle permet d'en contrôler le nombre sur toute la durée de la production d'électricité N .

Le choix d'une fonction trigonométrique est surtout porté par le fait que l'on connaît très bien leur variation et qu'elles sont aussi très simples à modifier, pour obtenir les oscillations que l'on veut.

Le second point est qu'en utilisant la commande Matlab *subplot*, on a rajouté le graphe des contrôles que l'algorithme a choisi d'appliquer au volume de la retenue en fonction du temps.

Cela permet de comprendre plus facilement la simulation et donc de mieux en interpréter la pertinence par rapport à la réalité.

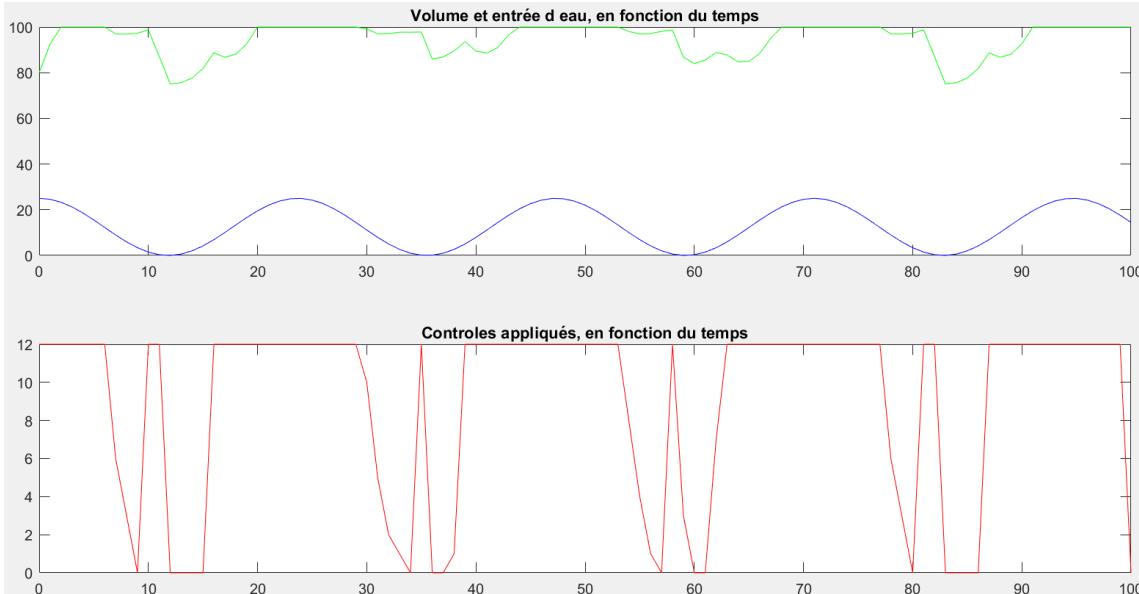
Dans ce nouveau cas, l'algorithme va devoir adapter ses contrôles face aux oscillations des entrées d'eaux, tout en continuant de chercher l'équilibre optimal entre une retenue pleine et un débit important envoyé au conduites forcées. Tout cela en essayant d'éviter au maximum le débordement de la retenue.

3.2.2 Quelques exemples de tests du code

On effectue quelques simulations, avec différentes données en paramètres et on observe principalement le graphe du volume de la retenue, ainsi que celui des contrôles appliqués, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité à la réalité physique.

On commence par ce premier test :

`[production,contrroles,U]=Optimise(100,12,100,80)`



La fonction périodique est ici donnée par :

$$W(i+1) = (L/8) + (L/8) * \cos((N/2)*i);$$

Pour commencer, on observe que le volume et les contrôles semblent suivre une variation périodique, dont la période est la même que celle de l'entrée d'eau.

On observe des contrôles très importants, un peu avant et au moment où les entrées d'eau sont les plus fortes et des contrôles presque nuls lorsque les entrées d'eaux sont les plus faibles.

On observe également des pics ponctuels de contrôles très importants, lorsque l'entrée d'eau est faible, mais que le volume de la retenue s'apprête à déborder.

On remarque que, lorsque l'entrée d'eau est maximale sur sa période, la retenue déborde et l'algorithme ne la vide pas plus au préalable, avant ce moment là. Cela montre bien que l'on est toujours sur cet équilibre entre maximiser la hauteur d'eau de la retenue, maximiser les contrôles appliqués au volume de cette retenue et minimiser les pertes d'eau lors des débordements.

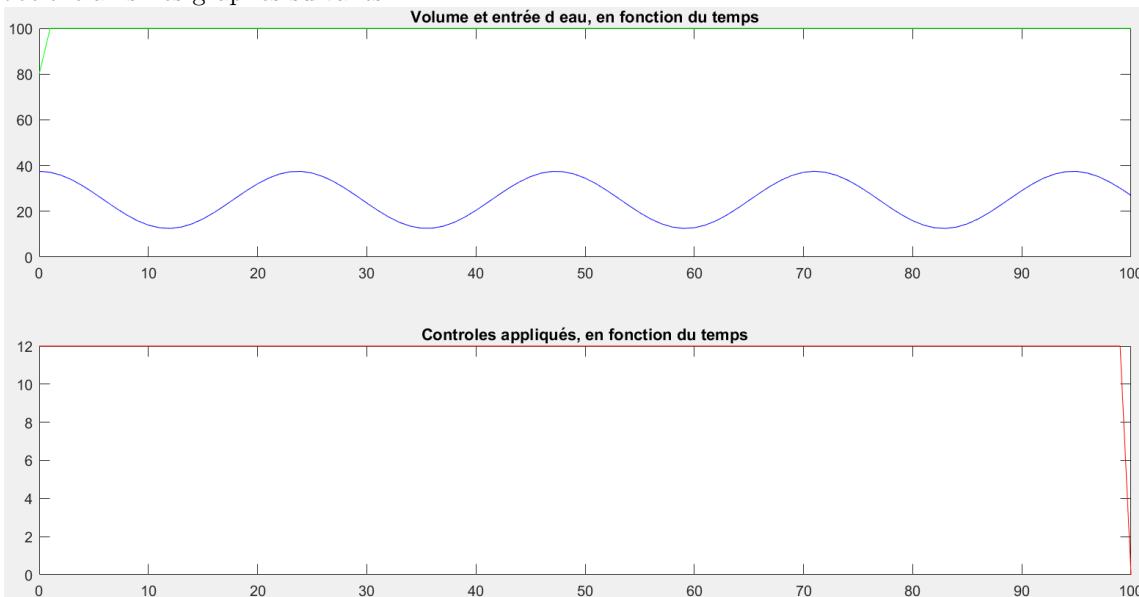
On effectue un nouveau test, avec les mêmes paramètres qu'au premier :

$$[production, controles, U] = Optimise(100, 12, 100, 80)$$

On modifie seulement la hauteur des oscillations à travers l'expression analytique de la fonction périodique de W , comme ceci :

$$W(i+1) = (L/4) + (L/8) * \cos((N/2)*i);$$

On obtient ainsi les graphes suivants :



L'entrée d'eau est alors trop forte, les contrôles sont tous maximaux et la retenue déborde sur toute la durée de la période de production d'électricité N .

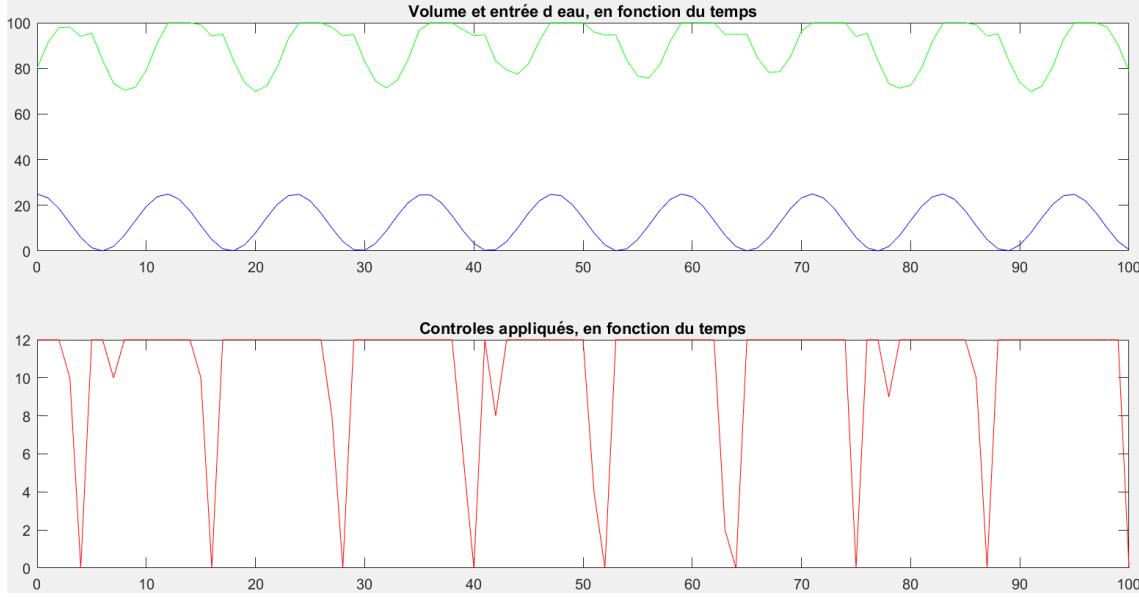
On effectue encore un nouveau test, avec les mêmes paramètres qu'au premier :

$$[production, controles, U] = Optimise(100, 12, 100, 80)$$

Cette fois-ci, on modifie seulement la période des oscillations à travers l'expression analytique de la fonction périodique de W , comme ceci :

$$W(i+1) = (L/8) + (L/8) * \cos((N/1)*i);$$

On obtient ainsi les graphes suivants :



On observe que les oscillations du volume et des contrôles appliqués sont beaucoup plus régulières, elle s'adaptent au nouveau rythme d'oscillations de l'entrée d'eau, sur toute la durée de la période de production d'électricité N .

L'algorithme commence toujours à vider la retenue un peu avant la vague et cesse toujours dès que l'entrée d'eau journalière commence à diminuer.

Comme avant-dernier test, on regarde ce que l'on obtient, lorsque l'on augmente la valeur maximale o des contrôles applicables au volume x présent dans la retenue.

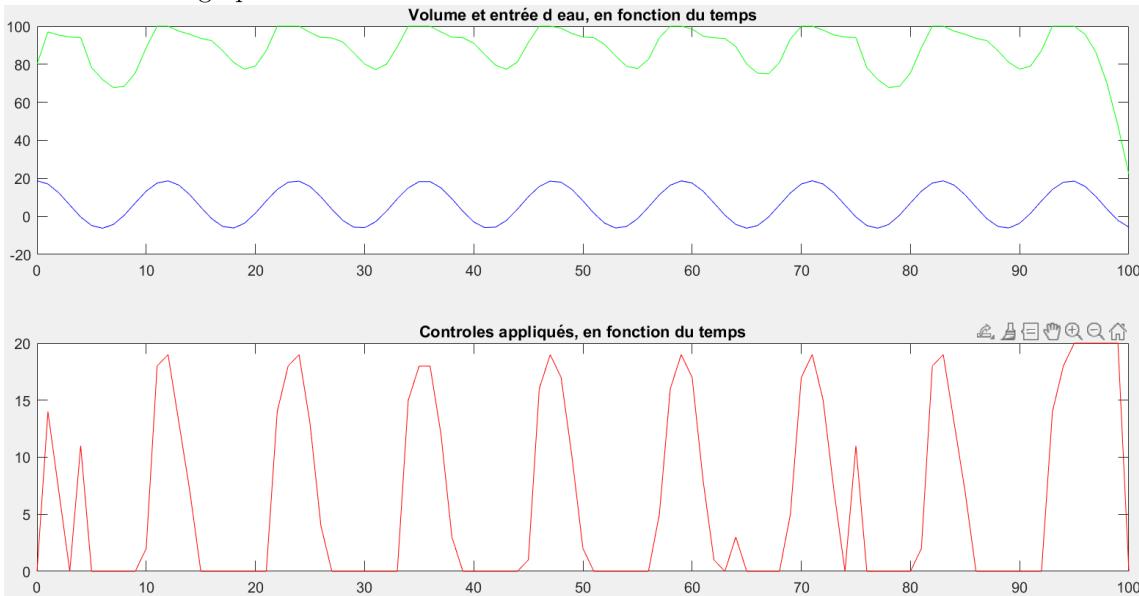
On effectue la simulation pour les paramètres suivants :

`[production,contrroles,U]=Optimise(100,20,100,80)`

On utilise la même fonction périodique W , pour l'entrée d'eau :

`W(i+1)=(L/8)+(L/8)*cos((N/1)*i);`

On obtient alors ces graphes :



On observe que l'amplitude des oscillations du volume présent dans la retenue a diminué. Leur période n'a pas changé, en revanche elle s'est décalée. Elle est plus proche de la période de l'entrée d'eau W .

Ceci s'explique par le fait que la capacité de contrôle ponctuel, étant plus importante, l'algorithme a moins besoin de vider la retenue avant la vague.

On observe d'ailleurs également que la retenue ne déborde presque plus car l'entrée d'eau la plus forte n'est plus suffisamment importante par rapport à la valeur maximale o des contrôles applicables au volume de la retenue.

Pour finir, on remarque également que la valeur maximale des contrôles applicables au volume d'eau de la retenue n'est presque jamais atteinte, à part à la fin de la période de production d'électricité, ce qui est parfaitement normal car on cherche alors à vider la retenue.

Cela s'interprète par le fait que l'algorithme de programmation dynamique privilégie la minimisation des pertes d'eau et la maximisation de la hauteur d'eau, dans la retenue, plutôt que la maximisation des contrôles envoyés aux conduites forcées pour maximiser la production électrique.

Comme dernier test, on vérifie que l'on obtient bien les mêmes résultats qu'avec le code de la partie 3.1 de ce rapport pour une entrée d'eau constante.

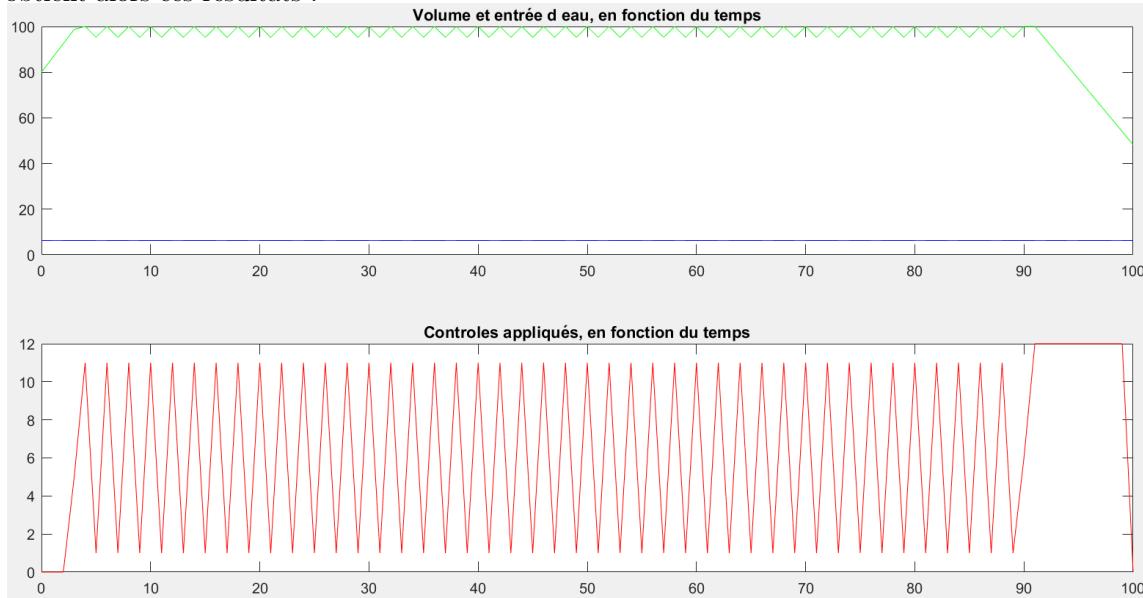
On utilise les paramètres suivants :

```
[production, controles, U]=Optimise(100, 12, 100, 80)
```

On utilise la fonction périodique W suivante :

$$W(i+1) = (L/16) + (0/8) * \cos((N/1)*i);$$

On obtient alors ces résultats :



La simulation que l'on observe est bien conforme à une résolution obtenue par le code de la partie 3.1, ce qui est normal puisqu'on utilise exactement le même algorithme de programmation dynamique et la même méthode de construction des graphes de la simulation.

3.3 Résolution avec une crue dans l'entrée d'eau

Pour cette nouvelle résolution, on simule maintenant l'entrée d'eau journalière W par une fonction gaussienne sur toute la durée de production N .

3.3.1 Présentation du code et de la résolution numérique en Matlab

Pour commencer, voici les paramètres de notre problème pour cette nouvelle situation :

```
% L = taille maximale de la retenue du barrage.
% o = taille maximale de l'ouverture des conduites forcées.
% N = durée de la période de production d'électricité.
% vol_depart = valeur arbitraire du volume présent dans le barrage au
% départ, pour pouvoir construire le graphe.
```

Voici le code qui effectue la résolution numérique de la nouvelle situation et qui renvoie également tous les résultats voulus et qui tracent les graphes correspondants :

```
function [prod_max,programme]=Optimise(L,o,N,vol_depart)
% Données
rho=1000;
g=9.80665;
mu=0.75;
% Définition des matrices
V=zeros(N+1,L+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1); % Matrices des valeurs des contrôles, liées aux valeurs de production, coefficient par coefficient.
Volume=[0:L]; % Vecteur des valeurs des niveaux de remplissages du barrage.
Controle=[0:O]; % Vecteur des tailles d'ouvertures de la conduite forcée.
% Construction du vecteur des entrées d'eau:
W=[]; % Vecteur des entrées journalières.
for i=0:N
    W(i+1)=5+49*exp(-(i-(N/2))^2/(N/4));
end
% Construction des Matrices U et V, par remontée.
for i=N:-1:1 % Boucle pour le temps
    for j=1:L+1 % Boucle pour le volume
        possible=zeros(1,o+1);
        for k=1:o+1 % Boucle pour les contrôles possibles.
            if(Volume(j)+W(i)-Controle(k)>=0) % Pour être sûr que le contrôle est physiquement possible, reste de l'eau dans la retenue.
                possible(1,k) = Volume(j)*rho*g*mu*Controle(k) + V(i+1,round(max(min(L,Volume(j)+W(i)-Controle(k)),1)));
            end
        end
        [V(i,j),indice]=max(possible);
        U(i,j)=Controle(indice); % Décalage, pour avoir le contrôle u=0, à cause de l'indexation des tableaux dans Matlab.
    end
end
[prod_max,rang]=max(V(1,:));
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX Volume_opt, grâce aux matrices U et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+W(i)-Controle_opt(i-1)),0);
    Controle_opt(i)=U(i,round(vol_courant(i))+1);
end
% Récupération des résultats
programme=Controle_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
% Test de la fonction crue:
x=[0:N];
clf
hold on
subplot(2,1,1);
plot(x,W,'blue') % Graphe de la sinusoïde.
hold on
plot(x,vol_courant,'green') % Graphe du volume d'eau, du barrage.
hold off
title('Volume et entrée d'eau, en fonction du temps')
subplot(2,1,2);
plot(x,Controle_opt,'red')
title('Contrôles appliqués, en fonction du temps')
hold off
end
```

Ce code est exactement le même que celui de la résolution précédente, le seul élément qui en diffère est l'expression analytique de la fonction, qui permet de calculer les valeurs du vecteur W des entrées d'eaux, en fonction du temps.

C'est ici une fonction gaussienne, dont la courbe est une cloche à asymptotes horizontales et dont l'expression analytique est de la forme suivante :

$$W(n) = A + B \exp\left(-\frac{(n - \mathcal{M})^2}{\sigma^2}\right)$$

La constante A permet de régler la hauteur de la crue.

La constante B permet de régler la hauteur de son point le plus haut.

La constante \mathcal{M} permet de régler le moment de la crue par rapport au temps, au cours de la période de production d'électricité N .

La constante σ^2 permet de régler la largeur de la crue, c'est à dire son importance du phénomène.

Le choix d'une fonction gaussienne s'explique surtout par le fait que, comme pour les fonctions trigonométriques, on connaît très bien ses variations et qu'elles sont aussi très simples à modifier, pour obtenir la crue que l'on veut.

On connaît bien ce genre de courbes car elles interviennent dans le cours de statistique inférentielle. En effet, les gaussiennes sont les densités de probabilités des variables aléatoires suivant une loi de probabilité dite "Normale" ou "Gaussienne" $\mathcal{N}(\mathcal{M}, \sigma^2)$.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mathcal{M})^2}{\sigma^2}\right)$$

La valeur \mathcal{M} représente la moyenne de la loi et la valeur σ^2 représente la variance de la loi. C'est la moyenne des carrés des écarts à la moyenne.

En statistiques, elle représente la diffusion quadratique des valeurs de la série autour de sa valeur moyenne. Donc, pour la courbe gaussienne, la variance représente la largeur de la cloche, plus la valeur de la variance est importante, plus la cloche de la gaussienne sera large.

À tout couple (\mathcal{M}, σ^2) , on peut associer une unique courbe gaussienne.

3.3.2 Quelques exemples de tests du code

On effectue quelques simulations avec différentes données en paramètres et on observe principalement le graphe du volume de la retenue, ainsi que celui des contrôles appliqués, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélités à la réalité physique.

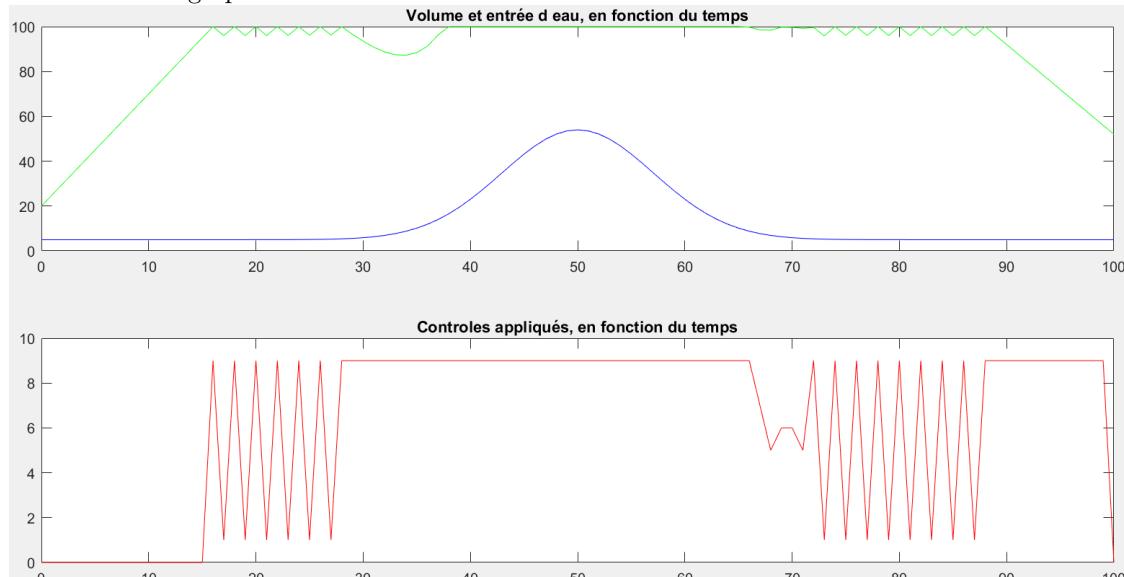
On commence par ce premier test :

```
[production,controles,U]=Optimise(100,9,100,20)
```

On utilise l'expression de la fonction gaussienne suivante :

```
W(i+1)=5+49*exp(-(i-(N/2))^2/(N/1));
```

On obtient alors les graphes suivants :



On observe que, dans la partie avant et après la crue, l'entrée d'eau est très proche d'être constante, l'algorithme fait alors les mêmes choix que dans le code des résolutions précédentes.

Il commence par remplir entièrement la retenue, puis il réalise les petites oscillations optimales, avant la crue, et après la crue il poursuit les petites oscillations et termine la période de production d'électricité N en vidant le plus possible la retenue.

Maintenant, concernant le moment de la crue, on peut observer que dès le début de l'augmentation de l'entrée d'eau journalière, l'algorithme choisit de cesser les oscillations et d'appliquer en permanence des contrôles maximaux sur la retenue.

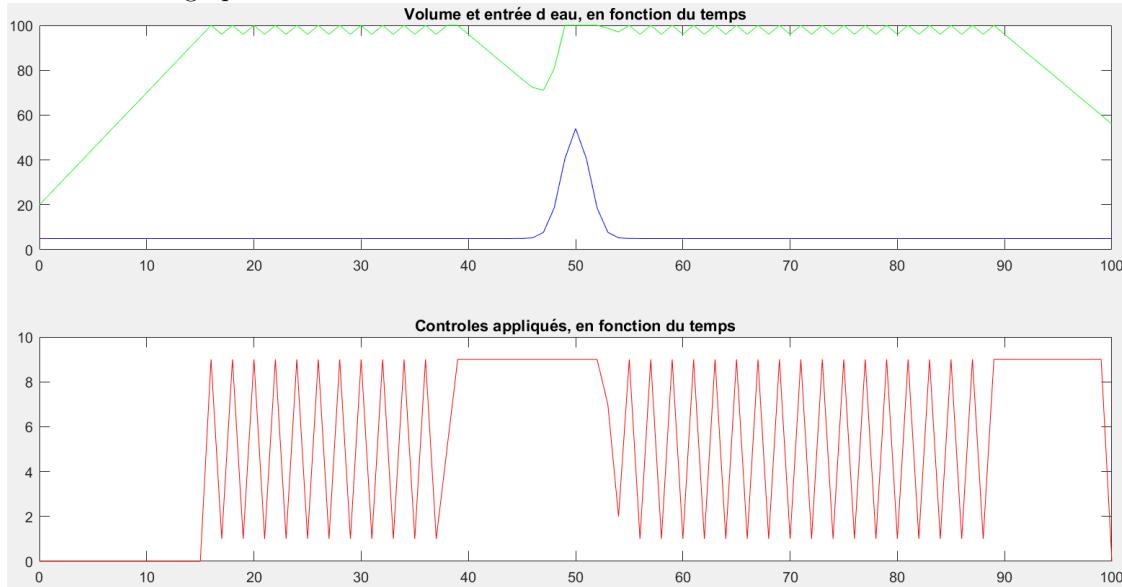
La retenue se vide un peu au début, puis l'importance maximale de la crue étant bien supérieure à la valeur maximale de contrôles applicables à la retenue, cette finit par se re-remplir et même par déborder.

On est ici encore sur un équilibre entre commencer à vider à fond assez tôt pour limiter les pertes de débordement de l'eau et ne pas vider non plus trop tôt, pour maintenir une hauteur d'eau suffisante dans la retenue, toujours dans le but de maximiser la production.

Avec les mêmes paramètres que pour le test précédent, on réalise un nouveau test, en diminuant la largeur de la gaussienne :

$$W(i+1) = 5 + 49 * \exp(-(i - (N/2))^2 / (N/32));$$

On obtient ainsi les graphes suivants :



On observe que, lorsque la largeur de la gaussienne est moins importante, le volume globale d'eau ajoutée par la crue, est diminué de beaucoup.

De plus, l'augmentation de l'entrée d'eau au moment de la crue est beaucoup moins progressive et beaucoup plus brutale, tout comme la redescente, lorsque le pic de la crue est passé.

On remarque que l'algorithme adapte sa résolution par rapport à cette autre crue, il commence à vider fortement la retenue bien avant le début de la crue et fait descendre la hauteur d'eau de la retenue bien plus bas qu'avant.

De même, la retenue se re-remplit encore très rapidement, étant donné que la hauteur maximale de la crue est toujours la même. Au plus fort de la crue, la retenue déborde encore d'ailleurs, mais beaucoup moins longtemps que dans le cas précédent.

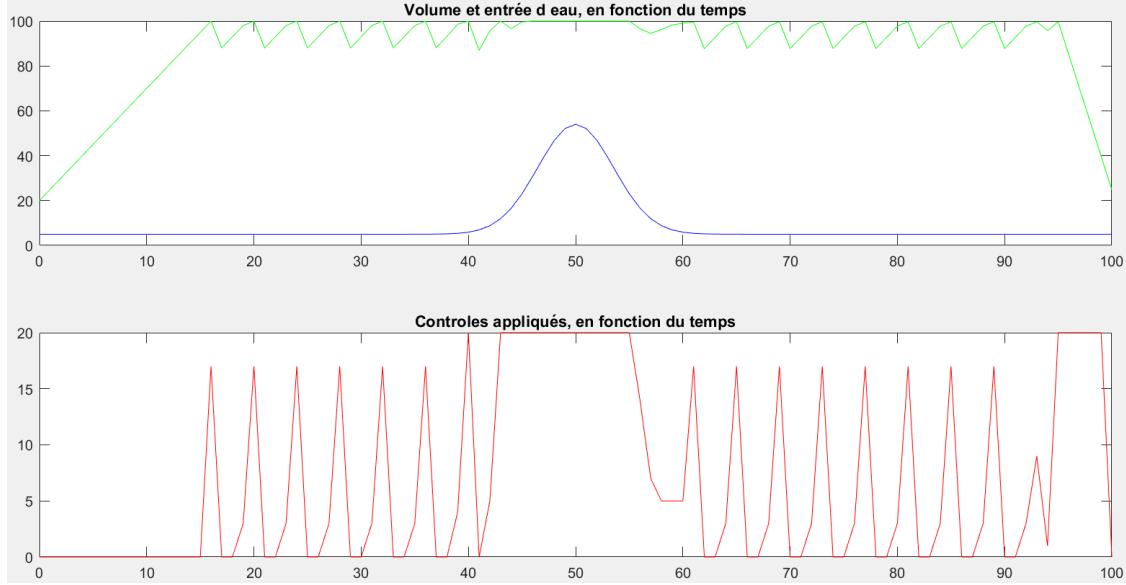
Au plus fort de la crue, l'équilibre d'optimalité trouvé par l'algorithme de programmation dynamique

tend à privilégier d'importants contrôles envoyés dans les conduites forcées, par rapport à une hauteur d'eau maximale, dans la retenue, puisque de toute façon, celle-ci se re-remplira et débordera rapidement.

On effectue un dernier test dans lequel on augmente seulement la valeur maximale o des contrôles applicables au volume d'eau présent dans la retenue du barrage :

```
[production, controles, U]=Optimise(100, 20, 100, 20)
```

On obtient alors ces graphes :



On observe que, comme dans le cas de l'entrée d'eau périodique, lorsque l'on augmente les capacités de contrôles sur le volume de la retenue, l'algorithme tend à diminuer le nombre d'oscillations sur la partie en dehors de la crue, de la période de production d'électricité N .

Cela s'explique par le fait que l'équilibre optimal tend alors à plus privilégier les contrôles importants, pour maximiser la production, car ils sont ici possibles dans cette situation.

Mais comme la vitesse de remplissage reste la même qu'avant, le rythme des oscillations est ralenti pour permettre à la retenue de se re-remplir et donc de maintenir une hauteur d'eau importante.

Enfin, la retenue déborde encore ici, mais cette fois, l'algorithme ne prend pas la peine de vider un peu la retenue avant, car le contrôle maximal est suffisant, pour limiter les pertes d'eau, au plus fort de la crue.

3.4 Résolution ajoutant la prise en compte des tarifs horaires de l'électricité

Pour cette nouvelle résolution, on simule l'entrée d'eau journalière W , au choix entre une fonction périodique et une fonction gaussienne, comme précédemment.

On ajoute maintenant la prise en compte par l'algorithme de programmation dynamique de l'évolution du tarif de l'électricité, en fonction du temps.

En effet, pour certaines heures de la journée, appelées "heures creuses", EDF vend son électricité jusqu'à environ 25% moins cher que le reste du temps, que l'on appelle les "heures pleines".

Les heures creuses se placent généralement entre 22h et 6h, du matin, mais cela peut évoluer en fonction du fournisseur d'énergie.

3.4.1 Présentation du code et de la résolution numérique en Matlab

Pour commencer, voici la liste des anciens et des nouveaux paramètres de notre problème :

```
% L = taille maximale de la retenue du barrage.
% o = taille maximale de l'ouverture des conduites forcées.
% N = durée de la période de production d'électricité.
% type_entree_deau = type d'entrée d'eau, au choix entre "périodique" et "crue".
% vol_depart = valeur arbitraire du volume présent dans le barrage au
% départ, pour pouvoir construire le graphe.
% P_H_Pleine = Prix de l'électricité, en heure pleine.
% P_H_Creuse = Prix de l'électricité, en heure creuse.
% Periode_H = Période entre les changements de tarifs, heures pleines et heures creuses.
```

Voici le nouveau code, qui effectue la résolution numérique de la situation, qui renvoie tous les résultats voulus et qui tracent les graphes correspondants :

```
function [prod_max,programme,U]=Optimise(L,o,N,type_entree_deau,vol_depart,P_H_Pleine,P_H_Creuse,Periode_H)
% Données
rho=1000;
g=9.80665;
mu=0.75;
% Définition des matrices
V=zeros(N+1,L+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1); % Matrices des valeurs des contrôles, liées aux valeurs de production.
Volume=[0:L];
% Vecteur des valeurs des niveaux de remplissages du barrage.
Controle=[0:0:]; % Vecteur des tailles d'ouvertures de la conduite forcée.
Production_elec_prix=ones(1,N+1); % Vecteur des valeurs de production d'électricité en prix, en fonction du temps.
Production_elec_prix_cumulee=ones(1,N+1); % Vecteur des valeurs des gains cumulés en prix, en fonction du temps.
% Construction du vecteur des entrées d'eau:
W=[]; % Vecteur des entrées d'eau.
if type_entree_deau=="periodique"
    for i=0:N
        W(i+1)=(L/32)+(L/64)*cos((N/2)*i);
    end
elseif type_entree_deau=="crue"
    for i=0:N
        W(i+1)=2+(N/4)*exp(-(i-(N/2))^2/(N/4));
    end
end
% Construction du vecteur des prix de l'électricité, en fonction du temps:
Prix=[]; % Vecteur des prix.
Compteur=1;
Changement=false;
```

```

for i=0:N
    if Compteur==Periode_H
        Compteur=0;
        Changement=not(Changement);
    end
    if Changement
        Prix(i+1)=P_H_Pleine;
    else
        Prix(i+1)=P_H_Creuse;
    end
    Compteur=Compteur+1;
end
% Construction des Matrices U et V, par remontée.
for i=N:-1:1 % Boucle pour le temps
    for j=1:L+1 % Boucle pour le volume
        possible=zeros(1,o+1);
        for k=1:+1 % Boucle pour les contrôles possibles.
            if(Volume(j)+W(i)-Controle(k)>=0) % Vérifie que le contrôle est physiquement possible, il reste de l'eau.
                possible(1,k) = Volume(j)*rho*g*mu*Controle(k)*Prix(i) + V(i+1,round(max(min(L,Volume(j)+W(i)-Controle(k)),1)));
            end
        end
        [V(i,j),indice]=max(possible);
        U(i,j)=Controle(indice); % Décallage, pour avoir le contrôle u=0, à cause de l'indexation des tableaux dans Matlab.
    end
end
[prod_max,rang]=max(V(1,:));
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX Volume_opt, grâce aux matrices U et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
Production_elec_prix(1)=vol_courant(1)*rho*g*mu*Controle_opt(1)*Prix(i);
Production_elec_prix_cumulee(1)=Production_elec_prix(1);
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+W(i)-Controle_opt(i-1)),0);
    Controle_opt(i)=U(i,round(vol_courant(i))+1);
    Production_elec_prix(i)=vol_courant(i)*rho*g*mu*Controle_opt(i)*Prix(i);
    for j=i:N+1
        Production_elec_prix_cumulee(j)=Production_elec_prix_cumulee(j)+Production_elec_prix(i);
    end
end
% Récupération des résultats
programme=Controle_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
% Test fonction sinusoïdale:
x=[0:N]; % Vecteur abscisses du temps.
clf
hold on
subplot(3,1,1);
plot(x,vol_courant,'red'); % Graphe du volume d'eau, du barrage.
title('Volume d'eau du barrage, en fonction du temps')
subplot(3,1,2);
plot(x,Prix,'blue');
title('Prix de l'électricité, en fonction du temps')
subplot(3,1,3);
plot(x,W,'green');
title('Entrée d'eau, en fonction du temps')
hold off
figure(2)
clf
hold on
subplot(3,1,1);
plot(x,Production_elec_prix_cumulee,'blue');
title('Production total cumulée, en fonction du temps')
subplot(3,1,2);
plot(x,Production_elec_prix,'magenta');
title('Production instantanée en prix, en fonction du temps')
subplot(3,1,3);
plot(x,Controle_opt,'green');
title('Contrôles appliqués, en fonction du temps')
hold off
end

```

Ce nouveau code est construit sur le modèle du précédent, mais il en diffère en quelques points.

Pour commencer, on a rajouté un paramètre nommé *type_entree_eau*, qui est une chaîne de caractère. Elle peut prendre les valeurs "periodique" ou "cruel". Elle permet de choisir le type d'entrée d'eau que l'on souhaite utiliser pour la simulation. On réutilise les fonctions *W*, des deux précédentes parties du projet (cf 3.2 et 3.3).

Dans le code, on utilise une structure conditionnelle *if*, dans laquelle le prédicat est un test qui vérifie

la valeur de la chaîne de caractère *type_entree_deau*, donnée en paramètre et qui lui associe la fonction correspondante pour la construction du vecteur W , toujours avec une boucle for itérant sur le temps n .

Ensuite, pour pouvoir ajouter la prise en compte du tarif horaire de l'électricité dans la résolution du problème, on a dû construire un vecteur nommé *Prix*, qui donne le prix de l'électricité, à chaque instant du temps discrétisé n .

On a choisi de modéliser la configuration des heures pleines et des heures creuses comme une boucle périodique. On se donne le prix en heures pleines *P_H_Pleine*, le prix en heures creuses *P_H_Creuse* et la durée pendant laquelle on reste sur le même tarif *Periode_H*.

On a choisi de mettre la même durée de temps discrétisé n pour la durée durant laquelle le tarif est celui des heures pleines et pour celle durant laquelle le tarif est celui des heures creuses. On a fait ce choix car c'est plus simple à construire, ça rend plus facile l'observation des graphes et ça permet d'avoir un paramètre de moins à donner en entrée de la fonction.

Pour construire le vecteur *Prix*, on utilise une variable nommée *Compteur*, initialisée à 0 et un booléen nommé *Changement*. Dans une boucle for, itérant sur le temps n , on teste dans une première structure conditionnelle *if*, si la valeur de la période *Periode_H*, donnée en paramètre de la fonction est atteinte. Si c'est le cas, on change la valeur du booléen *Changement* par sa négation et on remet le compteur à 0. Si ce n'est pas le cas, on ne fait rien.

Ensuite, à l'aide d'une seconde structure conditionnelle *if*, on affecte au vecteur *Prix* une certaine valeur de prix horaire au temps n , donnée par la boucle for, suivant la valeur du booléen *changement*. Une fois ceci fait, on incrémente le *compteur* de 1 et on passe au tour de boucle suivant.

Si on voulait maintenant construire le vecteur *Prix* avec deux durées différentes pour les deux tarifs horaires, on aurait juste à modifier le premier *if*, pour qu'il vérifie la valeur du compteur et qu'il vérifie que l'on est bien actuellement sur l'autre période de prix horaires, avec la valeur du booléen *changement*. Ceci afin de savoir si on doit passer à l'autre prix horaire, en changeant la valeur du booléen *changement*. Ensuite, on rajoute un *elseif* pour tester de la même manière, avec les deux conditions, sur le compteur et sur le booléen, s'il faut passer à l'autre prix horaire.

Une fois que l'on a notre vecteur *Prix*, on peut rajouter la valeur du Prix horaire de l'électricité en fonction du temps discrétisé n dans l'algorithme de programmation dynamique.

On effectue cela, en multipliant tout simplement le prix *Prix(i)* à la fonction production instantanée $L(X_n, U_n)$ (cf Partie 1.2.3).

À chaque instant n , pour chaque volume j , la recherche de la valeur maximale de production prend alors en compte le prix horaire de l'électricité. L'algorithme de programmation dynamique le fait donc aussi, en effectuant sa remontée. On en retrouve ainsi l'influence, dans les résolutions.

Enfin, dans ce nouveau code, on a également amélioré les graphes car on ajouté au graphe du volume de la retenue et à celui contrôles appliqués, celui du prix horaire de l'électricité, celui de l'entrée d'eau, ainsi que ceux des productions ponctuelles et cumulée, en prix et en fonction du temps.

On affiche les graphes sur deux fenêtres graphiques différentes, en colonne et les uns derrière les autres, à l'aide de la commande Matlab *subplot*, afin de pouvoir mieux les comparer, sur une même échelle de temps.

Concernant le reste du code, le fonctionnement est exactement le même que dans les résolutions des cas précédents.

3.4.2 Quelques exemples de tests du code

On effectue quelques simulations avec différentes données en paramètres et on observe principalement le graphe du volume de la retenue, ainsi que celui des contrôles appliqués en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélités à la réalité physique.

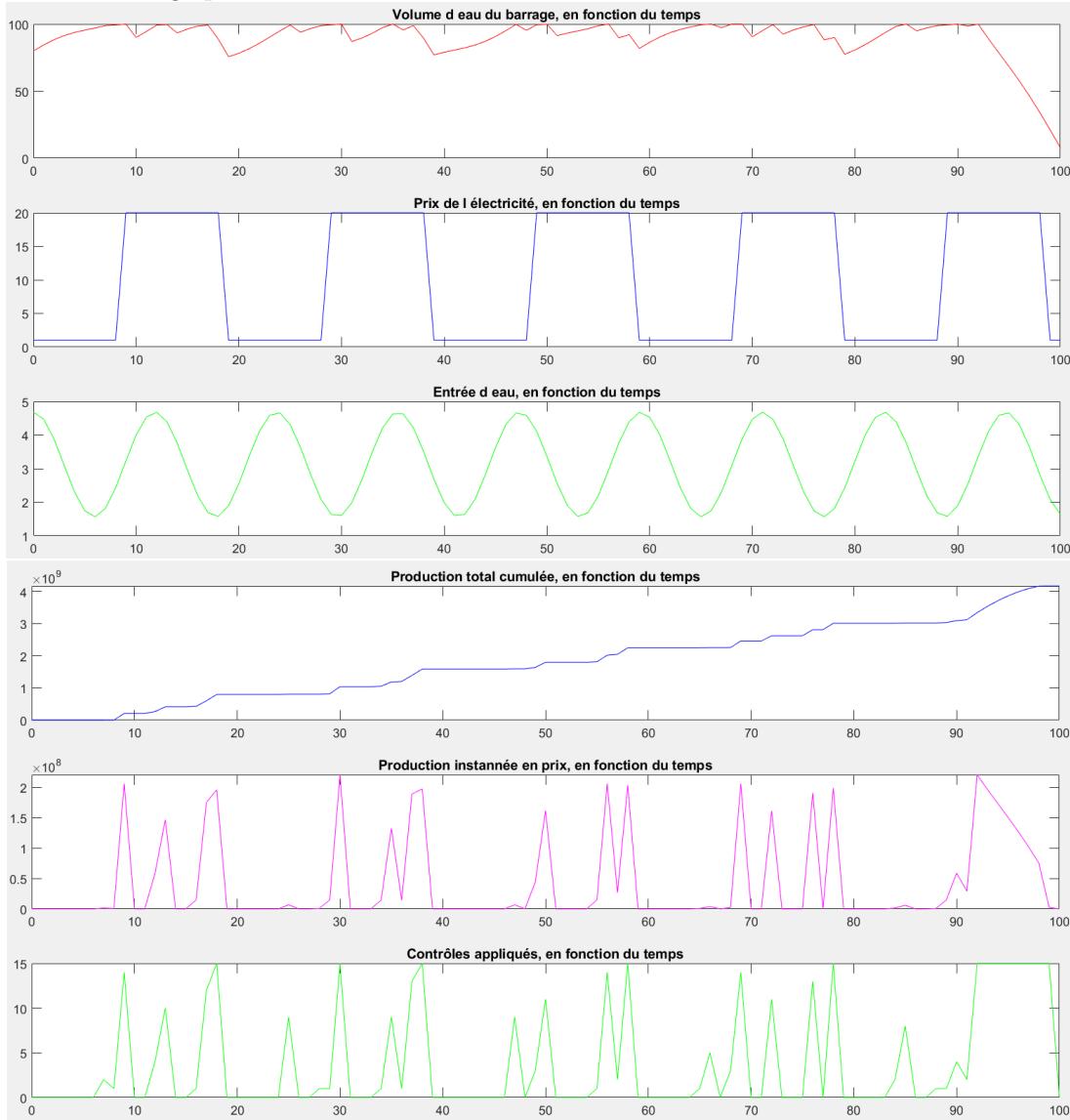
On commence par ce premier test :

```
[production, controles, U]=Optimise(100,15,100, "periodique", 80,20,1,10)
```

On utilise l'expression de la fonction périodique suivante :

$$W(i+1) = (L/32) + (L/64) * \cos((N/1)*i);$$

On obtient alors les graphes suivants :



Ici, l'équilibre du maximum de production, penchant entre maximiser la hauteur d'eau, présente dans la retenue et maximiser les contrôles appliqués, est influencée en même temps par deux facteurs différents. Ces deux facteurs sont l'entrée d'eau et le prix horaire de l'électricité.

Premièrement, on observe que la valeur maximale de contrôle applicable au volume de la retenue, est suffisamment importante par rapport au maximum des entrées d'eau pour que l'algorithme puisse trouver une suite de contrôles qui évite que la retenue ne déborde.

On observe en effet qu'à chaque fois que la retenue s'apprête à déborder, on a un pic de contrôle qui permet de réguler la hauteur d'eau dans la retenue.

Secondement, on observe également que la différence du prix horaire de l'électricité, entre les heures pleines et les heures creuses, a aussi une influence sur la résolution effectué par l'algorithme.

En effet, on observe dans un premier temps que l'algorithme de programmation dynamique laisse la retenue se remplir durant les plages horaires, où le tarif est celui des heures creuses, cela en évitant bien

sûr qu'elle ne déborde.

Puis, dans un second temps, on remarque également que l'algorithme vide systématiquement la retenue juste avant la fin de la plage horaire, où le tarif de l'électricité est celui des heures pleines.

On peut aussi voir que, comme précédemment, l'algorithme commence par laisser la retenue se remplir avant d'appliquer le moindre contrôle, sur son volume. On peut également voir qu'il vide toujours la retenue de manière très importante, un peu avant la fin de la période de production.

La résolution est ici influencée par ces deux facteurs en même temps, cela rend plus difficile l'interprétation des résultats et donc la vérification du bon fonctionnement de l'algorithme.

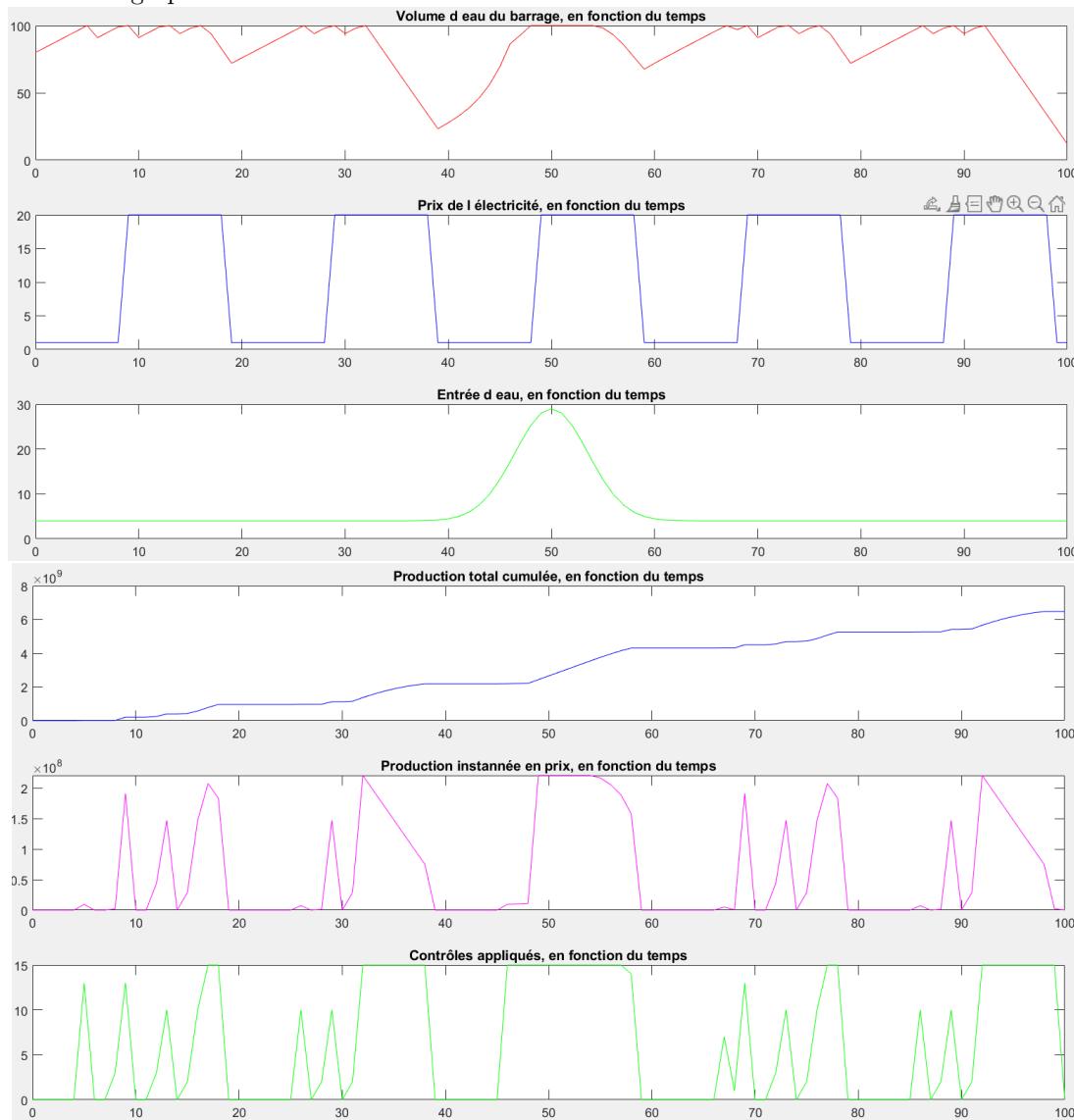
On effectue donc un autre test, avec les mêmes paramètres en entrée de la fonction, mais cette fois-ci, pour une entrée d'eau de type *crue* :

```
[production,contrroles,U]=Optimise(100,15,100,"crue",80,20,1,10)
```

On utilise l'expression de la fonction gaussienne suivante :

$$W(i+1)=4+(N/4)*\exp(-(i-(N/2))^2/(N/4));$$

On obtient les 6 graphes suivants :



Dans ce nouvelle exemple de test, on observe beaucoup mieux l'influence du prix horaire de l'électricité, sur l'optimisation de la production.

En effet, puisque avant et après la crue, l'entrée d'eau est quasiment constante, on voit beaucoup mieux que l'algorithme attend systématiquement la fin des périodes d'heures creuses pour vider un peu la retenue et laisse toujours la retenue se remplir en heures creuses, tout en évitant évidemment que celle-ci ne déborde.

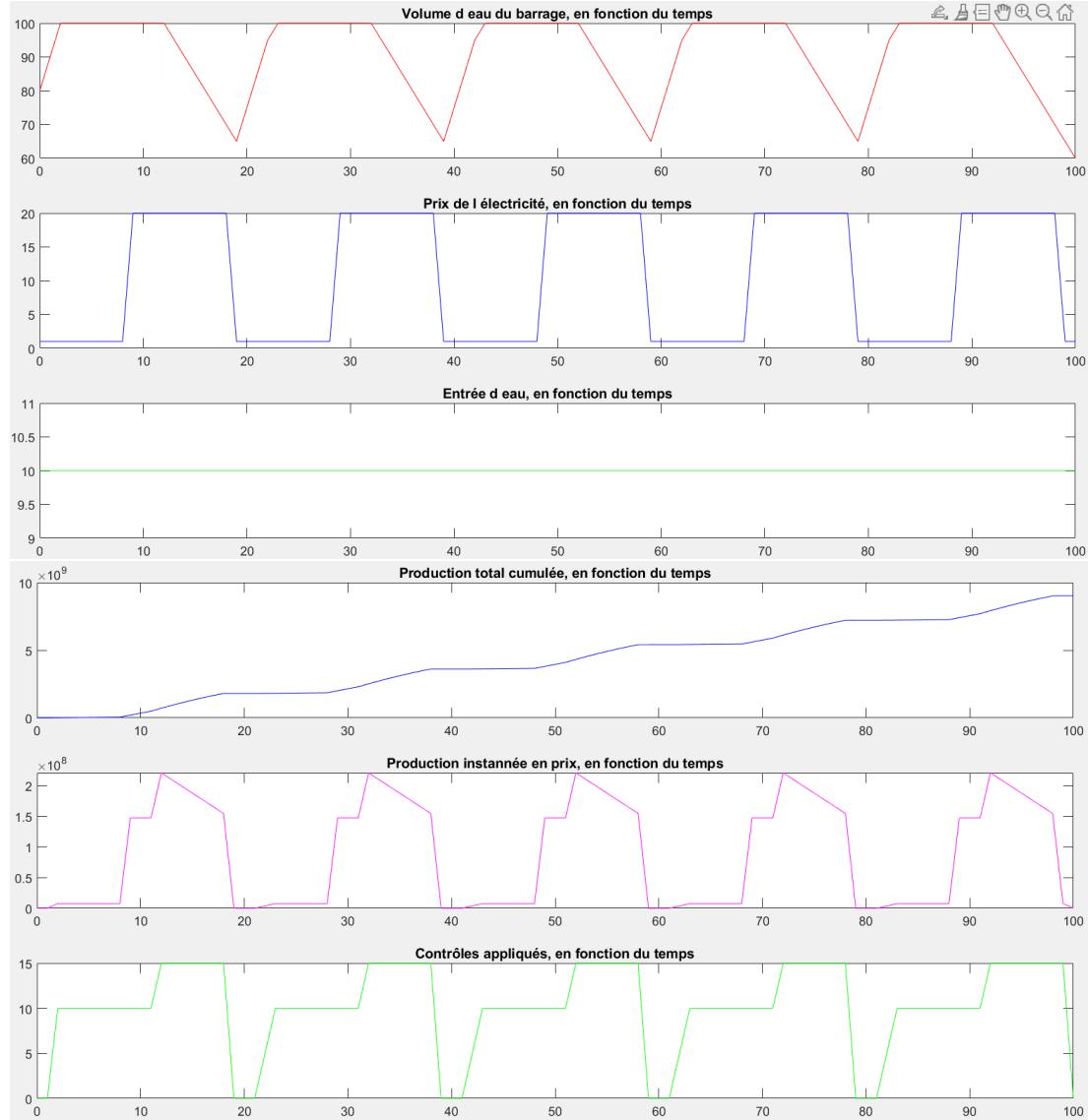
Maintenant, même concernant le moment de la crue, on peut observer que l'algorithme, décide de vider la retenue en prévention de l'importante entrée d'eau, largement bien avant la crue.

On observe d'ailleurs que ce moment là, correspond tout justement à la fin d'une période de tarif heures pleines.

On explique ce choix de l'algorithme par le fait que privilégier d'importants contrôles sur cette période là, ainsi qu'au plus fort de la crue, permet de produire davantage d'électricité, malgré une faible hauteur d'eau dans la retenue, pendant un cours moment, que de vider la retenue juste avant le début de la crue, en période de tarif heures creuses.

Pour confirmer la validité de nos observations des exemples précédents, on effectue maintenant un dernier test avec les mêmes paramètres que dans le cas précédent, mais pour une entrée constante, en fonction du temps.

On obtient alors les résultats suivants :



Pour le cas d'une entrée d'eau constante, on peut effectivement voir que les observations précédentes,

sont bien confirmées par les résultats de ce test.

On observe en effet les mêmes comportements de l'algorithme de programmation dynamique, mais cette fois-ci sans l'influence de l'entrée d'eau sur les contrôles.

On observe même l'apparition d'une périodicité dans les contrôles appliqués au volume de la retenue. Cette périodicité est parfaitement synchronisée à celle du tarif de l'électricité en heures pleines.

3.5 Résolution probabiliste, en générant une chaîne de Markov

Après avoir effectué une résolution déterministe de notre problème, on s'attaque maintenant au point de vue probabiliste.

On remplace pour cela l'entrée d'eau journalière W , auparavant calculée par une fonction périodique ou bien une fonction gaussienne, par une chaîne de Markov, que l'on générera aléatoirement et que l'on pourra adapter plus facilement en fonction des situations.

Afin de simuler l'entrée d'eau à l'aide d'un chaîne de Markov, on commence par voir comment en générer une aléatoirement.

3.5.1 Génération aléatoire d'une Chaîne de Markov

On commence par définir précisément ce qu'est une chaîne de Markov.^[4]

Définition Soient S un ensemble fini ou dénombrable et $(X_n)_{n \in \mathbb{N}}$ une suite de variables aléatoires définies, sur le même univers Ω , muni de la mesure de probabilité \mathbb{P} et à valeurs dans S . On dit que (X_n) est une chaîne de Markov homogène si :

- i) (Propriété de Markov) $\forall n \in \mathbb{N}$ et $\forall x_0, x_1, \dots, x_{n+1} \in S$

$$\mathbb{P}(X_{n+1} = x_{n+1} | X_{0:n} = x_{0:n}) = \mathbb{P}(X_{n+1} = x_{n+1} | X_n = x_n)$$

- ii) (Homogénéité) la probabilité de transition $\mathbb{P}(X_{n+1} = y | X_n = x)$ ne dépend pas de n , $\forall (x, y) \in S^2$.
On note $p(x, y)$, cette probabilité de transition.

La loi de X_0 est appelée loi initiale de la chaîne de Markov et on appelle matrice de transition la matrice définie par $P = p(x, y)_{x, y \in S}$. On remarque que la matrice P vérifie $p(x, y) \geq 0$ et $\sum_y p(x, y) = 1$. Une telle matrice est dite *stochastique*, c'est à dire que chacune de ses lignes est une mesure de probabilité.

Une chaîne de Markov est donc une suite de variables aléatoires $(X_n)_{n \in \mathbb{N}}$ qui décrivent à chaque instant n l'état d'un système aléatoire défini par récurrence.

Les mesures de probabilités de chacune des variables aléatoires X_i de la suite nous sont données uniquement à partir de la loi de X_0 et de la matrice de transition P . Pour obtenir ces mesures, on les multiplie entre elles en faisant des produits matriciels successifs, avec la matrice de transition P , multipliée par la droite au vecteur ligne de la loi initiale.

Maintenant, pour construire aléatoirement une chaîne de Markov, on procède différemment, car l'état du système au rang $n + 1$ ne doit dépendre que de celui dans lequel il est au rang n , afin que la chaîne vérifie la propriété de Markov.

Chaque ligne et chaque colonne de P représente un état du système, et les valeurs des coefficients d'une même ligne représentent les différentes probabilités de passer dans chacun des différents états du système au rang suivant de la suite, dans le cas où le système se trouve dans l'état correspondant à la ligne en question.

À partir de cette observation sur la matrice P , on peut construire facilement un court algorithme qui va générer une chaîne de Markov de manière aléatoire, à partir de la matrice P et de la donnée de l'état initial du système.

Le principe de cet algorithme est le suivant :

- On se donne un état initial x de notre système, qui sera le premier terme de la chaîne.
- On se donne également un nombre réel aléatoire a , compris entre 0 et 1 inclus.
- On regarde la ligne de la matrice de transition P , qui correspond à l'état x , de notre système.

- En se déplaçant de gauche à droite sur cette même ligne, on compare à chaque colonne, notre nombre aléatoire a , avec la somme de tous les coefficients de la ligne, compris entre la 1 ère colonne et la colonne, sur laquelle on est.
- Si a est strictement supérieur à cette somme, on passe à la colonne suivante et on ajoute à la somme, le nouveau coefficient de la ligne de la matrice, correspondant à cette colonne et on refait le test.
- Si a est inférieur ou égale à cette somme, on récupère l'indice de la colonne, sur laquelle on est et on le stocke, car c'est le terme suivant de notre chaîne, qui représente le nouvel état du système.
- On recommence alors le même procédé, avec un nouveau nombre aléatoire a , compris entre 0 et 1 inclus et avec la ligne de la matrice de transition P , correspondant au nouvel état du système.

On écrit maintenant cet algorithme sur Matlab.

Voici les 3 paramètres dont on a besoin pour générer une chaîne de Markov aléatoire :

```
n=longueur de la chaîne de markov.
nbr_detats=nombre d'états que le système peut atteindre.
etat_initial=état initial du système.
```

Voici le code, qui à partir de la donnée de ces 3 paramètres va retourner une chaîne de Markov générée aléatoirement :

```
function [W]=cree_chaine_markov(n,nbr_detats,etat_initial)
    % On génère la matrice P de transition:
    P=0.8*eye(nbr_detats);
    P=P+diag(0.1*ones(1,nbr_detats-1),1);
    P=P+diag(0.1*ones(1,nbr_detats-1),-1);
    P(nbr_detats,nbr_detats)=0.9;P(1,1)=0.9;
    % On initialise l'état initial de notre système:
    W=[etat_initial];
    % Algorithme de génération:
    for i=1:n-1
        aleatoire=rand;
        j=W(i);
        somme=P(j,1);
        compteur=1;
        while (somme<=aleatoire) && (compteur<nbr_detats)
            compteur=compteur+1;
            somme=somme+P(j,compteur);
        end
        W(i+1)=compteur;
    end
end
```

On a choisi de ne pas mettre la matrice P en paramètre de la fonction, en préférant plutôt donner sa taille, avec le paramètre nbr_detats .

Ce choix est plus avantageux, car il permet ensuite de tester plus facilement la fonction sans avoir à reconstruire une matrice P stochastique avec la bonne taille.

On a choisi d'utiliser une matrice bistochastique, pour générer notre chaîne mais on peut toujours modifier la fonction pour qu'elle construise une autre matrice stochastique qui générera la chaîne.

La fonction *cree_chaine_markov* commence donc par construire la matrice de transition P de la taille voulue. Après cela, elle crée le vecteur W qui va contenir les termes successifs de la chaîne, elle lui affecte son premier terme, qui correspond à l'état initial du système et qui est donné en paramètre.

Ensuite, la fonction construit la chaîne, en appliquant l'algorithme décrit plus haut, à la matrice P .

On teste notre fonction avec les paramètres suivants :

```
[chaine]=cree_chaine_markov(10,5,2)
```

On génère la chaîne à partir de cette matrice de transition P :

$P =$

$$\begin{matrix} 0.9000 & 0.1000 & 0 & 0 & 0 \\ 0.1000 & 0.8000 & 0.1000 & 0 & 0 \\ 0 & 0.1000 & 0.8000 & 0.1000 & 0 \\ 0 & 0 & 0.1000 & 0.8000 & 0.1000 \\ 0 & 0 & 0 & 0.1000 & 0.9000 \end{matrix}$$

On obtient alors successivement les trois chaînes suivantes :

chaine =

2 2 3 3 4 4 3 3 3 4

chaine =

2 3 3 4 5 5 5 5 5 5

chaine =

2 2 2 1 1 2 3 3 2 2

On remarque que les trois chaînes obtenues sont cohérentes par rapport à la matrice de transition P , servant à leur génération.

Elles commencent bien à l'état initial, donné en paramètre, de l'appel à la fonction et on peut aussi voir que les termes successifs des chaînes sont toujours distants de au plus une unité.

Cela est parfaitement conforme à la matrice de transition P , qui a des termes non nuls, uniquement sur sa diagonale, sa première surdiagonale et sa première sousdiagonale.

À chaque ligne, il y a au maximum trois possibilités, passer à l'état du dessous, passer à l'état du dessus ou rester sur le même état.

On applique maintenant la génération aléatoire de chaîne de Markov à la résolution de notre problème d'optimisation.

3.5.2 Application aux problèmes d'optimisation de la production électrique du barrage hydraulique

Dans notre problème, le système est l'entrée d'eau ponctuelle, les états du système sont les différentes valeurs discrétisées que peut prendre l'entrée d'eau. La loi initiale représente la première valeur que l'entrée va prendre, elle est ici donnée en paramètre de la résolution.

On introduit maintenant la fonction présentée précédemment, dans le fichier.m, qui va contenir l'application à notre algorithme d'optimisation.

On ajoute donc les variables *nbr_detats* et *entree_deau_depart* au paramètre de la nouvelle version de notre fonction Matlab *Optimise* et on lui enlève la variable *type_entree_deau*, car on en a plus besoin ici :

```

% L = taille maximale de la retenue du barrage.
% o = taille maximale de l'ouverture des conduites forcées.
% N = durée de la période de production d'électricité.
% nbr_detats = Nombre de valeurs d'entrée d'eau possible.
% vol_depart = valeur arbitraire du volume présent dans le barrage au départ, pour pouvoir construire le graphe.
% entree_d'eau_depart = valeur de l'entrée le premier jour, celle qui débute la chaîne de markov, donc la simulation.
% P_H_Pleine = Prix de l'électricité, en heure pleine.
% P_H_Creuse = Prix de l'électricité, en heure creuse.
% Periode_H = Période entre les changements de tarifs, heures pleines et heures creuses.

```

Voici le code de la nouvelle fonction *Optimise*, qui effectue la résolution probabiliste à l'aide d'une entrée d'eau, simulée par une chaîne de Markov, aléatoirement générée :

```

function [prod_max,programme,U]=optimise(L,o,N,nbr_detats,vol_depart,entree_d'eau_depart,P_H_Pleine,P_H_Creuse,Periode_H)
% Données
rho=1000;
g=9.80665;
mu=0.75;
% Définition des matrices
V=zeros(N+1,L+1,nbr_detats+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1,nbr_detats+1); % Matrices des valeurs des contrôles, liées aux valeurs de production, coefficient par coefficient
Volume=[0:L]; % Vecteur des valeurs des niveaux de remplissages du barrage.
Controle=[0:o];
Production_elec_prix=ones(1,N+1); % Vecteur des tailles d'ouvertures de la conduite forcée.
Production_elec=ones(1,N+1); % Vecteur des valeurs de production d'électricité en prix, en fonction du temps.
Production_elec_prix_cumule=ones(1,N+1); % Vecteur des valeurs de production d'électricité, en fonction du temps.
Entree_d'eau=[0:nbr_detats]; % Vecteur discrétisé, des différentes valeurs d'entrées d'eau possibles, dans le barrage.
% Construction de la matrice de transition de la chaîne de markov.
P=0.8*eye(nbr_detats+1);
P=P+diag(0.1*ones(1,nbr_detats),1);
P=P+diag(0.1*ones(1,nbr_detats),-1);
P(1,1)=0.9;P(nbr_detats+1,nbr_detats+1)=0.9;
% Construction du vecteur des prix de l'électricité, en fonction du temps:
Prix=[]; % Vecteur des prix.
Compteur=0;
Changement=false;
for i=0:N
    if Compteur==Periode_H
        Compteur=0;
        Changement=not(Changement);
    end
    if Changement
        Prix(i+1)=P_H_Pleine;
    else
        Prix(i+1)=P_H_Creuse;
    end
    Compteur=Compteur+1;
end
% Construction des Matrices U et V, par remontée.
for i=N:-1:1 % Boucle pour le temps
    for j=1:L+1 % Boucle pour le volume
        for l=1:nbr_detats+1 % Boucle pour l'entrée d'eau
            possible=zeros(1,o+1);
            for u=1:o+1 % Boucle pour les contrôles possibles.
                if(Volume(j)-Controle(u)+Entree_d'eau(l)>=0)
                    esperance=0;
                    for k=1:nbr_detats+1 % Seconde Boucle pour l'entrée d'eau
                        esperance=esperance+P(l,k)*V(i+1,round(max(min(L,Volume(j)+k-Controle(u)),1)),k); % Calcul de l'espérance.
                    end
                    possible(1,u)=Volume(j)*rho*g*mu*Controle(u)*Prix(i)+esperance;
                end
            end
            [V(i,j,l),indice]=max(possible);
            U(i,j,l)=Controle(indice);
        end
    end
end

```

```

[prod_max,rang]=max(V(1,:,:));
% Construction de la chaîne de markov, à partir de P et de entree_deau_depart:
[chaîne]=cree_chaine_markov(N+1,nbr_detats+1,entree_deau_depart); % On met nombre d'états +1, car on considère aussi l'état 0.
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart+1,entree_deau_depart+1);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
Production_elec(1)=vol_courant(1)*rho*g*mu*Controle_opt(1);
Production_elec_prix(1)=vol_courant(1)*rho*g*mu*Controle_opt(1)*Prix(i);
Production_elec_prix_cumulee(1)=Production_elec_prix(1);
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+Entree_deau(chaîne(i-1)+1)-Controle_opt(i-1)),0);
    Controle_opt(i)=U(i,round(vol_courant(i))+1,chaîne(i)+1);
    Production_elec(i)=vol_courant(i)*rho*g*mu*Controle_opt(i);
    Production_elec_prix(i)=vol_courant(i)*rho*g*mu*Controle_opt(i)*Prix(i);
    for j=i:N+1
        Production_elec_prix_cumulee(j)=Production_elec_prix_cumulee(j)+Production_elec_prix(i);
    end
end
% Récupération des résultats
programme=Controle_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
% Test fonction sinusoïdale:
x=[0:N]; % Vecteur abscisses du temps.
clf
hold on
subplot(3,1,1);
plot(x,vol_courant,'blue'); % Graphe du volume d'eau, du barrage.
title('Volume d'eau du barrage, en fonction du temps')
subplot(3,1,2);
plot(x,chaîne,'cyan');
title('Entrée d'eau, en fonction du temps')
subplot(3,1,3);
plot(x,Prix,'red');
title('Prix de l'électricité, en fonction du temps')
hold off
figure(2)
clf
hold on
subplot(3,1,1);
plot(x,Production_elec_prix_cumulee,'green');
title('Production électrique cumulée, en fonction du temps')
subplot(3,1,2);
plot(x,Production_elec_prix,'magenta');
title('Production électrique en prix, en fonction du temps')
subplot(3,1,3);
plot(x,Controle_opt,'black');
title('Contrôles appliqués, en fonction du temps')
hold off

```

À partir des éléments de la situation donnés en paramètre, cette fonction effectue la résolution du problème, à l'aide de l'algorithme de programmation dynamique et de l'équation stochastique associé et va également produire une simulation avec différents graphes, pour pouvoir en interpréter la pertinence.

On peut noter que, même pour des données identiques, en paramètre de la fonction *Optimise*, la génération de la chaîne étant aléatoire, on obtiendra pas les mêmes résultats d'une compilation du code à l'autre.

Par rapport au code précédent, il y a plusieurs modifications, dont la majorité correspondent à l'adaptation du code pour une entrée d'eau sous la forme d'une chaîne de Markov.

Pour commencer, on a ajouté un vecteur nommé *Entree_deau*, qui va permettre de discréteriser toutes les valeurs d'entrées d'eaux ponctuelles possibles. Elle contient ici tous les entiers naturels, allant de 0 jusqu'au nombre d'états *nbr_detats* du système.

On construit ensuite la matrice stochastique de transition *P* choisie pour simuler l'entrée d'eau et la remplacer dans l'algorithme de programmation dynamique.

On utilise pour cela la commande Matlab *eye*, qui génère une matrice identité de la taille voulue ainsi que la commande *diag*, qui peut extraire une diagonale d'une matrice carrée ou bien construire une matrice, avec une certaine diagonale à l'emplacement voulu, à partir d'un vecteur de la bonne taille.

P =

$$\begin{matrix} 0.9000 & 0.1000 & 0 & 0 & 0 \\ 0.1000 & 0.8000 & 0.1000 & 0 & 0 \\ 0 & 0.1000 & 0.8000 & 0.1000 & 0 \\ 0 & 0 & 0.1000 & 0.8000 & 0.1000 \\ 0 & 0 & 0 & 0.1000 & 0.9000 \end{matrix}$$

On choisit une matrice comme celle-ci car elle est premièrement symétrique et peu diffuse, ce qui la rend beaucoup plus simple à construire.

Ensuite, on a choisi d'attribuer une pondération très importante sur la diagonale principale de la matrice pour représenter une météo liée à un climat stable.

Pour mieux le comprendre, il faut savoir que les lignes de la matrice représentent chacune un état du système, donc une valeur de l'entrée d'eau allant de 0 jusqu'au nombre d'états *nbr_detats*.

La première ligne représente une entrée d'eau nulle et la dernière ligne représente une entrée d'eau maximale. La première colonne représente le passage à une entrée d'eau nulle et la dernière représente le passage à une entrée d'eau maximale.

Pour que la représentation de la météo soit stable et cohérente, on veut éviter de pouvoir passer d'un extrême à l'autre concernant la valeur de l'entrée d'eau. C'est pourquoi on attribue une part aussi importante de la pondération à la diagonale principale de la matrice de transition.

Cela va avoir pour effet de rendre plus probable la conservation de l'état actuel du système, plutôt que le changement d'état. De plus, le fait que la matrice soit très peu diffuse rend impossible un changement d'état ponctuelle, de plus d'une unité de valeur. Cela permet d'éviter à coup sûr les changements d'entrée d'eau trop brusques.

Maintenant, concernant l'algorithme de programmation dynamique qui effectue la résolution du problème par remontée, il a fallu le modifier, pour qu'il fasse une résolution probabiliste et non pas déterministe, de notre problème.

Comme première grande modification, on a dû rajouter une dimension supplémentaire aux matrices U et V. Cette troisième dimension, indiqué par la lettre *l*, permet de représenter l'entrée d'eau *l*, pour un instant *i* et un volume *j*.

Ensuite une autre modification est celle de l'équation de programmation dynamique et stochastique :

$$V(n, x) = \max_{u \in \mathbb{U}^N} \mathbb{E}(L(x, u) + V(n + 1, f_n(x, u, l), k), \forall n \in \llbracket 0; N - 1 \rrbracket).$$

La variable aléatoire liée à cette espérance est *l*.

Par la linéarité de l'espérance, on obtient d'ailleurs :

$$V(n, x) = \max_{u \in \mathbb{U}^N} (L(x, u) + \mathbb{E}(V(n + 1, f_n(x, u, l), k))), \forall n \in \llbracket 0; N - 1 \rrbracket.$$

Enfin, dans le code même de l'algorithme de résolution, on place en troisième position une nouvelle boucle for, itérant sur *l* de 1 jusqu'à *nbr_detats* + 1, car on doit compter la valeur 0 sachant que les tableaux Matlab sont indiqués à partir de 1.

Ce nouvelle boucle permet de calculer les valeurs de production maximale de V et de contrôles optimaux de U et de le ranger dans les matrices U et V, suivant les trois grandeurs du problème (temps, volume, entrée d'eau) dans les trois différentes dimensions.

Ensuite, plus loin dans l'algorithme on doit calculer chacune des valeurs données par la nouvelle équation de programmation dynamique et stochastique pour pouvoir ensuite récupérer le maximum.

Pour cela, on doit calculer l'espérance pour chacune des valeurs d'entrée d'eau *l*.

On calcule cette espérance grâce à la formule suivante :

$$\mathbb{E}(l) = \sum_{k=1}^{nbr_detats+1} P(l, k) * V(i + 1, f_n(x, u, l), k)$$

On calcule ces espérances pour chacune des valeurs de contrôles possibles en rajoutant une cinquième boucle for, itérant sur k de 1 à $nbr_detats + 1$.
 On multiplie la valeur de production futur, associée à chaque nouvelle entrée d'eau possible, à la probabilité de passé de l'état d'entrée d'eau l actuel au possible futur état d'entrée d'eau k .
 Enfin, on en fait la somme et on l'ajoute à la valeur de production instantanée et on prend le maximum parmi tous les contrôles physiquement applicables.

Dans la seconde moitié de la fonction *Optimise*, on fait un appel à la fonction Matlab *cree_chaine_markov*, présentée dans la partie 3.5.1 de ce rapport.

Cet appel permet de générer aléatoirement une chaîne de Markov à partir de la matrice de transition P et des autres paramètres, puis de la stocker dans la variable nommée *chaine*.

Comme expliqué précédemment, cette chaîne de Markov permet de simuler l'entrée d'eau lors de la construction des vecteurs des graphes de la simulation.

Dans la partie où l'on construit les vecteurs des différents graphes, on rajoute la troisième coordonnée l aux appels des coefficients des matrices U et V .

On remplace aussi les appels aux coefficients de l'ancien vecteur W par les appels aux coefficients du nouveau vecteur de l'entrée d'eau *chaine*.

Concernant le reste du code, il est exactement identique à celui de la résolution précédente.

Quelques exemples de tests du code

On effectue quelques simulations, en faisant variés les paramètres des résolutions et des simulations de différentes manières.

On commence par tester *Optimise* avec la matrice bi-stochastique présentée dans la partie précédente comme matrice de transition pour la génération de la chaîne de Markov.

Avec cette matrice, on va faire deux simulations successives avec exactement les mêmes données en paramètres, afin de bien montrer que la résolution s'adapte, au changement de chaîne, pour l'entrée d'eau.

Toujours avec cette matrice bi-stochastique, on effectuera encore deux autres simulations, dont l'une permettra de mettre en avant l'influence de l'entrée d'eau, sur la résolution du problème et dont l'autre servira à mettre en lumière l'influence du prix horaire de l'électricité sur la résolution de ce même problème.

Ensuite, on teste *Optimise* en modifiant cette fois la matrice de transition P servant à la résolution du problème et à la génération de la chaîne de Markov pour la simulation.

On commence par remplacer P par la matrice identité qui est bien une matrice stochastique (et même bistochastique), car chacune de ses lignes est une mesure de probabilités.

Ce test nous donnera évidemment une chaîne de Markov constante, égale à son premier terme. Le but étant de vérifier que le nouvel algorithme de programmation dynamique nous donne bien des résultats cohérents et conformes à ceux déjà obtenus avec l'algorithme précédent, pour une entrée d'eau constante.

Enfin, on remplacera la matrice P par une autre matrice bi-stochastique, du même type que la première matrice, mais dont les coefficients seront un peu plus diffus autour de la diagonale, toutefois en conservant toujours une pondération importante sur la diagonale principale, pour la cohérence du modèle.

Cette nouvelle matrice de transition donne des chaînes aux variations plus rapide et plus importantes, ce qui rend intéressant de voir comment la résolution s'y adapte.

On observera principalement le graphe du volume de la retenue, ainsi que ceux des contrôles appliquées et de l'entrée d'eau, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité, à la réalité physique.

On commence par les 4 tests utilisant la première matrice de transition P suivante :

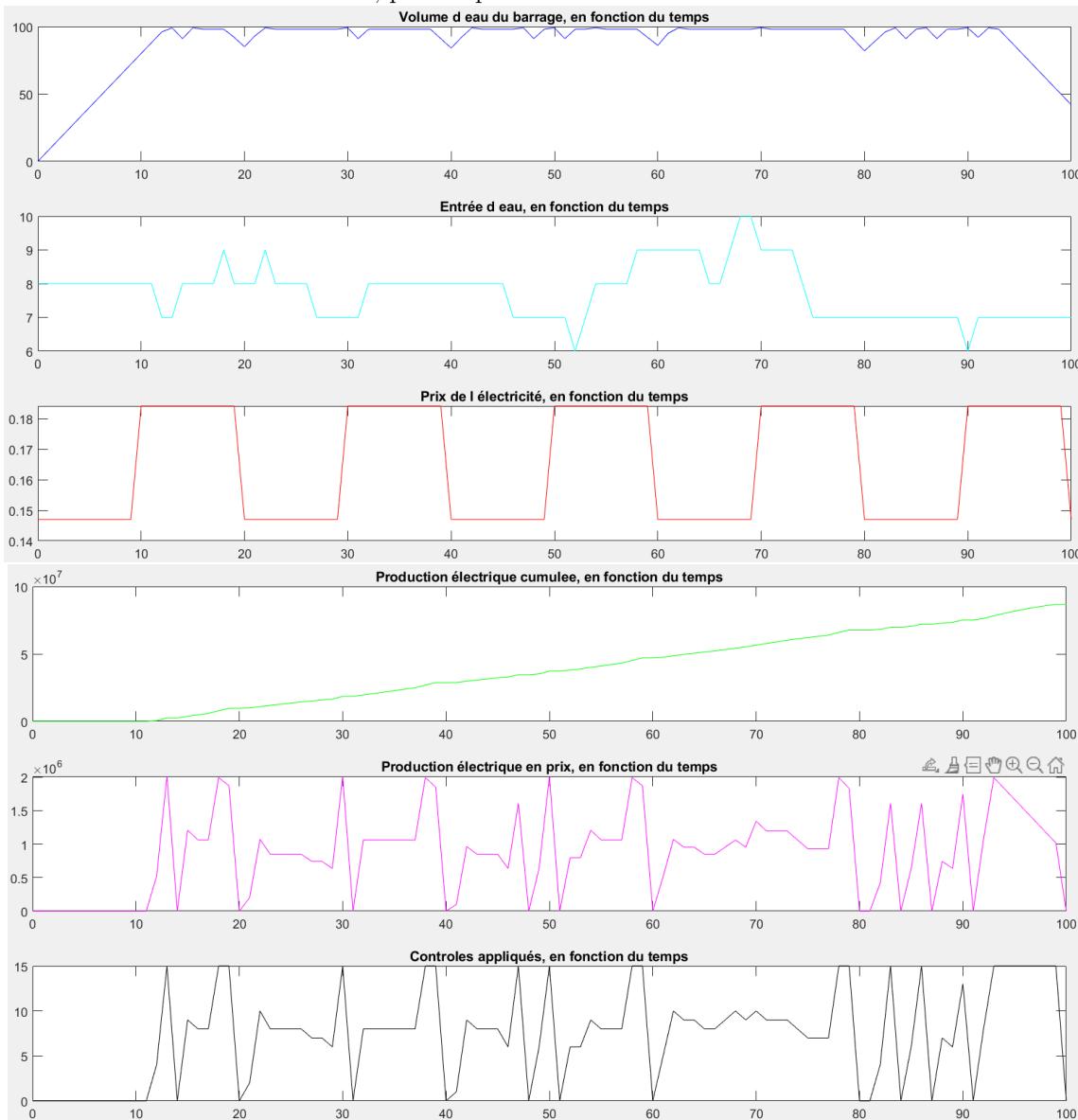
P

Columns 1 through 14

Pour les deux premiers tests avec des paramètres identiques, on utilise les données de résolution suivantes :

```
[production,controles,U]=Optimise(100,15,100,15,0,8,0.1841,0.147,10)
```

On obtient alors les résultats suivants, pour le premier test :



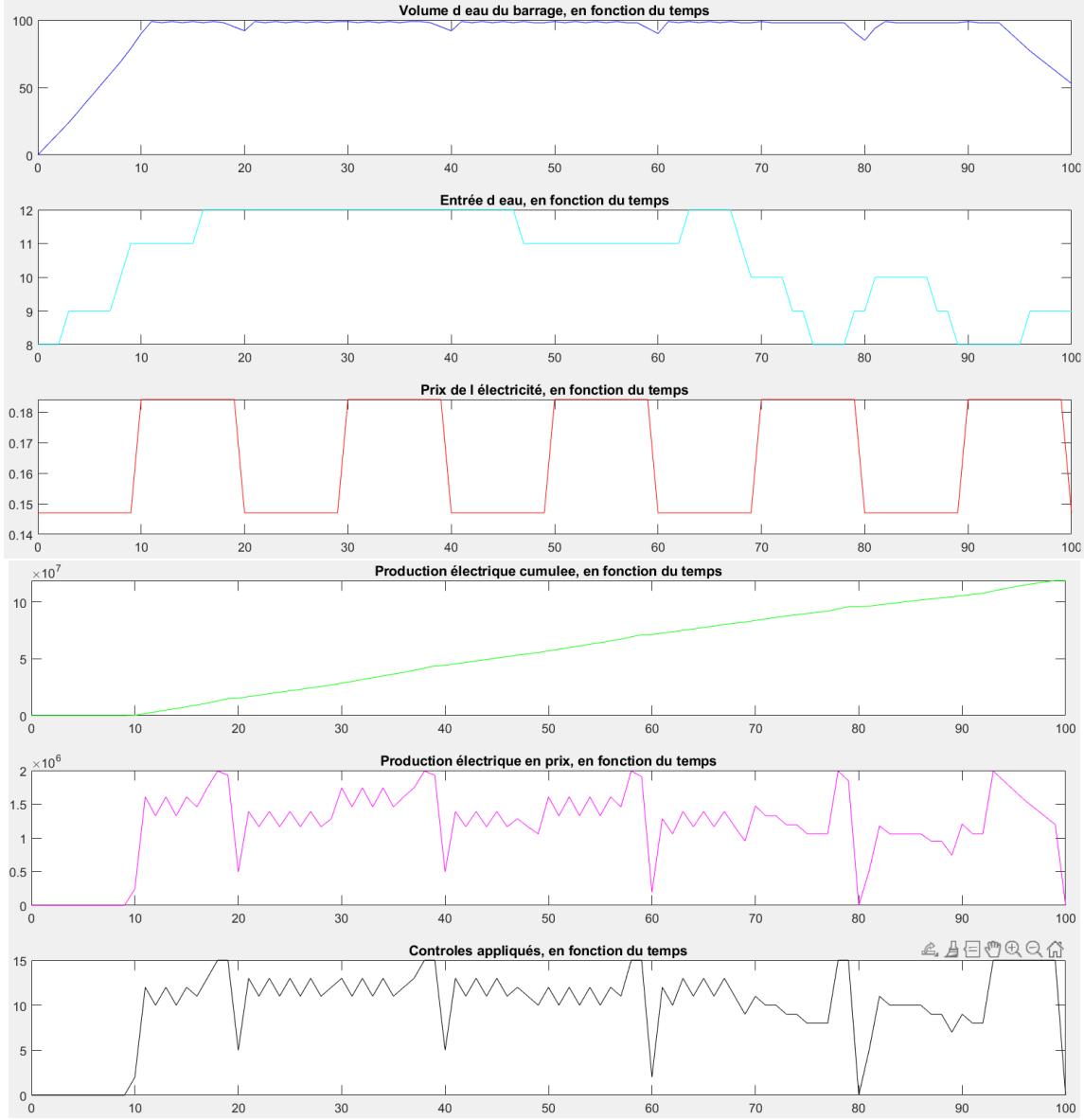
On remarque que les graphes sont bien cohérents entre eux, on peut observer les influences superposées du prix horaire de l'électricité, ainsi que de l'entrée d'eau, dans la retenue du barrage.

L'entrée d'eau est d'ailleurs bien conforme à ce qui est attendu, les variations de la valeur de l'entrée d'eau ne sont pas trop nombreuses mais il y en a quand même, car la probabilité n'est pas non plus nulle et la chaîne est assez longue.

De plus, les changements ponctuels de valeur d'entrée d'eau sont toujours d'au plus une unité, comme prévu par le choix de la matrice de transition.

On effectue maintenant un second test avec exactement les mêmes données en paramètres et la même matrice de transition P .

On obtient alors les résultats suivants :



On peut observer, pour ce nouveau test, que la simulation obtenue est bien différente de celle du premier test. Les paramètres et la matrice de transition P étant les mêmes, la résolution est elle aussi la même.

En revanche, comme dit précédemment, la chaîne de Markov est générée aléatoirement dans la partie simulation du code de la fonction Matlab *Optimise*, les résultats obtenus sont donc bien différents des précédents.

Plus globalement, les graphes sont parfaitement cohérents, l'algorithme remplit la retenue au début de la période de production N . Il ne laisse pas la retenue déborder et il vide toujours fortement la retenue, un peu avant la fin de la période de production d'électricité.

On observe également toujours l'influence du prix horaire de l'électricité et de l'entrée d'eau, sur la résolution de l'algorithme et sur la simulation.

Essayons maintenant de mieux discerner l'influence de ces deux facteurs sur l'adaptation de la résolution par l'algorithme.

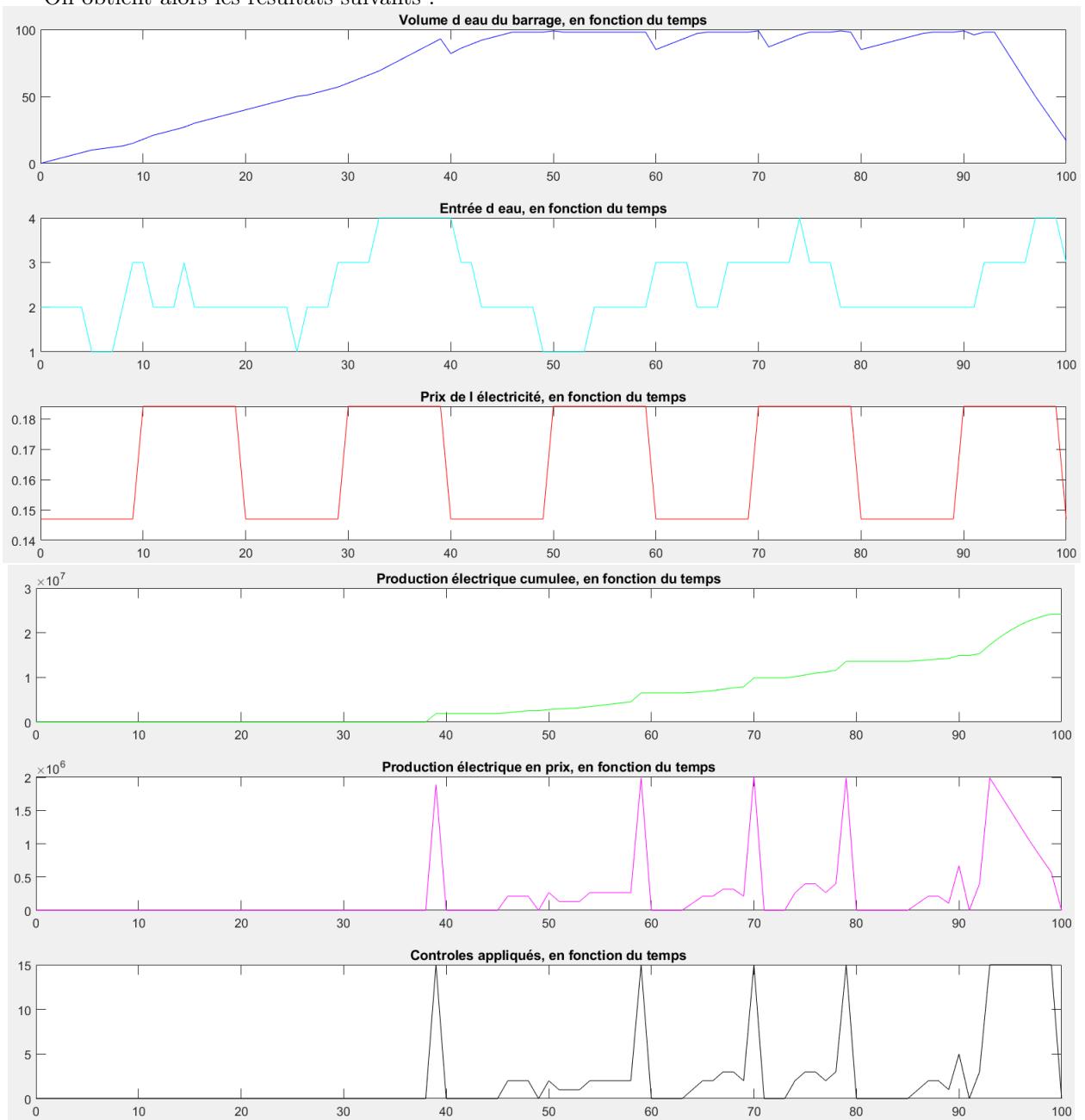
Influence de l'entrée d'eau

Dans ce nouvel exemple de test, on conserve tous les paramètres des tests précédents et on diminue seulement la valeur de départ, de l'entrée d'eau :

```
[production,controles,U]=Optimise(100,15,100,15,0,2,0.1841,0.1470,10)
```

L'objectif est de mieux mettre en avant l'influence de l'entrée d'eau sur la résolution du problème et la simulation de la situation.

On obtient alors les résultats suivants :



On observe que la retenue du barrage met alors beaucoup plus de temps que précédemment à se remplir entièrement.

Ceci s'explique par le fait que l'entrée d'eau initiale est beaucoup plus faible qu'avant et parce que la matrice de transition P est choisie exprès pour limiter les chances de changements de valeur d'entrée d'eau.

On remarque que, tant que la retenue n'atteint pas sa capacité maximale, l'algorithme n'applique absolument aucun contrôle non nul.

On remarque aussi que l'on observe toujours des pics de contrôles très importants à la fin des périodes de tarifs heures pleines de l'électricité.

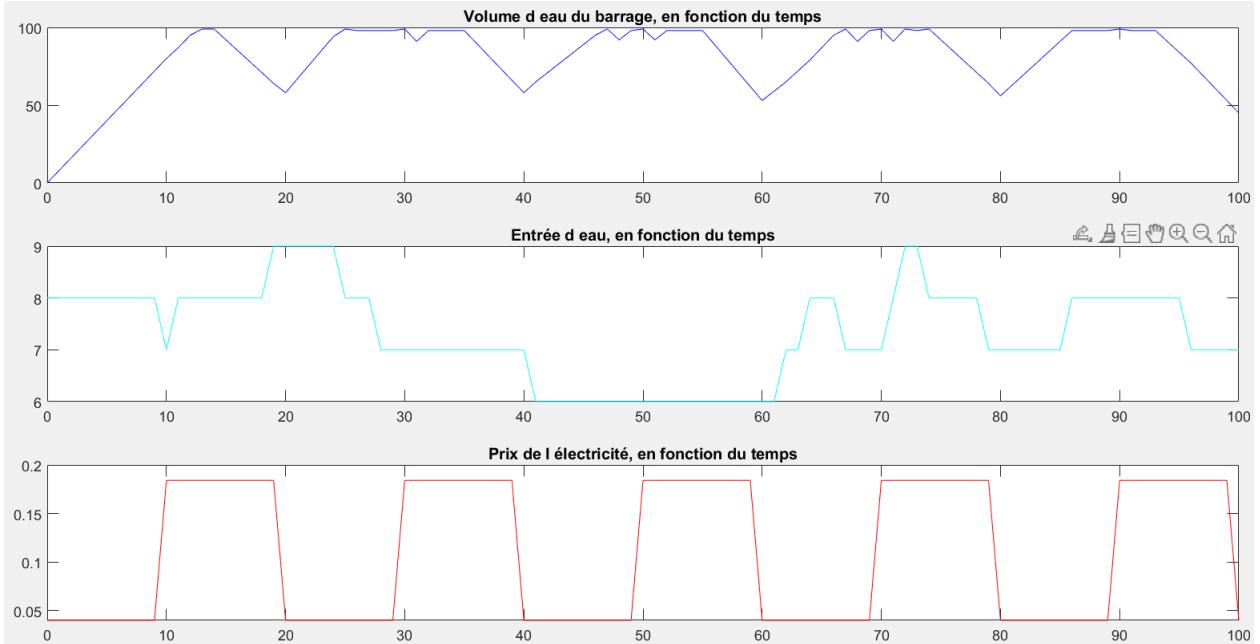
Essayons maintenant de encore mieux discerner l'influence du tarif horaire de l'électricité sur l'adaptation de la résolution par l'algorithme.

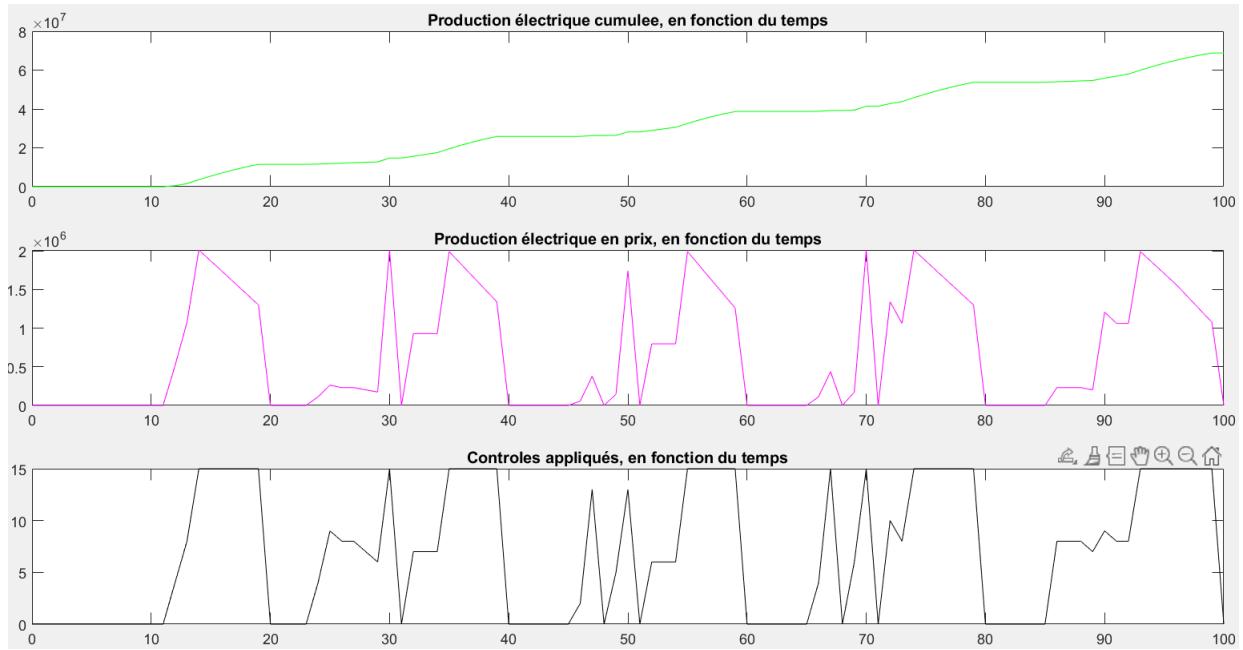
Influence des prix

Dans ce nouvel exemple de test, on conserve tous les paramètres des tests précédents et on diminue seulement la valeur du tarif heures creuses de l'électricité afin d'accroître la différence entre les deux tarifs :

```
[production,contrôles,U]=Optimise(100,15,100,15,0,8,0.1841,0.04,10)
```

On obtient alors les résultats suivants :





Les résultats sont très représentatifs de l'importance de l'influence du tarif horaire de l'électricité, sur la résolution du problème.

On observe que l'algorithme de programmation dynamique vide systématiquement la retenue du barrage de manière synchronisée à la période du tarif horaire de l'électricité à la fin de la période de tarif heures pleines.

De plus, on observe à ces moments-là que l'algorithme vide la retenue, de manière assez importante, ce qui montre une prévalence sur la nécessité, de conserver une hauteur d'eau maximale en permanence dans la retenue.

On remarque aussi que l'algorithme fait toujours attention, à ne jamais laisser la retenue déborder.

Faisons maintenant de nouveau tests, avec des matrices stochastiques de transition différentes.

Optimisation avec une chaîne de Markov, générée à partir de la matrice identité

Pour ce nouveau test, on choisit la matrice identité pour matrice de transition de la chaîne de Markov, qui sera donc constante.

On construit la matrice identité sur Matlab très facilement, grâce à la commande suivante :

% Construction de la matrice de transition de la chaîne de markov.

P=eye(nbr_detats);

on obtient alors la matrice suivante :

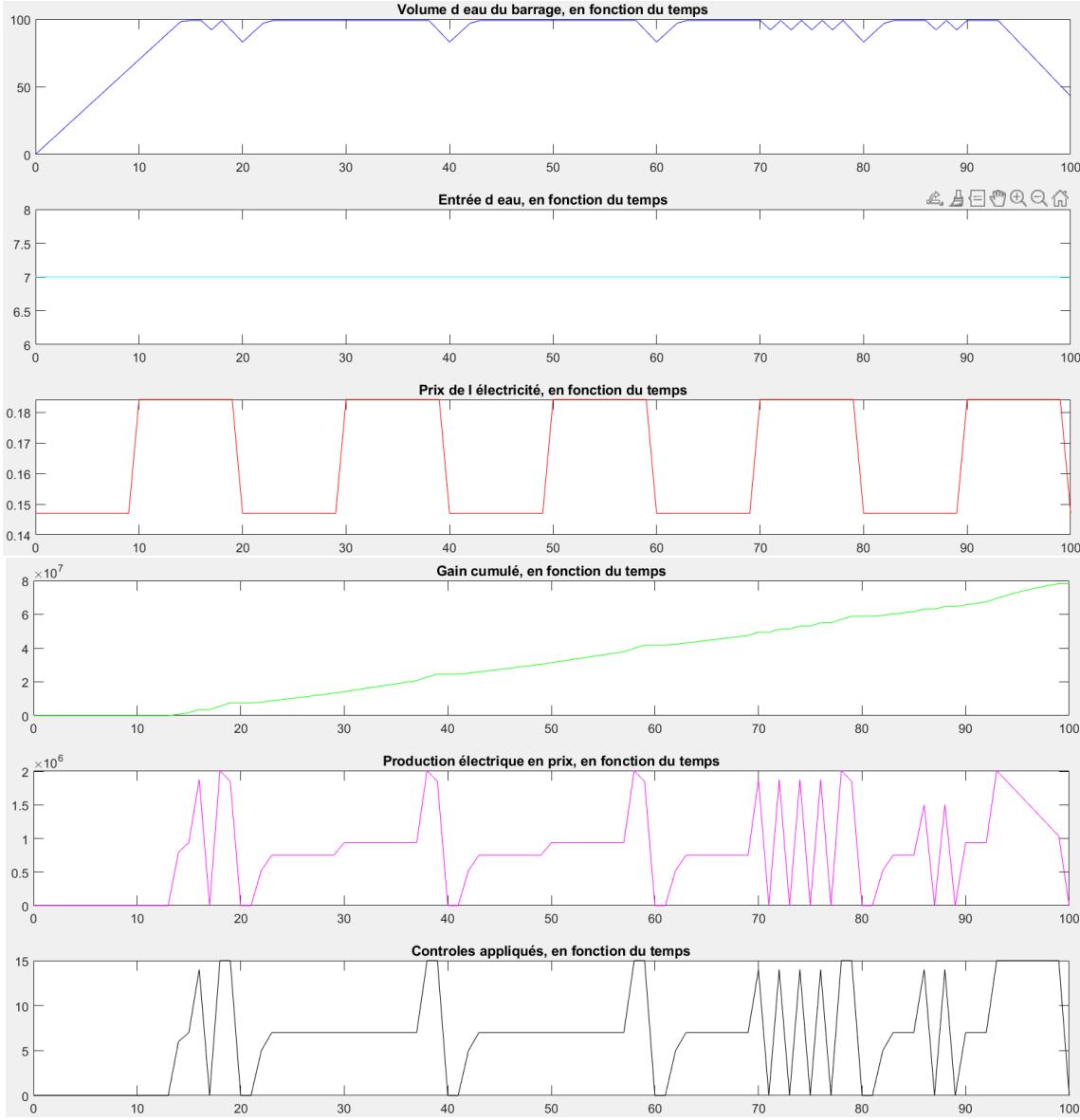
P =

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Pour le test, on utilise les paramètres suivants :

```
[production,controles,U]=Optimise(100,15,100,15,0,7,0.1841,0.1470,10)
```

On obtient ainsi ces résultats :



On observe effectivement, comme attendu, que l'entrée d'eau est bien constante et égale à l'entrée d'eau de départ.

On observe également l'influence du tarif horaire et du prix de l'électricité, principalement sur la fin des différentes périodes de tarif heures pleines.

La résolution est bien conforme aux autres résultats obtenus par l'algorithme des résolutions précédentes, on en déduit que ce nouvel algorithme effectue bien la bonne résolution, pour la nouvelle situation.

Optimisation avec une chaîne de Markov, générée à partir d'une matrice de transition stochastique plus diffuse

Pour ce dernier test, on choisit d'utiliser une matrice de transition stochastique plus diffuse, afin de voir comment l'algorithme s'adapte à des variations plus brusques de la valeur de l'entrée d'eau.

On construit cette nouvelle matrice de transition à l'aide du code suivant :

```

P=0.5*eye(nbr_detats);
P=P+diag(0.2*ones(1,nbr_detats-1),1);
P=P+diag(0.05*ones(1,nbr_detats-2),2);
P=P+diag(0.2*ones(1,nbr_detats-1),-1);
P=P+diag(0.05*ones(1,nbr_detats-2),-2);
P(1,1)=0.75;
P(nbr_detats,nbr_detats)=0.75;

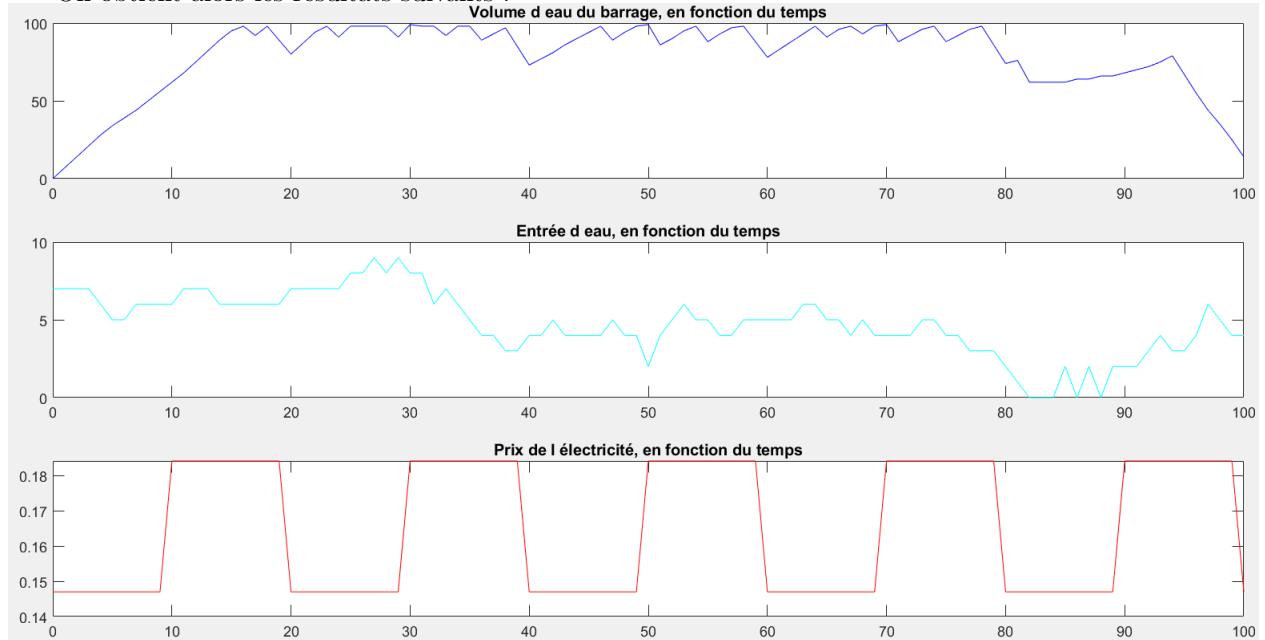
```

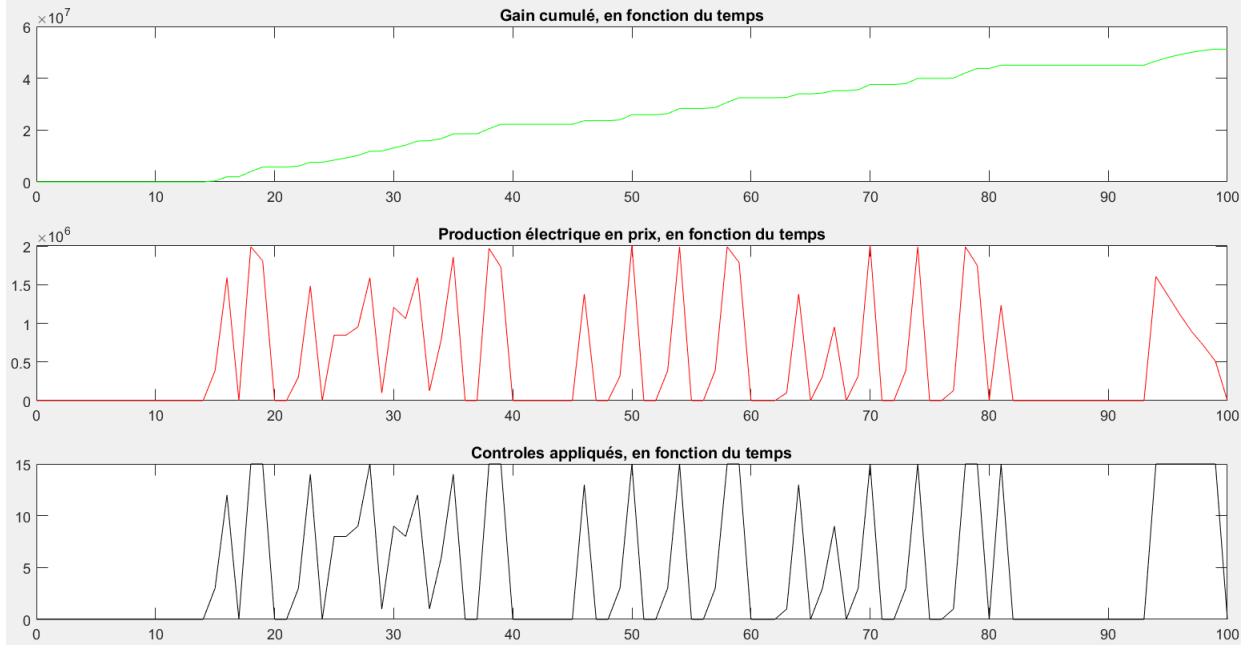
On obtient alors la matrice stochastique suivante :

On choisit de réutiliser exactement les mêmes paramètres que ceux du test précédent :

```
[production,controles]=Optimise(100,15,100,15,0,7,0.1841,0.1470,10)
```

On obtient alors les résultats suivants :





On observe effectivement une entrée d'eau beaucoup moins constant, avec plus de variations, mais aussi et surtout des variations de plus grande amplitude que précédemment.

On observe que l'algorithme adapte effectivement sa résolution et sa simulation au changement plus brusque de l'entrée d'eau.

On voit que le graphe du volume, présent dans la retenue d'eau, est bien moins lisse que précédemment.

Les contrôles appliqués au volume de la retenue n'ont d'ailleurs plus du tout de comportement périodique. Ils sont adaptés en permanence à l'évolution de la situation.

On observe toujours des pics de contrôles liés aux tarifs horaires du prix de l'électricité, en fin de période de tarif heures pleines.

Mais ces pics sont entourés de beaucoup d'autres, dont certains permettent d'empêcher le débordement de la retenue, et dont d'autres permettent plutôt d'adapter la régulation du volume à un nouveau changement de la valeur de l'entrée d'eau.

L'élément le plus probant de ce test est observable à la fin de la période de production d'électricité N , sur les graphes.

En effet, on peut voir que la valeur de l'entrée d'eau s'effondre, jusqu'à même devenir nulle et que le volume de la retenue diminue alors fortement, dans le même temps.

Or on peut aussi remarquer que c'est exactement à ce moment là que l'algorithme stoppe net tout les contrôles ponctuels appliqués au volume de la retenue.

Ensuite, l'entrée d'eau remonte un peu et, au bout d'un moment, l'algorithme se remet à vider au maximum la retenue, avant la fin définitive de la période de production d'électricité N .

Cette observation permet de bien montrer que l'algorithme s'adapte en permanence à l'évolution de la situation, tout en prenant toujours en compte les contraintes à respecter, pour pouvoir optimiser la production totale d'électricité sur l'ensemble de la période de production N .

En dehors tout cela, les autres aspects de la simulation sont bien similaires à ceux des tests précédents.

4 Résolution pour un barrage à 2 retenues

On passe maintenant à la résolution de notre problème d'optimisation de la production électrique pour un barrage comportant 2 retenues d'eaux différentes.

4.1 Résolution avec entrées d'eaux constantes

Pour cette première résolution de notre problème avec un barrage à 2 retenues, on se donne maintenant deux valeurs d'entrées d'eaux journalières W_1 et W_2 , constantes sur toute la durée de production N .

4.1.1 Présentation du code et de la résolution numérique en Matlab

Voici l'ensemble des paramètres de notre nouveau problème :

```
% L = taille maximale de la retenue principale du barrage.
% L2 = taille maximale de la retenue secondaire du barrage.
% o = taille maximale de l'ouverture des conduites forcées, de la retenue principale.
% T = taille maximale de l'ouverture du canal de transfert, de la retenue secondaire.
% N = durée totale de la période de production d'électricité.
% W1 = entrée d'eau journalière, dans la retenue principale.
% W2 = entrée d'eau journalière, dans la retenue secondaire.
% vol_depart = valeur arbitraire du volume présent dans le barrage au départ, pour pouvoir construire le graphe.
% vol_reserve_depart = valeur arbitraire du volume au départ, dans la seconde retenue du barrage.
```

Par rapport à la première résolution, avec entrée d'eau journalière W constante, on a rajouté la valeur de la taille de la seconde retenue du barrage L_2 .

On ajoute aussi la valeur maximale des contrôles applicables T au volume de cette retenue secondaire. On parlera plus de transferts pour désigner ces contrôles, et les dissocier des contrôles appliqués au volume de la retenue principale du barrage.

On ajoute également W_1 et W_2 , les valeurs constantes des entrées d'eaux, respectivement de la première et de la seconde retenue du barrage.

Enfin, on ajoute, en complémentarité de vol_depart , la nouvelle variable $vol_reserve_depart$, qui contiendra la valeur du volume d'eau contenu dans la retenue secondaire au début de la période de production d'électricité N dans la partie simulation du code.

Voici le code de la fonction qui effectue la résolution basée sur la nouvelle adaptation de l'algorithme de programmation dynamique, et qui réalise aussi une simulation à partir des paramètres de la situation, qui sont donnés en entrée de la fonction, avant de retourner tous les résultats voulus et d'afficher les graphes de la simulation :

```
function [prod_max,Programme1,Programme2]=Optimise_Production(L,L2,o,T,N,W1,W2,vol_depart,vol_reserve_depart)
% Données
rho=1000;
g=9.80665;
mu1=0.2;
mu2=0.8;
% Définition des matrices
V=zeros(N+1,L+1,L2+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1,L2+1); % Matrices des valeurs des contrôles, pour la retenue principale.
U2=zeros(N+1,L+1,L2+1); % Matrices des valeurs des transferts, entre la retenue secondaire et la retenue principale.
Volume=[0:L]; % Vecteur des valeurs des niveaux de remplissages, de la retenue principale du barrage.
Volume2=[0:L2]; % Vecteur des valeurs des niveaux de remplissages, de la retenue secondaire du barrage.
Controle=[0:o]; % Vecteur des tailles d'ouvertures de la conduite forcée.
Transfert=[0:T]; % Vecteur des valeurs possibles de transferts entre la retenue principale et secondaire.
% Construction des Matrices U et V, par remontée.
for i=N:-1:1 % Boucle sur le temps
    for j=1:L+1 % Boucle sur le volume du barrage
        for k=1:L2+1 % Boucle sur le volume de la retenue
            optimal2=zeros(o+1,T+1); % Contiendra le maximum par rapport aux transferts, pour chaque contrôle.
            for t=1:T+1
                if(Volume2(k)+W2-Transfert(t)>=0)
                    for u=1:o+1
                        if((Volume(j)+W1-Controle(u)+Transfert(t)>=0))
                            optimal2(u,t)=rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t))+V(i+1,round(max(min(L,
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```

```

        end
    V(i,j,k)=max(max(optimal2));
    [x,y]=find(optimal2==max(max(optimal2)));
    U(i,j,k)=Controle(x(1));
    U2(i,j,k)=Transfert(y(1));
end
end
[prod_max,rang]=max(V(1,:,:));
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U,U2 et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart+1,vol_reserve_depart+1);
Transfert_opt=zeros(1,N+1);
Transfert_opt(1)=U2(1,vol_depart+1,vol_reserve_depart+1);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
vol_reserve=zeros(1,N+1);
vol_reserve(1)=vol_reserve_depart;
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+W1-Controle_opt(i-1)+Transfert_opt(i-1)),0);
    vol_reserve(i)=max(min(L2,vol_reserve(i-1)+W2-Transfert_opt(i-1)),0);
    Controle_opt(i)=U(i,round(vol_courant(i))+1,round(vol_reserve(i))+1);
    Transfert_opt(i)=U2(i,round(vol_courant(i))+1,round(vol_reserve(i))+1);
end
% Récupération des résultats
Programme1=Controle_opt(1:N);
Programme2=Transfert_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
% Test fonction sinusoidale:
x=[0:N]; % Vecteur abscisses du temps.
clf
hold on
subplot(4,1,1);
plot(x,vol_courant,'blue'); % Graphe du volume d'eau, du barrage.
title('Volume d eau de la retenue principale, en fonction du temps')
subplot(4,1,2);
plot(x,vol_reserve,'cyan');
title('volume d eau de la retenue secondaire, en fonction du temps')
subplot(4,1,3);
plot(x,Controle_opt,'red');
title('Controle envoyé à la centrale, en fonction du temps')
subplot(4,1,4);
plot(x,Transfert_opt,'green');
title('Transfert d eau, en fonction du temps')
hold off

```

Voici la ligne coupée dans la capture d'écran précédente du code :

```
optimal2(u,t)=rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t))+V(i+1,round(max(min(L,Volume(j)+W1-Controle(u)+Transfert(t)),1)),round(max(min(L2,Volume2(k)+W2-Transfert(t)),1)));
```

Ce code est une adaptation de celui qui résout notre problème dans le cas d'un barrage à une retenue.

Les rendements μ_1 et μ_2 sont des constantes dont les valeurs sont affectées à deux variables respectives du même nom dans la fonction. On choisit de les initialiser dans la fonction plutôt que de les placer en paramètre, car dans la réalité ce sont des données fixes. Les rendements dépendent du barrage et ne peuvent pas être modifiés comme on le souhaite.

μ_1 correspond au rendement associé à la retenue principale et μ_2 correspond au rendement associé à la retenue secondaire du barrage.

Dans la définition des matrices et des vecteurs de discréétisation, qui servent à la résolution et à la simulation du problème, on rajoute de nouveau une troisième dimension au matrice U et V .

Cette nouvelle dimension permettra des stocker les valeurs optimales de production et de contrôles par rapport au niveau de remplissage de la retenue secondaire du barrage. Il y aura donc $L2 + 1$ cases pour chaque valeur de temps i et de volume de la retenue principale j données.

On ajoute également une nouvelle matrice nommée $U2$, elle est du même format que V et U et elle permet de stocker les valeurs des transferts d'eaux optimaux de la retenue secondaire à la retenue principale, pour chaque valeur de temps i , chaque valeur de remplissage de la retenue principale j et chaque valeur de remplissage de la retenue secondaire k .

Enfin, pour compléter les vecteurs de discréétisation $Volume$ et $Controle$, on ajoute les vecteurs $Volume2$

et *Transfert*, qui permettent de discréteriser respectivement les valeurs des différents niveaux de remplissage de la retenue secondaire et les valeurs des différents transferts d'eaux applicables, au volume d'eau présent dans cette même retenue secondaire du barrage.

On modifie également l'algorithme de programmation dynamique pour qu'il fasse maintenant la résolution du problème pour un barrage à deux retenues.

On rajoute pour cela une nouvelle boucle for, itérant pour k allant de 1 à $L2 + 1$, elle permet d'itérer sur la troisième dimension des matrices V, U et $U2$, pour représenter les valeurs du volume présent dans la retenue secondaire.

Ensuite, on ajoute également une autre boucle for, cette fois-ci itérant pour t allant de 1 à $T + 1$, pour représenter les valeurs de tous les transferts d'eaux applicables de la retenue secondaire à la retenue principale.

Pour accompagner cette boucle for, on ajoute également un nouveau test, utilisant une structure conditionnelle *if*, qui permet de vérifier que la valeur de transfert itérée par la boucle for est bien physiquement applicable à la retenue secondaire du barrage.

C'est à dire que ce test vérifie qu'il reste bien assez d'eau dans la retenue secondaire pour pouvoir faire le transfert en question. Si ce n'est pas le cas, il ne fera pas le calcul et laissera la valeur 0 aux coefficients correspondant aux données (i,j,k) dans les matrice V , U et $U2$.

Enfin, dans l'équation de programmation dynamique, au coeur de l'algorithme de remontée, on modifie la fonction valeur de production instantanée $L(x, u)$ (cf partie 1.2.1), pour qu'elle prenne maintenant en compte les valeurs des rendements μ_1 et μ_2 des deux retenues du barrage.

Toujours dans l'équation de programmation dynamique, mais cette fois-ci dans l'appel récursif au coefficient de la matrice V , on modifie la mise à jour du volume de la première retenue et on ajoute aussi celle de la seconde retenue, sur la troisième dimension du coefficient de la matrice V .

Étant donné que l'on considère dans ce modèle, que la retenue secondaire du barrage est placé en amont de la retenue principale, on suit alors les formules de mise à jour des volumes suivantes :

$$x_{n+1} = x_n + W1_n - u_n + t_n$$

$$y_{n+1} = y_n + W2_n - t_n$$

Où x_n est le volume de la retenue principale au temps n et y_n est celui de la retenue secondaire au temps n .

De plus, on doit maintenant récupérer la valeur maximale de production et celles des contrôles associés pour un temps i et des volumes j et k données, parmi tous le couples de contrôles et de transferts possibles. Comme en plus on peut maintenant avoir plusieurs couples de contrôles optimaux différents pour une même donnée de (i,j,k) , on choisit pour faire plus simple de prendre le premier max obtenu, dans l'ordre.

Pour trouver ce max, un double appel à la commande Matlab *max*, sur la matrice *optimal*, regroupant toutes les possibilités de production, pour tous les couples de contrôles possibles, suffit. Pour obtenir les valeurs du couple de contrôles associé au maximum de production, on utilise la commande Matlab *find* avec la commande *max*.

De la même manière que dans la résolution précédente du problème pour un barrage à une retenue, on construit les vecteurs des valeurs de productions ponctuels et de productions cumulées, en adaptant bien évidemment la mise à jour des volumes d'eaux pour les deux retenues du barrage.

On construit ensuite tous les graphes de la même manière que précédemment.

4.1.2 Quelques exemples de tests du code

On effectue quelques simulations avec différentes données en paramètres et on observe principalement les graphes des volumes des retenues et des différents contrôles appliquées, en fonction du temps. On tente

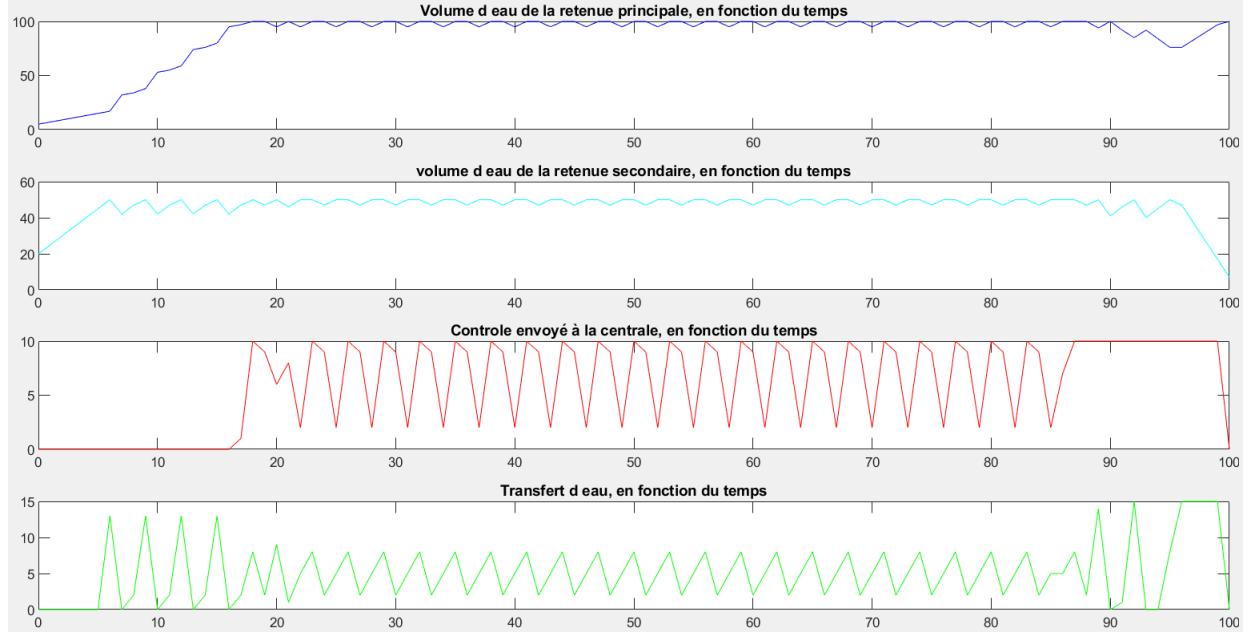
d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité à la réalité physique.

On commence par ce premier test :

```
[production,prog1,prog2]=Optimise_Production(100,50,10,15,100,2,5,5,20)
```

Pour ce premier test, le rendement μ_1 de la retenue principale est de 0.2 et le rendement μ_2 de la retenue secondaire est de 0.8.

On obtient alors ces résultats :



On peut observer, comme attendu, qu'au début de la période de production N la retenue secondaire aide beaucoup la retenue principale à se remplir, avec de grands pics de transferts.

Une fois la retenue principale pleine, ces pics de transfert cessent et se stabilisent sur un rythme toujours périodique et soutenu, mais avec une amplitude plus faible.

Les contrôles sur le volume de la retenue principale commencent à ce moment précis.

L'algorithme fait toujours en sorte de maintenir en permanence les volumes des retenues à un niveau maximal, tout en évitant toujours soigneusement les débordements.

On peut quand même remarquer que, comme la retenue principale reçoit les transferts provenant de la retenue secondaire, l'algorithme privilégie alors des contrôles très importants et à un rythme très soutenu sur le volume de cette retenue.

Pour la retenue secondaire, les transferts sont moins importants car ils sont principalement influencés par l'entrée d'eau $W2$.

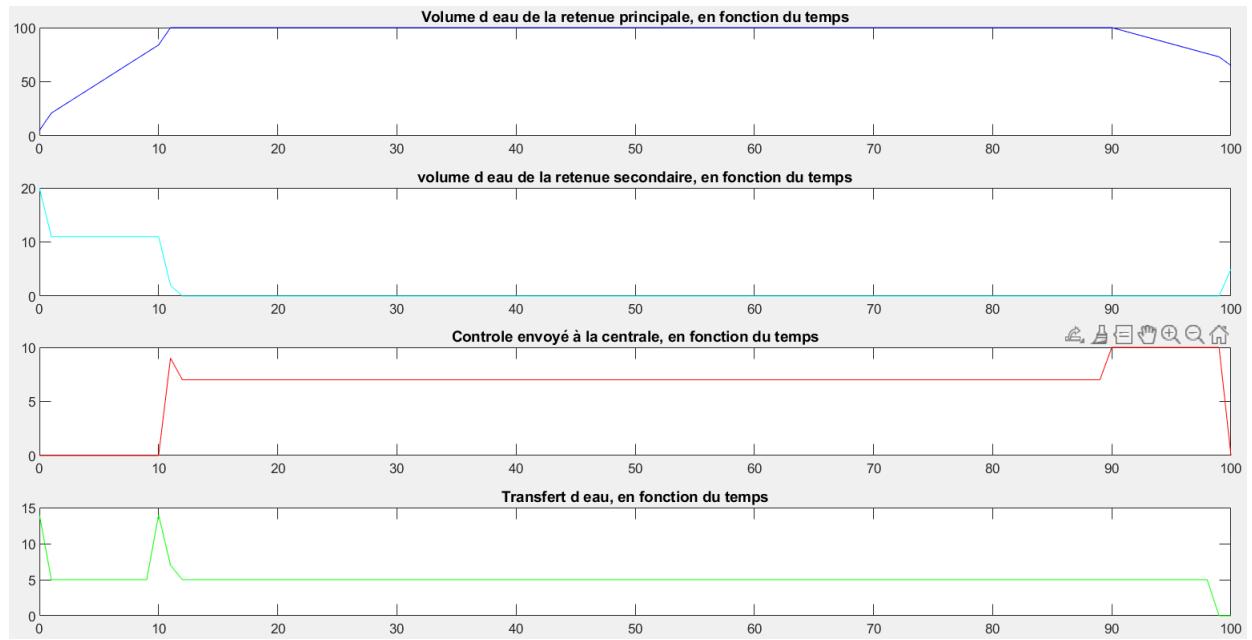
On observe un équilibre de la résolution entre un rendement 4 fois plus important pour la seconde retenue et un volume maximale deux fois plus important pour la première retenue.

On remarque également que, sur la fin de la période de production d'électricité N , les contrôles de la retenue principale sont maximaux jusqu'au bout, et pourtant le volume d'eau présent dans la retenue principale augmente. Cela est dû au fait que la retenue secondaire se vide dedans au même moment.

On reprend exactement les mêmes paramètres pour ce second test, car on souhaite pouvoir observer l'influence de l'évolution des deux rendements sur la résolution et la simulation de l'algorithme :

```
[production,prog1,prog2]=Optimise_Production(100,50,10,15,100,2,5,5,20)
```

Pour ce deuxième test, on a inversé les valeurs des deux rendements, le rendement μ_1 de la retenue principale est de 0.8 et le rendement μ_2 de la retenue secondaire est de 0.2.



En inversant les rendements, on observe que la résolution et la simulation sont radicalement différentes du test précédent.

En effet, on observe que dès le début de la période de production N , la retenue secondaire se vide immédiatement de moitié, puis stabilise ses transferts au rythme de son entrée d'eau W_2 , afin d'aider la première retenue à se remplir.

Une fois la première retenue remplie, la seconde retenue se vide en totalité et envoie par la suite toute son entrée d'eau journalière W_2 directement à la retenue principale, restant ainsi vide pendant tous le reste de la période de production N .

En ce qui concerne la retenue principale, une fois remplie, elle stabilise ses contrôles à un niveau constant, dépendant de son entrée d'eau W_1 et des transferts de la retenue secondaire, c'est à dire de W_2 .

Le volume de la retenue principale est toujours maximal jusqu'à la fin de la période de production d'électricité N , où la retenue se vide un peu, car les contrôles sont alors maximaux pour optimiser la production.

Ce choix de l'algorithme, qui consiste à vider entièrement la retenue secondaire, est parfaitement logique.

Dans notre situation, le rendement μ_1 de la retenue principale est 4 fois plus important que celui de la retenue secondaire.

De plus, le volume maximal de la retenue principale est toujours deux fois plus important que celui de la seconde retenue.

Tout cela fait que vider entièrement la retenue secondaire en permanence, au profit du volume de la retenue principale, permet de produire plus qu'en exploitant une hauteur d'eau maximale dans les deux retenues.

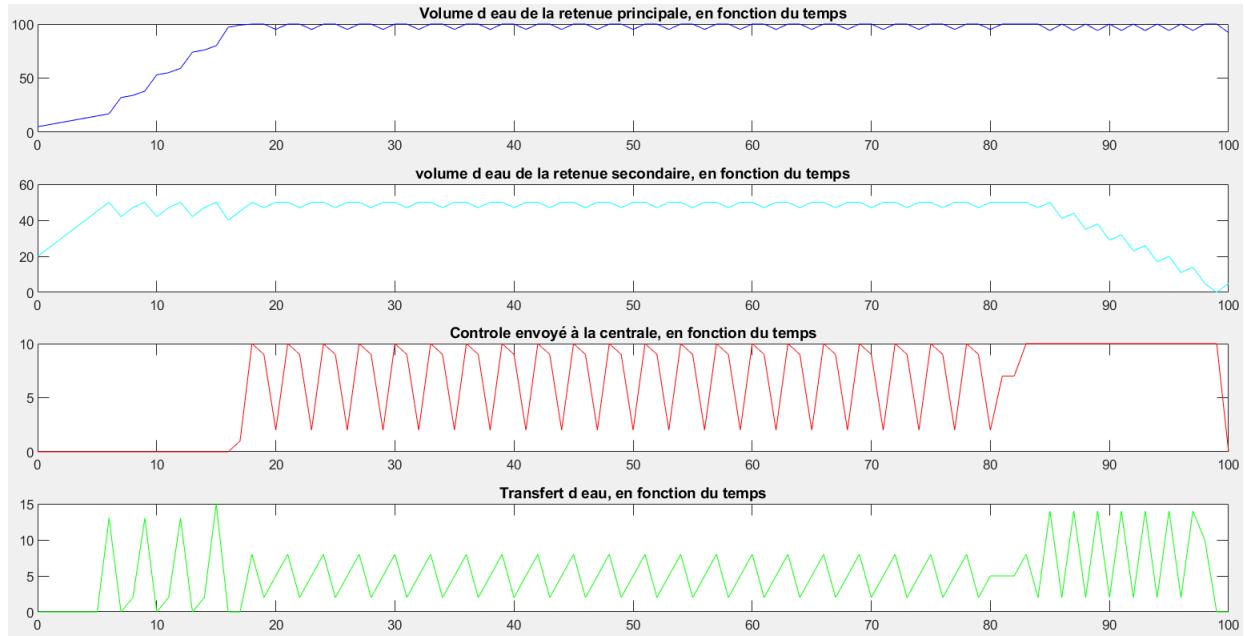
Il est à noter que ce choix est aussi influencé par la capacité maximale de contrôle, des deux retenues.

On reprend encore exactement les mêmes paramètres pour ce troisième test, car on souhaite pouvoir observer l'influence des deux rendements sur la résolution et la simulation réalisé par l'algorithme lorsqu'ils sont égaux :

```
[production,prog1,prog2]=Optimise_Production(100,50,10,15,100,2,5,5,20)
```

Pour ce troisième test, on a choisi la même valeur pour les deux rendements, le rendement μ_1 de la

retenue principale est de 0.5 et le rendement μ_2 de la retenue secondaire est aussi de 0.5.



Même en fixant cette fois-ci les valeurs des rendements des deux retenues à égalité, on peut toujours observer l'influence de la différence entre le volume maximal de la retenue principale et celui de la seconde retenue.

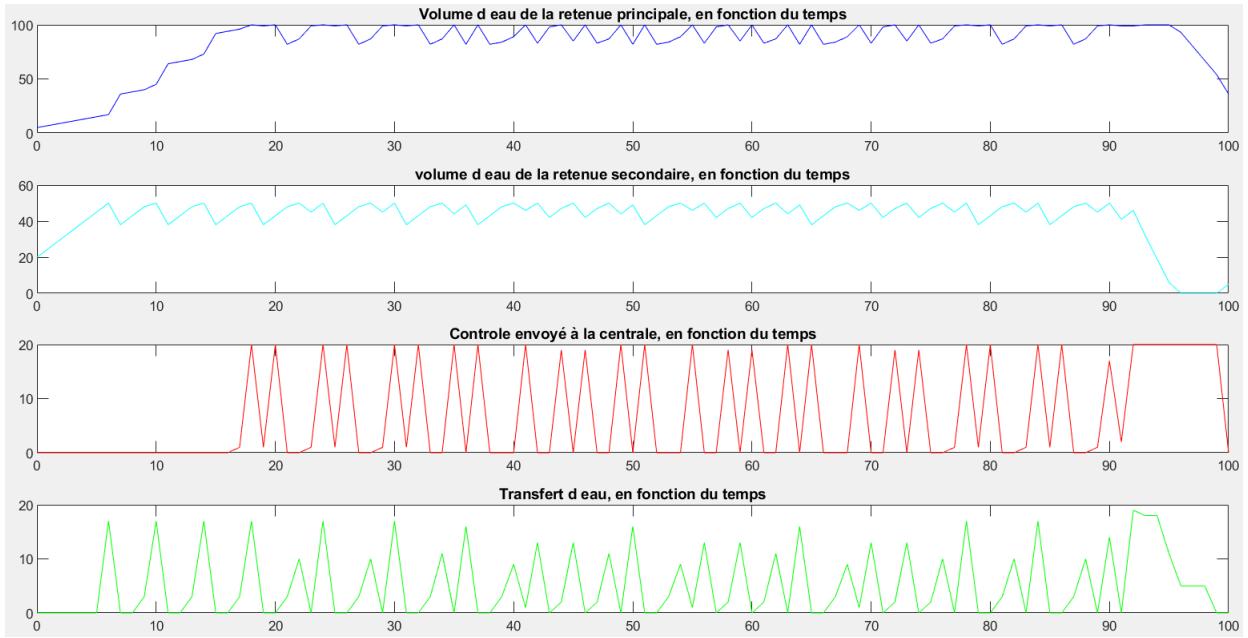
La retenue secondaire aide toujours la première à se remplir au début de la période de production N , et les deux retenues se vident toujours fortement à la fin de cette même période.

On observe encore que la retenue principale privilégie des contrôles très importants car elle reçoit toujours les transferts de la retenue secondaire.

On effectue maintenant un dernier test, où l'on augmente en même temps, les capacités maximales des contrôles applicables sur les volumes d'eaux contenus dans les deux retenues du barrage :

```
[production,prog1,prog2]=Optimise_Production(100,50,20,20,100,2,5,5,20)
```

Pour ce dernier test, on a toujours la même valeur pour les deux rendements, le rendement μ_1 est de 0.5 et le rendement μ_2 est aussi de 0.5.



En augmentant maintenant les valeurs maximales des contrôles applicables aux deux retenues, on observe toujours un comportement de l'algorithme similaire à celui du test précédent, mais quand même avec quelques adaptations.

On observe que la première retenue se remplit légèrement plus rapidement, et que, par la suite, ses pics de contrôles sont moins resserrés et atteignent bien la valeur maximale applicable.

Enfin, pour finir, on remarque également qu'à la fin de la période de production N , les deux retenues se vident toute deux presque entièrement.

4.2 Résolution avec ajouts des entrées d'eaux non constantes

Pour cette nouvelle résolution de notre problème avec un barrage à 2 retenues, on simule maintenant les entrées d'eaux journalières W_1 et W_2 , au choix par une fonction périodique ou gaussienne sur toute la durée de production N .

4.2.1 Présentation du code et de la résolution numérique en Matlab

Voici l'ensemble des paramètres, de notre nouveau problème :

```
% L = taille maximale de la retenue principale du barrage.
% L2 = taille maximale de la retenue secondaire du barrage.
% o = taille maximale de l'ouverture des conduites forcées, de la retenue principale.
% T = taille maximale de l'ouverture du canal de transfert, de la retenue secondaire.
% N = durée totale de la période de production d'électricité.
% type_retenue1 = sélectionne le type d'entrée d'eau dans la retenue, les valeurs possibles sont "periodique" ou "crue".
% type_retenue2 = sélectionne le type d'entrée d'eau dans la retenue, les valeurs possibles sont "periodique" ou "crue".
% vol_depart = valeur arbitraire du volume présent dans le barrage au départ, pour pouvoir construire le graphe.
% vol_reserve_depart = valeur arbitraire du volume au départ, dans la seconde retenue du barrage.
```

Exactement sur le même principe que pour la résolution de ce problème pour un barrage à une retenue, on introduit deux nouvelles données, $type_retenue1$ et $type_retenue2$.

Ce sont des chaînes de caractères qui peuvent prendre pour valeur "periodique" ou "crue", elles servent à choisir respectivement les entrées de la première et seconde retenue du barrage.

Voici le code de la fonction qui effectue la résolution et qui réalise aussi une simulation à partir des paramètres de la situation qui sont donnés en entrée, avant de retourner tous les résultats voulus et d'afficher les graphes de la simulation :

```
function [prod_max,Programme1,Programme2]=Optimise_Production(L,L2,o,T,N,type_retenue1,type_retenue2,vol_depart,vol_reserve_depart)
    % Données
    rho=1000;
    g=9.80665;
    mu1=0.2;
    mu2=0.8;
    % Définition des matrices
    V=zeros(N+1,L+1,L2+1);      % Matrices des valeurs de production d'électricité entre les instants n et N.
    U=zeros(N+1,L+1,L2+1);      % Matrices des valeurs des contrôles, pour la retenue principale.
    U2=zeros(N+1,L+1,L2+1);     % Matrices des valeurs des transferts, entre la retenue secondaire et la retenue principale.
    Volume=[0:L];               % Vecteur des valeurs des niveaux de remplissages, de la retenue principale du barrage.
    Volume2=[0:L2];              % Vecteur des valeurs des niveaux de remplissages, de la retenue secondaire du barrage.
    Contrôle=[0:o];              % Vecteur des tailles d'ouvertures de la conduite forcée.
    Transfert=[0:T];             % Vecteur des valeurs possibles de transferts entre la retenue principale et secondaire.
    Production_elec=ones(1,N+1); % Vecteur des valeurs de production d'électricité, en fonction du temps.
    Production_elec_cumulée=ones(1,N+1); % Vecteur des valeurs de production d'électricité cumulés, en fonction du temps.
    % Construction des vecteurs des entrées d'eau:
    W1=[];                      % Vecteurs des entrées journalières.
    W2=[];
    if(type_retenue1=="crue")
        for i=0:N
            W1(i+1)=2+(L/4)*exp(-(i-(N/2))^2/(N/4));
        end
    elseif(type_retenue1=="periodique")
        for i=0:N
            W1(i+1)=(L/8)+(L/8)*cos((N/2)*i);
        end
    end
    Programme1=U;
    Programme2=U2;
    prod_max=Production_elec;
    for i=1:N
        prod_max(i+1)=prod_max(i)+Production_elec(i+1);
    end
    % Affichage des résultats
    % Graphiques
    % ...
end
```

```

        end
    else
        for i=0:N
            W1(i+1)=2;
        end
    end
    if(type_retenue2=="crue")
        for i=0:N
            W2(i+1)=2+(L2/4)*exp(-(i-(N/2))^2/(N/4));
        end
    elseif(type_retenue2=="periodique")
        for i=0:N
            W2(i+1)=(L2/8)+(L2/8)*cos((N/2)*i);
        end
    else
        for i=0:N
            W2(i+1)=2;
        end
    end
    % Construction des Matrices U et V, par remontée.
    for i=N:-1:1           % Boucle sur le temps
        for j=1:L+1          % Boucle sur le volume du barrage
            for k=1:L2+1       % Boucle sur le volume de la retenue
                optimal2=zeros(0+1,T+1); % Contiendra le maximum par rapport aux transferts, pour chaque contrôle.
                for t=1:T+1
                    if(Volume2(k)+W2(i)-Transfert(t)>=0)
                        for u=1:o+1
                            if((Volume(j)+W1(i)-Controle(u)+Transfert(t)>=0))
                                optimal2(u,t)=rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t))+V(i+1,round(max(min(L,

```

```

                                end
                            end
                        end
                    end
                end
                V(i,j,k)=max(max(optimal2));
                [x,y]=find(optimal2==max(max(optimal2)));
                U(i,j,k)=Controle(x(1));
                U2(i,j,k)=Transfert(y(1));
            end
        end
    end
    [prod_max,rang]=max(V(1,:,:));
    % Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U,U2 et V.
    Controle_opt=zeros(1,N+1);
    Controle_opt(1)=U(1,vol_depart+1,vol_reserve_depart+1);
    Transfert_opt=zeros(1,N+1);
    Transfert_opt(1)=U2(1,vol_depart+1,vol_reserve_depart+1);
    vol_courant=zeros(1,N+1);
    vol_courant(1)=vol_depart;
    vol_reserve=zeros(1,N+1);
    vol_reserve(1)=vol_reserve_depart;
    Production_elec(1)=rho*g*(vol_courant(1)*mu1*Controle_opt(1)+vol_reserve(1)*mu2*Transfert_opt(1));
    Production_elec_cumulee(1)=Production_elec(1);
    for i=2:N+1
        vol_courant(i)=max(min(L,vol_courant(i-1)+W1(i)-Controle_opt(i-1)+Transfert_opt(i-1)),0);
        vol_reserve(i)=max(min(L2,vol_reserve(i-1)+W2(i)-Transfert_opt(i-1)),0);
        Controle_opt(i)=U(i,round(vol_courant(i))+1,round(vol_reserve(i))+1);
        Transfert_opt(i)=U2(i,round(vol_courant(i))+1,round(vol_reserve(i))+1);
        Production_elec(i)=rho*g*(vol_courant(i)*mu1*Controle_opt(i)+vol_reserve(i)*mu2*Transfert_opt(i));
        for j=i:N+1
            Production_elec_cumulee(j)=Production_elec_cumulee(j)+Production_elec(i);
        end
    end
    % Récupération des résultats
    Programme1=Controle_opt(1:N);
    Programme2=Transfert_opt(1:N);
    % Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
    % Test fonction sinusoïdale:
    x=[0:N];      % Vecteur abscisses du temps.
    clf
    hold on
    subplot(3,1,1);
    plot(x,vol_reserve,'cyan');
    hold on
    plot(x,W2,'black')
    hold off
    title('volume et entrée d eau de la retenue secondaire, en fonction du temps')
    subplot(3,1,2);

```

```

plot(x,vol_courant,'blue'); % Graphe du volume d'eau, du barrage.
hold on
plot(x,W1,'magenta')
hold off
title('Volume et entrée d'eau de la retenue principale, en fonction du temps')
subplot(3,1,3);
plot(x,Transfert_opt,'green');
title('Transfert d'eau, en fonction du temps')
hold off
figure(2)
clf
hold on
subplot(3,1,1);
plot(x,Controle_opt,'black');
title('Contrôle envoyé à la centrale, en fonction du temps')
subplot(3,1,2);
plot(x,Production_elec,'red');
title('Production électrique, en fonction du temps')
subplot(3,1,3);
plot(x,Production_elec_cumulee,'green')
title('Production électrique cumulée, en fonction du temps')
hold off
end

```

Dans ce nouveau code, il y a très peu de modifications par rapport à la résolution précédente, l'algorithme utilisé est exactement le même, il n'y a que les entrées d'eaux qui changent.

Tout d'abord, pour choisir la fonction à utiliser pour construire les vecteurs des entrées d'eaux, on utilise des structures conditionnelles de test en *if*, pour chacun des vecteurs des entrées d'eaux *W1* et *W2*.

On teste si le paramètre *type_retenue1* ou *type_retenue2*, suivant le cas, est égal à "cruel" et, si c'est le cas, on construit le vecteur avec une boucle for itérant sur le temps *i* et l'expression analytique d'une fonction gaussienne donnée.

Si ce n'est pas le cas, on teste alors si le paramètre est égal à "périodique". Si c'est le cas, on construit le vecteur avec une autre boucle for, itérant toujours sur le temps, mais cette fois-ci avec l'expression analytique d'une fonction périodique donnée.

Si ce n'est toujours pas le cas, on construit alors le vecteur avec une valeur constante donnée.

Ensuite, pour sélectionner les valeurs des entrées *W1* et *W2* au temps *i*, on remplace partout dans le reste de la fonction les variables *W1* et *W2* par *W1(i)* et *W2(i)*.

Enfin, de la même manière que dans le cas d'un barrage à une retenue, on construit les vecteurs des valeurs de production ponctuelle et cumulée, en fonction du temps et on ajoute les graphes correspondant.

4.2.2 Quelques exemples de tests du code

On effectue quelques simulations, avec différentes données en paramètre et on observe principalement les graphes des volumes des retenues et des différents contrôles appliqués, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité à la réalité physique.

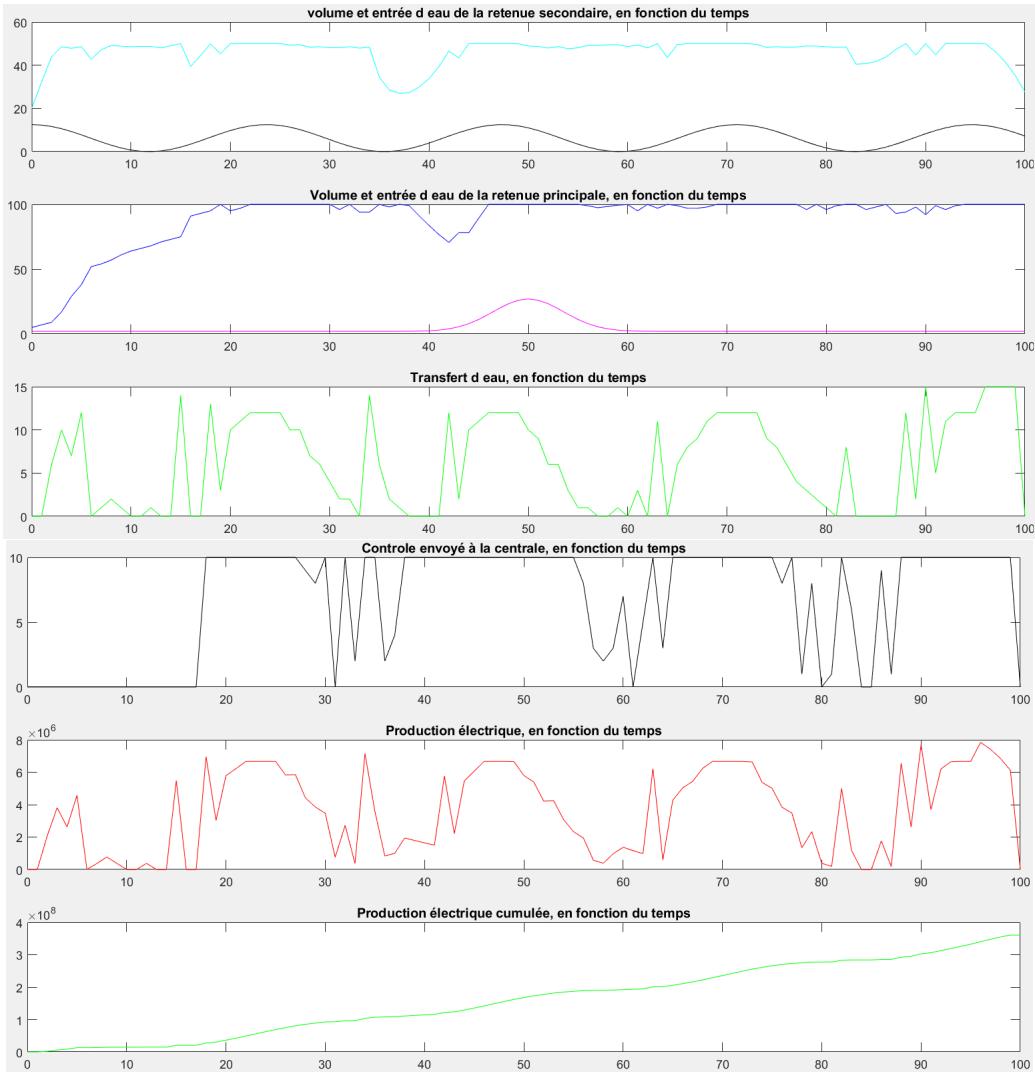
On ne peut malheureusement pas tout tester dans ce même rapport, au vu du nombre croissant de paramètres, qui peuvent influencer les résolutions et les simulations au fil des situations.
On se concentre donc ici sur l'ajout complémentaire de la nouvelle résolution.

On commence par ce premier test :

```
[production,prog1,prog2]=Optimise_Production(100,50,10,15,100,"cruel","periode",5,20)
```

Pour ce premier test, le rendement μ_1 de la retenue principale est de 0.2 et le rendement μ_2 de la retenue secondaire est de 0.8.

On obtient alors les résultats suivants :



On observe bien l'influence du rendement par la comparaison entre le graphe des transferts et celui de la production instantanée, au cours du temps. Les deux graphes ont leurs pics synchronisés et avec les mêmes amplitudes, formes et proportions.

On peut aussi observer que, non seulement la retenue secondaire anticipe la crue de son entrée d'eau en diminuant fortement son volume un peu avant, mais en plus la retenue principale anticipe aussi cette crue, car les transferts de la retenue secondaire vont alors alimenter son propre volume d'eau. Cela montre que l'algorithme anticipe les changements et adapte bien sa résolution à la nouvelle situation.

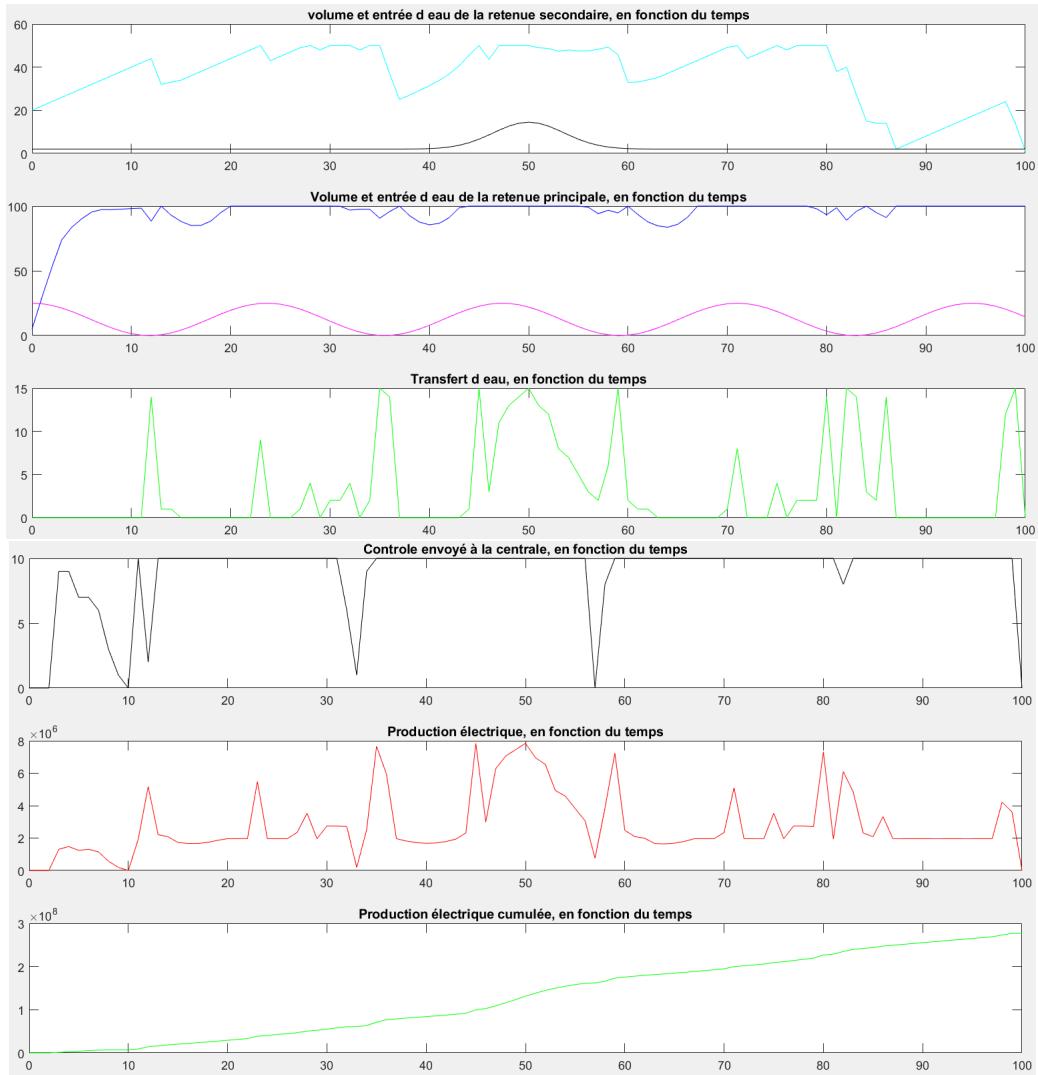
On remarque aussi que, l'entrée d'eau et le rendement étant plus forts pour la retenue secondaire, même si le volume maximal est deux fois plus faible, l'algorithme choisit de maintenir en permanence la seconde retenue à un niveau de remplissage maximal.

On effectue un dernier test, avec exactement les mêmes paramètres, mais en intervertissant cette fois les types d'entrées, pour les deux retenues :

```
[production,prog1,prog2]=Optimise_Production(100,50,10,15,100,"periodique","crue",5,20)
```

Pour ce dernier test, le rendement μ_1 de la retenue principale est de 0.2 et le rendement μ_2 de la retenue secondaire est de 0.8.

On obtient alors les résultats suivants :



Comme dans le test précédent, on observe encore que la production ponctuelle est surtout représentée par les transferts de la retenue secondaire.

On observe que l'entrée d'eau, étant beaucoup moins importante qu'avant, les transferts de la seconde retenue sont moins importants aussi.

En revanche, les contrôles de la retenue principale sont presque en permanence maximaux, car l'entrée est beaucoup plus importante et aussi parce qu'elle reçoit les transferts de la retenue secondaire, située en amont de celle-ci.

On peut également remarquer que l'algorithme choisit de vider la retenue secondaire bien avant la fin de la période de production d'électricité N . Ceci s'explique par l'importance du rendement de la retenue secondaire par rapport à celui de la retenue principale, dans cette simulation.

4.3 Résolution avec ajouts des tarifs horaires de l'électricité

Pour cette nouvelle résolution de notre problème, avec un barrage à 2 retenues, on ajoute maintenant en plus de tout le reste la prise en compte des tarifs horaires du prix de l'électricité dans la résolution de l'algorithme de programmation dynamique, sur toute la durée de production N .

4.3.1 Présentation du code et de la résolution numérique en Matlab

Voici l'ensemble des paramètres, de notre nouveau problème :

```
% L = taille maximale de la retenue principale du barrage.
% L2 = taille maximale de la retenue secondaire du barrage.
% o = taille maximale de l'ouverture des conduites forcées, de la retenue principale.
% T = taille maximale de l'ouverture du canal de transfert, de la retenue secondaire.
% N = durée totale de la période de production d'électricité.
% type_retenue1 = sélectionne le type d'entrée d'eau dans la retenue, les valeurs possibles sont "periodique" ou "crue".
% type_retenue2 = sélectionne le type d'entrée d'eau dans la retenue, les valeurs possibles sont "periodique" ou "crue".
% vol_depart = valeur arbitraire du volume présent dans le barrage au départ, pour pouvoir construire le graphe.
% vol_reserve_depart = valeur arbitraire du volume au départ, dans la seconde retenue du barrage.
% P_H_Pleine = Prix de l'électricité, en heure pleine.
% P_H_Creuse = Prix de l'électricité, en heure creuse.
% Periode_H = Période entre les changements de tarifs, heures pleines et heures creuses.
```

On rajoute simplement les trois paramètres P_H_Pleine , P_H_Creuse et $Periode_H$, exactement comme dans le cas d'un barrage à une seule retenue.

Voici le code de la fonction qui effectue la résolution et qui réalise aussi une simulation à partir des paramètres de la situation, qui sont donnés en entrée, avant de retourner tous les résultats voulus et d'afficher les graphes de la simulation :

```
function [prod_max,Program1,Program2]=Optimise_Production(L,L2,o,T,N,type_retenue1,type_retenue2,vol_depart,vol_reserve_depart,P_H_Pleine,P_H_Creuse,
% Données
rho=1000;
g=9.80665;
mu1=0.2;
mu2=0.8;
% Définition des matrices
V=zeros(N+1,L+1,L2+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1,L2+1); % Matrices des valeurs des contrôles, pour la retenue principale.
U2=zeros(N+1,L+1,L2+1); % Matrices des valeurs des transferts, entre la retenue secondaire et la retenue principale.
Volume=[0:L]; % Vecteur des valeurs des niveaux de remplissages, de la retenue principale du barrage.
Volume2=[0:L2]; % Vecteur des valeurs des niveaux de remplissages, de la retenue secondaire du barrage.
Controle=[0:o]; % Vecteur des tailles d'ouvertures de la conduite forcée.
Transfert=[0:T]; % Vecteur des valeurs possibles de transferts entre la retenue principale et secondaire.
Production_elec=ones(1,N+1); % Vecteur des valeurs de production d'électricité, en fonction du temps.
Production_elec_Prix=ones(1,N+1); % Vecteur des valeurs de production d'électricité, en prix, en fonction du temps.
Production_elec_cumulée=ones(1,N+1); % Vecteur des valeurs des gains de production cumulés, en prix, en fonction du temps.
% Construction des vecteurs des entrées d'eau:
W1=[]; % Vecteurs des entrées journalières.
W2=[];
if(type_retenue1=="crue")
    for i=0:N
        W1(i+1)=2+(L/4)*exp(-(i-(N/2))^2/(N/4));
    end
elseif(type_retenue1=="periodique")
    for i=0:N
        W1(i+1)=(L/8)+(L/8)*cos((N/2)*i);
    end
end
```

```

else
    for i=0:N
        W1(i+1)=2;
    end
end
if(type_retenue2=="crue")
    for i=0:N
        W2(i+1)=2+(L2/4)*exp(-(i-(N/2))^2/(N/4));
    end
elseif(type_retenue2=="periodique")
    for i=0:N
        W2(i+1)=(L2/8)+(L2/8)*cos((N/2)*i);
    end
else
    for i=0:N
        W2(i+1)=2;
    end
end
% Construction du vecteur des prix de l'électricité, en fonction du temps:
Prix=[]; % Vecteur des prix.
Compteur=1;
Changement=false;
for i=0:N
    if Compteur==Periode_H
        Compteur=0;
        Changement=not(Changement);
    end
    if Changement
        Prix(i+1)=P_H_Pleine;
    else
        Prix(i+1)=P_H_Creuse;
    end
    Compteur=Compteur+1;
end
% Construction des Matrices U et V, par remontée.
for i=N:-1:1 % Boucle sur le temps
    for j=1:L+1 % Boucle sur le volume du barrage
        for k=1:L2+1 % Boucle sur le volume de la retenue
            optimal2=zeros(o+1,T+1); % Contiendra le maximum par rapport aux transferts, pour chaque contrôle.
            for t=1:T+1
                if(Volume2(k)+W2(i)-Transfert(t)>=0)
                    for u=1:o+1
                        if((Volume(j)+W1(i)-Controle(u)+Transfert(t))>=0)
                            optimal2(u,t)=rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t))*Prix(i) + V(i+1,round(max(min(L,Volu
                        end
                    end
                end
                V(i,j,k)=max(max(optimal2));
                [x,y]=find(optimal2==max(max(optimal2)));
                U(i,j,k)=Controle(x(1));
                U2(i,j,k)=Transfert(y(1));
            end
        end
    end
end
[prod_max,rang]=max(V(1,:,:));
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U,U2 et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart+1,vol_reserve_depart+1);
Transfert_opt=zeros(1,N+1);
Transfert_opt(1)=U2(1,vol_depart+1,vol_reserve_depart+1);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
vol_reserve=zeros(1,N+1);
vol_reserve(1)=vol_reserve_depart;
Production_elec(1)=rho*g*(vol_courant(1)*mu1*Controle_opt(1)+vol_reserve(1)*mu2*Transfert_opt(1));
Production_elec_Prix(1)=rho*g*(vol_courant(1)*mu1*Controle_opt(1)+vol_reserve(1)*mu2*Transfert_opt(1))*Prix(1);
Production_elec_cumulee(1)=Production_elec_Prix(1);
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+W1(i)-Controle_opt(i-1)+Transfert_opt(i-1)),0);
    vol_reserve(i)=max(min(L,vol_reserve(i-1)+W2(i)-Transfert_opt(i-1)),0);
    Controle_opt(i)=U(i,round(vol_courant(i))+1,round(vol_reserve(i))+1);
    Transfert_opt(i)=U2(i,round(vol_courant(i))+1,round(vol_reserve(i))+1);
    Production_elec(i)=rho*g*(vol_courant(i)*mu1*Controle_opt(i)+vol_reserve(i)*mu2*Transfert_opt(i));
    Production_elec_Prix(i)=rho*g*(vol_courant(i)*mu1*Controle_opt(i)+vol_reserve(i)*mu2*Transfert_opt(i))*Prix(i);
    for j=1:N+1
        Production_elec_cumulee(j)=Production_elec_cumulee(j)+Production_elec_Prix(i);
    end
end
% Récupération des résultats
Program1=Controle_opt(1:N);
Program2=Transfert_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.

```

```

% Test fonction sinusoïdale:
x=[0:N]; % Vecteur abscisses du temps.
clf
hold on
subplot(4,1,1);
plot(x,vol_courant,'blue'); % Graphe du volume d'eau, du barrage.
hold on
plot(x,W1,'magenta')
hold off
title('Volume d eau de la retenue principale, en fonction du temps')
subplot(4,1,2);
plot(x,vol_reserve,'cyan');
hold on
plot(x,W2,'green')
hold off
title('volume d eau de la retenue secondaire, en fonction du temps')
subplot(4,1,3);
plot(x,Controle_opt,'red');
title('Contrôle envoyé à la centrale, en fonction du temps')
subplot(4,1,4);
plot(x,Transfert_opt,'black');
title('Transfert d eau, en fonction du temps')
hold off
figure(2)
clf
hold on
subplot(4,1,1);
plot(x,Production_elec,'red');
title('Production électrique, en fonction du temps')
subplot(4,1,2);
plot(x,Production_elec_Prix,'cyan')
title('Production électrique en prix, en fonction du temps')
subplot(4,1,3);
plot(x,Production_elec_cumulee,'black')
title('Production électrique cumulée en prix, en fonction du temps')
subplot(4,1,4);
plot(x,Prix,'green');
title('Prix de l électricité, en fonction du temps')
hold off
end

```

Ce code comporte aussi assez peu de modifications par rapport à celui de la situation précédente.

On commence par ajouter le morceau de code qui permet de générer le vecteur du prix de l'électricité, en fonction du temps, exactement de la même manière que dans le cas d'un barrage à une seule retenue.

Ensuite, dans l'algorithme de programmation dynamique, on ajoute de nouveau le prix de l'électricité en fonction du temps $P(i)$, au calcul de la fonction valeur de production instantanée $L(x, u)$ (cf partie 1.2.1).

Enfin, pour finir, on ajoute aussi dans la partie simulation de fonction la valeur du prix de l'électricité en fonction du temps $Prix(i)$ dans la construction des vecteurs des valeurs de production ponctuelles et cumulées, puis on ajoute également le graphe du prix de l'électricité, en fonction du temps.

4.3.2 Quelques exemples de tests du code

On effectue quelques simulations, avec différentes données en paramètres et on observe principalement les graphes des volumes des retenues et des différents contrôles appliquées, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité à la réalité physique.

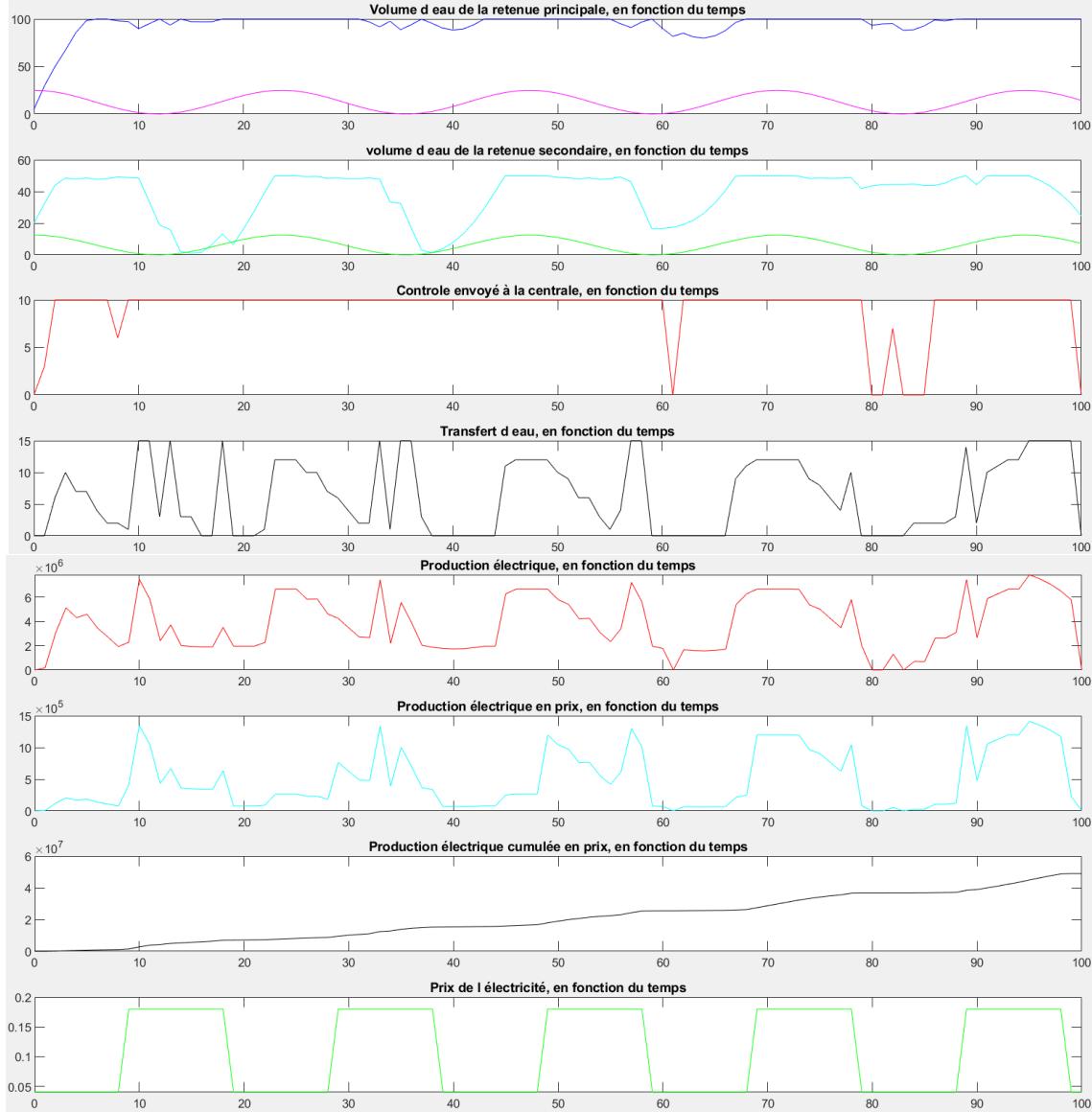
On ne peut malheureusement pas tout tester dans ce même rapport, au vu du nombre croissant de paramètres, qui peuvent influencer les résolutions et les simulations au fil des situations. On se concentre donc ici sur l'ajout complémentaire de la nouvelle résolution.

On commence par ce premier test :

```
[production,prog1,prog2]=Optimise_Production(100,50,10,15,100,"periodique","periodique",5,20,0.18,0.04,10)
```

Pour ce premier test, le rendement μ_1 de la retenue principale est de 0.2 et le rendement μ_2 de la retenue secondaire est de 0.8.

On obtient alors les résultats suivants :



Comme le rendement de la seconde retenue est beaucoup plus important que celui de la première, on observe que la production électrique ponctuelle en fonction du temps dépend très majoritairement des transferts de la seconde retenue.

Maintenant, il apparaît que ces transferts sont influencés par les entrées d'eaux, mais aussi par les tarifs horaires du prix de l'électricité. Comme le montre le graphe de la production ponctuelle en prix en fonction du temps, dont les pics sont beaucoup plus synchronisés sur les heures pleines.

On observe d'ailleurs également que ce phénomène est globalement le cas pour la régulation du volume des deux retenues. Les pics de baisses des contrôles de la retenue principale ont toujours lieu au moment des heures creuses.

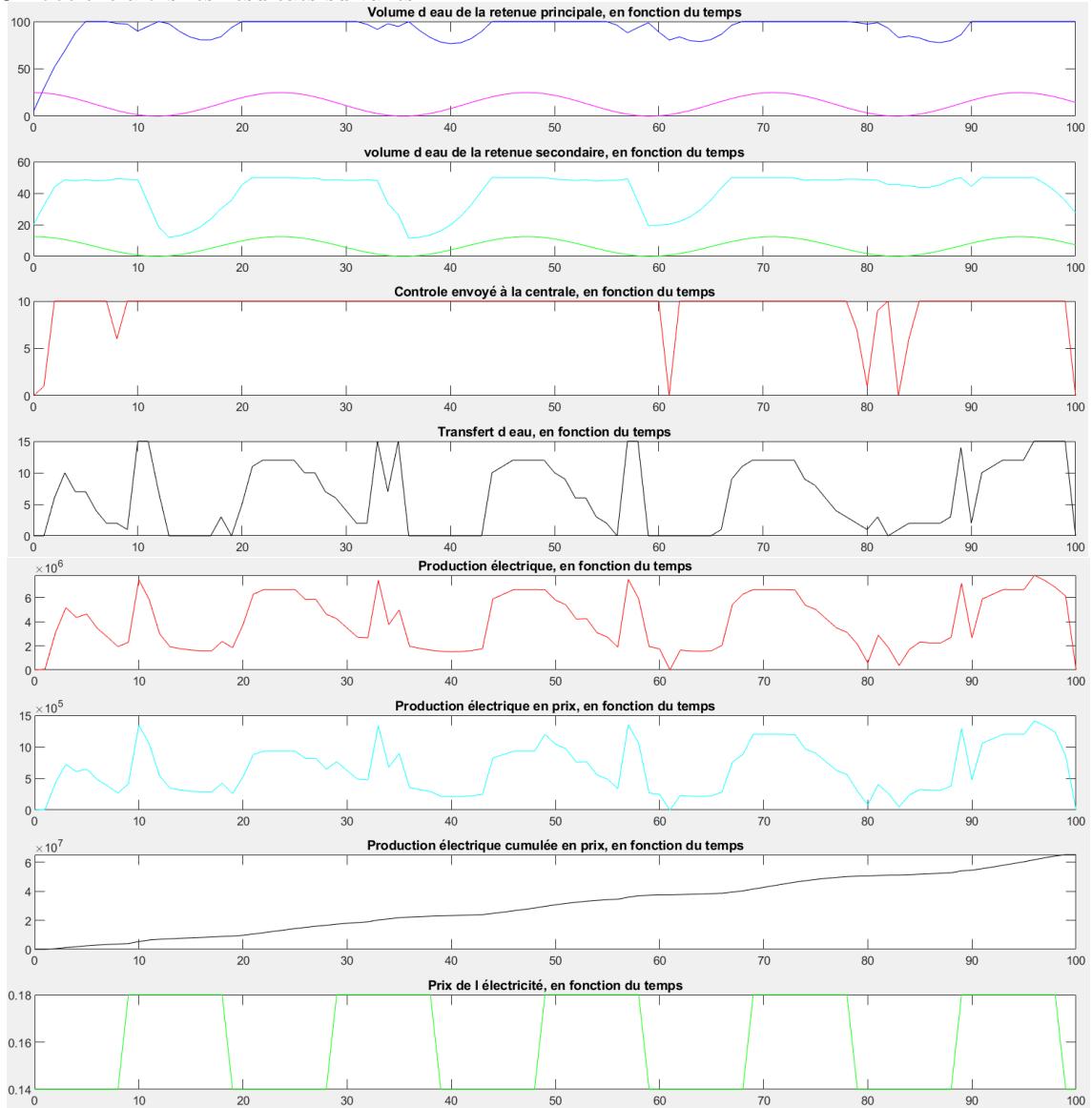
La retenue secondaire se vide pendant les heures pleines et se remplit pendant les heures creuses, en accord avec l'entrée d'eau.

On effectue maintenant un dernier test, en modifiant la différence entre les tarifs heures pleines et heures creuses du prix de l'électricité, pour mieux en voir l'influence sur la résolution de l'algorithme :

```
[production,prog1,prog2]=Optimise_Production(100,50,10,15,100,"periodique","periodique",5,20,0.18,0.14,10)
```

Pour ce dernier test, le rendement μ_1 de la retenue principale est de 0.2 et le rendement μ_2 de la retenue secondaire est de 0.8.

On obtient alors les résultats suivants :



On observe alors que les transferts de la seconde retenue semble beaucoup moins privilégier les tarifs horaires du prix de l'électricité.

La résolution tient compte principalement de l'entrée d'eau, dont les pics sont synchronisés avec les pics de celle-ci.

L'algorithme prend toujours en compte le prix de l'électricité, mais lui accorde une bien moindre importance, comme le montre les graphes des transferts et des productions ponctuelles et ponctuelles en prix. Cela montre clairement que l'algorithme adapte bien sa résolution et sa simulation, de manière cohérente, en fonction de l'évolution des tarifs, du prix de l'électricité.

Les autres aspects de la simulation sont similaires et conformes à ceux de la précédente.

4.4 Résolution en simulant les entrées d'eaux des intempéries par des chaînes de Markov

Après avoir effectué la résolution déterministe de notre nouveau problème, on s'attaque maintenant également à son point de vue probabiliste.

Pour cela, on décide de simuler chacune des deux entrées d'eaux de ces deux retenues en deux parties distinctes.

Les premières parties de ces entrées d'eaux seront notées $W1$ et $W2$ respectivement et permettront de représenter l'apport du réseau hydrographique au volume des retenues. Elles seront générées comme précédemment, à l'aide d'une fonction périodique ou gaussienne.

Les secondes parties de ces entrées d'eaux, seront notées $Entree_deau1$ et $Entree_deau2$ ou $Chaine1$ et $Chaine2$, suivant si l'on est dans l'algorithme de résolution ou bien dans la partie simulation du code. Ces parties des entrées d'eaux, représenteront les flux apportés par les intempéries.

Sous l'hypothèse que les deux retenues ne sont pas très éloignées géographiquement et aussi pour réduire la longueur du code, on utilisera la même matrice de transition, afin de simuler la pluie pour les deux retenues.

4.4.1 Présentation du code et de la résolution numérique en Matlab

Voici l'ensemble des paramètres, de notre nouveau problème :

```
% L = taille maximale de la retenue principale du barrage.  
% L2 = taille maximale de la retenue secondaire du barrage.  
% o = taille maximale de l'ouverture des conduites forcées, de la retenue principale.  
% T = taille maximale de l'ouverture du canal de transfert, de la retenue secondaire.  
% N = durée totale de la période de production d'électricité.  
% type_retenue1 = sélectionne le type d'entrée d'eau dans la retenue, les valeurs possibles sont "periodique" ou "crue".  
% type_retenue2 = sélectionne le type d'entrée d'eau dans la retenue, les valeurs possibles sont "periodique" ou "crue".  
% nbr_detats1 = Nombre de valeurs d'entrée d'eau possible, pour la pluie, sur la retenue 1.  
% nbr_detats2 = Nombre de valeurs d'entrée d'eau possible, pour la pluie, sur la retenue 2.  
% entree_deau_depart1 = valeur de l'entrée le premier jour, celle qui débute la chaîne de markov, pour la retenue 1.  
% entree_deau_depart2 = valeur de l'entrée le premier jour, celle qui débute la chaîne de markov, pour la retenue 2.  
% vol_depart = valeur arbitraire du volume présent dans le barrage au départ, pour pouvoir construire le graphe.  
% vol_reserve_depart = valeur arbitraire du volume au départ, dans la seconde retenue du barrage.  
% P_H_Pleine = Prix de l'électricité, en heure pleine.  
% P_H_Creuse = Prix de l'électricité, en heure creuse.  
% Periode_H = Période entre les changements de tarifs, heures pleines et heures creuses.
```

En plus de tous les autres paramètres déjà utilisés dans la version précédente du problème, on ajoute les 4 variables $nbr_detats1$, $nbr_detats2$, $Entree_deau1$ et $Entree_deau2$. Ceci exactement sur le modèle de la résolution analogue pour un barrage à une seule retenue, mais maintenant adapté pour deux.

On note que l'on conserve les chaînes de caractère $type_retenue1$ et $type_retenue2$, afin de représenter l'eau provenant des différents courts d'eaux qui fournissent les retenues du barrage.

Voici le code de la fonction qui effectue la résolution et qui réalise aussi une simulation à partir des paramètres de la situation, qui sont donnés en entrées, avant de retourner tous les résultats voulus et d'afficher les graphes de la simulation :

```

function [prod_max,Programme1,Programme2]=Optimise_Production(L,L2,o,T,N,type_retenue1,type_retenue2,nbr_detats1,entree_deau_depart1,nbr_detats2,entree_
% Données
rho=1000;
g=9.80665;
mu1=0.8;
mu2=0.2;
% Définition des matrices
V=zeros(N+1,L+1,L2+1,nbr_detats1+1,nbr_detats2+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1,L2+1,nbr_detats1+1,nbr_detats2+1); % Matrices des valeurs de contrôles, pour la retenue principale.
U2=zeros(N+1,L+1,L2+1,nbr_detats1+1,nbr_detats2+1); % Matrices des valeurs de transferts, entre la retenue secondaire et la retenue principale.
Volumee=0:L; % Vecteur des valeurs des niveaux de remplissages, de la retenue principale du barrage.
Volume2=0:L2; % Vecteur des valeurs des niveaux de remplissages, de la retenue secondaire du barrage.
Controle=o:o; % Vecteur des tailles d'ouvertures de la conduite forcée.
Transfert=0:T; % Vecteur des valeurs possibles de transferts entre la retenue principale et secondaire.
Production_elec=ones(1,N+1); % Vecteur des valeurs de production d'électricité, en fonction du temps.
Production_elec_Prix=ones(1,N+1); % Vecteur des valeurs de production d'électricité, en prix, en fonction du temps.
Production_elec_cumulee=ones(1,N+1); % Vecteur des valeurs des gains de production cumulés, en prix, en fonction du temps.
Entree_deau1=[0:nbr_detats1]; % Vecteur discréétisé, des différentes valeurs d'entrées d'eau possibles, dans la retenue 1.
Entree_deau2=[0:nbr_detats2]; % Vecteur discréétisé, des différentes valeurs d'entrées d'eau possibles, dans la retenue 2.
% Construction des matrices de transitions des chaînes de markov.
% matrice pour la retenue 1:
P=0.8*eye(nbr_detats1+1);
P=P+diag(0.1*ones(1,nbr_detats1),1);
P=P+diag(0.1*ones(1,nbr_detats1),-1);
P(1,1)=0.9;P(nbr_detats1+1,nbr_detats1+1)=0.9;
% matrice pour la retenue 2:
P2=0.8*eye(nbr_detats2+1);
P2=P2+diag(0.1*ones(1,nbr_detats2),1);
P2=P2+diag(0.1*ones(1,nbr_detats2),-1);
P2(1,1)=0.9;P2(nbr_detats2+1,nbr_detats2+1)=0.9;
% Construction des vecteurs des entrées d'eau des rivières:
W1=zeros(1,N+1); % Vecteurs des entrées des rivières.
W2=zeros(1,N+1);
if(type_retenue1=="crue")
    for i=0:N
        W1(i+1)=0+(L/4)*exp(-(i-(N/2))^2/(N/4));
    end
elseif(type_retenue1=="periodique")
    for i=0:N
        W1(i+1)=(L/8)+(L/8)*cos((6*N)*i);
    end
else
    for i=0:N
        W1(i+1)=0;
    end
end
if(type_retenue2=="crue")
    for i=0:N
        W2(i+1)=0+(L2/4)*exp(-(i-(N/2))^2/(N/4));
    end
elseif(type_retenue2=="periodique")
    for i=0:N
        W2(i+1)=(L2/8)+(L2/8)*cos((6*N)*i);
    end
else
    for i=0:N
        W2(i+1)=0;
    end
end
% Construction du vecteur des prix de l'électricité, en fonction du temps:
Prix=[]; % Vecteur des prix.
Compteur=1;
Changement=false;
for i=0:N
    if Compteur==Periode_H
        Compteur=0;
        Changement=not(Changement);
    end
    if Changement
        Prix(i+1)=P_H_Pleine;
    else
        Prix(i+1)=P_H_Creuse;
    end
    Compteur=Compteur+1;
end
% Construction des Matrices U et V, par remontée.
for i=N:-1:1 % Boucle sur le temps
    for j=1:L+1 % Boucle sur le volume du barrage
        for k=1:L2+1 % Boucle sur le volume de la retenue
            for l=1:nbr_detats1+1 % Boucle pour l'entrée d'eau, dans la retenue 1.
                for m=1:nbr_detats2+1 % Boucle pour l'entrée d'eau, dans la retenue 2.
                    optimal2=zeros(o+1,T+1); % Contiendra le maximum par rapport aux transferts, pour chaque contrôle.
                    for t=1:T+1
                        if(Volume2(k)+W2(i)+Entree_deau2(m)-Transfert(t)>=0)

```

```

        for u=1:o+1
            if(Volume(j)+W1(i)+Entree_deau1(1)-Controle(u)+Transfert(t)>=0)
                esperance=0;
                for a=1:nbr_detats1+1 % Seconde Boucle pour l'entrée d'eau de la retenue 1
                    for b=1:nbr_detats2+1 % Seconde Boucle pour l'entrée d'eau de la retenue 2
                        esperance=esperance+P(1,a)*P2(m,b)*V(i+1,round(max(min(L,Volume(j)+a+W1(i)-Controle(u)+Transfert(t)))
                    end
                end
                optimal2(u,t)=rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t))*Prix(i) + esperance;
            end
        end
    end
end
V(i,j,k,l,m)=max(max(optimal2));
[x,y]=find(optimal2==max(max(optimal2)));
U(i,j,k,l,m)=Controle(x(1));
U2(i,j,k,l,m)=Transfert(y(1));
end
end
end
[prod_max,rang]=max(V(1,:,:,:,:));
% Construction des chaînes de markov, à partir de P et de entree_deau_depart1 et entree_deau_depart2:
[chaîne1]=creer_chaine_markov(N+1,nbr_detats1+1,entree_deau_depart1); % On met nombre d'états +1, ccar on considère aussi l'état 0.
[chaîne2]=creer_chaine_markov(N+1,nbr_detats2+1,entree_deau_depart2); % On met nombre d'états +1, ccar on considère aussi l'état 0.
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U,U2 et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=(1,vol_depart+1,vol_reserve_depart+1,entree_deau_depart1+1,entree_deau_depart2+1);
Transfert_opt=zeros(1,N+1);
Transfert_opt(1)=U(1,vol_depart+1,vol_reserve_depart+1,entree_deau_depart1+1,entree_deau_depart2+1);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
vol_reservezeros(1,N+1);
vol_reserve(1)=vol_reserve_depart;
Production_elec(1)=rho*g*(vol_courant(1)*mu1*Controle_opt(1)+vol_reserve(1)*mu2*Transfert_opt(1));
Production_elec_Prix(1)=rho*g*(vol_courant(1)*mu1*Controle_opt(1)+vol_reserve(1)*mu2*Transfert_opt(1))*Prix(1);
Production_elec_cumulee(1)=Production_elec_Prix(1);
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+Entree_deau1(chaine1(i-1)+1)+W1(i-1)-Controle_opt(i-1)+Transfert_opt(i-1)),0);
    vol_reserve(i)=max(min(L2,vol_reserve(i-1)+Entree_deau2(chaine2(i-1)+1)+W2(i-1)-Transfert_opt(i-1)),0);
    Controle_opt(i)=U(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,chaine1(i)+1,chaine2(i)+1);
    Transfert_opt(i)=U2(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,chaine1(i)+1,chaine2(i)+1);
    Production_elec(i)=rho*g*(vol_courant(i)*mu1*Controle_opt(i)+vol_reserve(i)*mu2*Transfert_opt(i));
    Production_elec_Prix(i)=rho*g*(vol_courant(i)*mu1*Controle_opt(i)+vol_reserve(i)*mu2*Transfert_opt(i))*Prix(i);
    for j=i:N+1
        Production_elec_cumulee(j)=Production_elec_cumulee(j)+Production_elec_Prix(i);
    end
end
% Récupération des résultats
Programme1=Controle_opt(1:N);
Programme2=Transfert_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
% Test fonction sinusoïdale:
x=[0:N]; % Vecteur abscisses du temps.
clf
hold on
subplot(4,1,1);
plot(x,vol_courant,'blue'); % Groupe du volume d'eau, du barrage.
hold on
plot(x,W1,'magenta')
hold off
title('Volume d eau de la retenue principale, en fonction du temps')
subplot(4,1,2);
plot(x,vol_reserve,'cyan');
hold on
plot(x,W2,'green')
hold off
title('volume d eau de la retenue secondaire, en fonction du temps')
subplot(4,1,3);
plot(x,Controle_opt,'red');
title('Controle envoyé à la centrale, en fonction du temps')
subplot(4,1,4);
plot(x,Transfert_opt,'black');
title('Transfert d eau, en fonction du temps')
hold off
figure(2)
clf
hold on
subplot(4,1,1);
plot(x,Production_elec,'red');
title('Production électrique, en fonction du temps')
subplot(4,1,2);
plot(x,Production_elec_Prix,'blue')

```

```

title('Production électrique en prix, en fonction du temps')
subplot(4,1,3);
plot(x,Production_elec_cumulee,'black')
title('Production électrique cumulée en prix, en fonction du temps')
subplot(4,1,4);
plot(x,Prix,'green');
title('Prix de l'électricité, en fonction du temps')
hold off
figure(3)
clf
hold on
subplot(2,1,1);
plot(x,chaine1,'green');
title('entrée eau de pluie, pour la retenue 1, en fonction du temps')
subplot(2,1,2);
plot(x,chaine2,'blue')
title('entrée eau de pluie, pour la retenue 2, en fonction du temps')
hold off
end

```

Concernant le code de cette nouvelle version de notre problème, on effectue de nouveau quelques modifications du code déterministe précédent se basant principalement sur la première résolution probabiliste de ce rapport.

Pour commencer, dans la partie initialisation des matrices de résolutions et des vecteurs de discrétisations, on ajoute deux nouveaux vecteurs nommés *Entree_deau1* et *Entree_deau2* respectivement. Ils vont permettre de discrétiser les différentes valeurs des entrées d'eaux de pluies pour les deux retenues du barrage.

Toujours dans cette partie du code, on ajoute également 2 nouvelles dimensions supplémentaires aux matrices de résolutions V , U et $U2$, pour représenter les valeurs des entrées d'eaux de pluie dans les deux retenues, exactement comme dans le cas à une seule retenue.

Ensuite, conformément à ce qui a été expliqué plus haut, on génère les deux matrices de transition P et $P2$, avec le même code pour générer la même matrice, mais écrit deux fois successivement, afin de pouvoir modifier plus facilement si l'on ne souhaite plus que les deux chaînes soient générées à partir de la même matrice de transition.

Les structures conditionnelles permettant de construire les vecteurs représentant la part de l'entrée d'eau liées aux rivières ne sont pas modifiés.

Concernant les modifications de l'algorithme, on rajoute 4 nouvelles boucles for. Il y en a 2 pour les valeurs des entrées d'eaux de pluies de chaque retenue du barrage.

Les deux premières boucles ajoutées permettent d'itérer sur les dimensions des matrices de résolutions, représentant les différents niveaux d'entrées d'eaux possibles pour les deux retenues. Ceci afin de ranger les valeurs de production optimales aux bons emplacements dans les matrices V , U et $U2$.

Les deux dernières boucles for permettent aussi d'itérer sur les différents niveaux d'entrées d'eaux, mais cette fois-ci pour calculer l'espérance de production pour chacun des volumes et chacune des entrées d'eaux, conformément à la formule de la partie 3.5.2 de ce rapport.

Dans la partie simulation, on génère aléatoirement les deux chaînes de Markov imitant la pluie, avec la fonction *cree_chaine_markov*, présentée précédemment dans la partie 3.5.1 de ce rapport.

Ensuite, on rajoute les entrées d'eaux des chaînes de Markov pour ajouter la pluie, et on ajoute aussi les coefficients des nouvelles dimensions des matrices de résolutions dans la construction des vecteurs des graphes.

Enfin, on ajoute les graphes des deux entrées d'eaux de pluies, simulés par des chaînes de Markov.

4.4.2 Quelques exemples de tests du code

On effectue quelques simulations, avec différentes données en paramètre et on observe principalement les graphes des volumes des retenues et des différents contrôles appliquées, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité à la réalité physique.

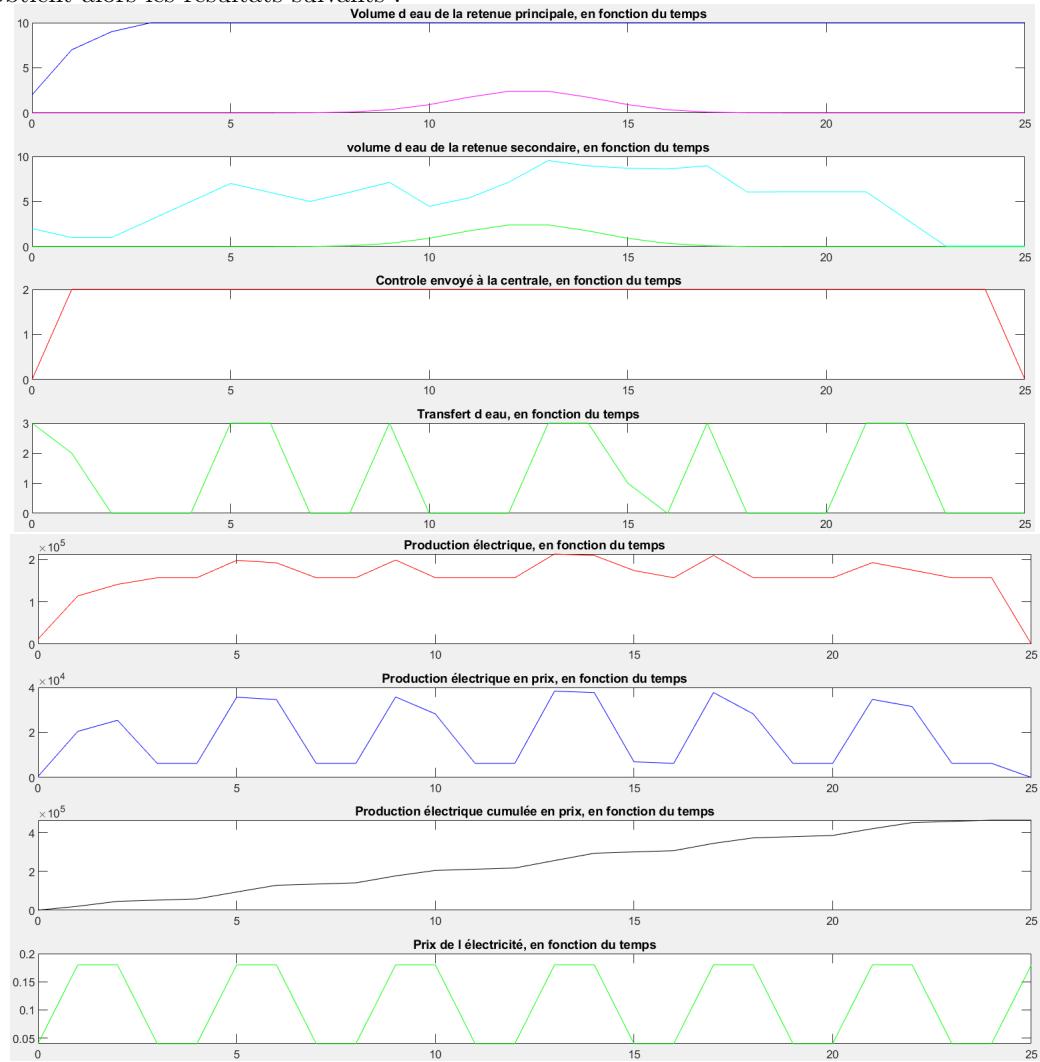
On ne peut malheureusement pas tout tester dans ce même rapport, au vu du nombre croissant de paramètres, qui peuvent influencer les résolutions et les simulations au fil des situations.
On se concentre donc ici sur l'ajout complémentaire de la nouvelle résolution.

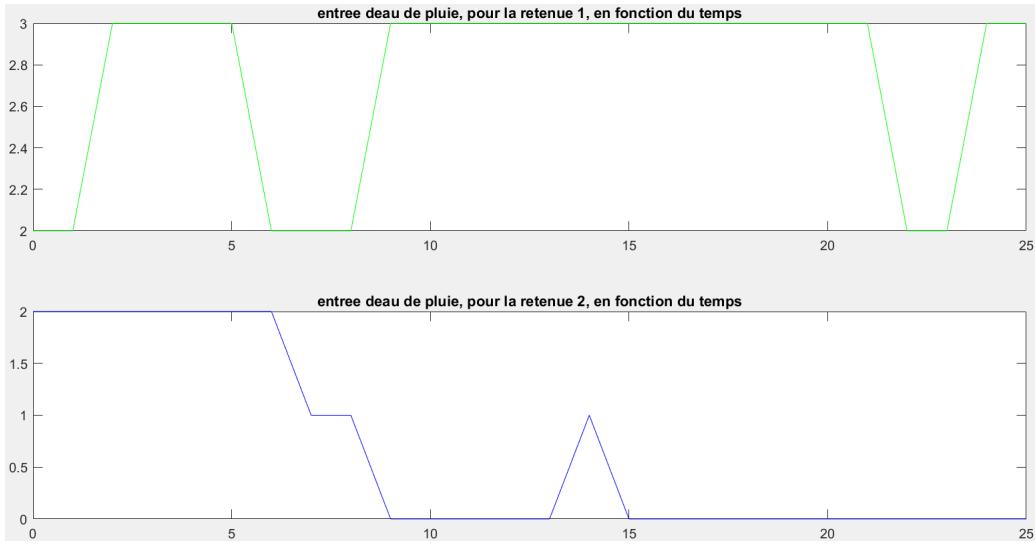
On commence par ce premier test :

```
[production,prog1,prog2]=Optimise_Production(10,10,2,3,25,"crue","crue",3,2,3,2,2,2,0.18,0.04,2)
```

Pour ce premier test, le rendement μ_1 de la retenue principale est de 0.2 et le rendement μ_2 de la retenue secondaire est de 0.8.

On obtient alors les résultats suivants :





Avec ce nouveau code, le temps de calcul a été fortement rallongé, principalement à cause de l'augmentation du nombre de dimensions des matrices de résolutions, ce qui fait également augmenté le nombre de boucle for de l'algorithme.

Sans oublier le calcul de l'espérance de production pour chaque volumes présents dans les retenues, pour chaque entrées d'eaux et aussi pour chaque paire de contrôles applicables.

En raison de cette augmentation du temps de compilation du code, on a réduit la durée de la période de production N par rapport aux tests des versions précédentes du code et on a également réduit les tailles des retenues et des contrôles maximaux possibles, afin de limiter le nombre de tour de boucles for.

On observe dans ce premier test que l'entrée d'eau des rivières, étant très faible sur la majorité du temps, l'algorithme privilégie alors les tarifs du prix de l'électricité pour choisir les paires de contrôles à appliquer. On peut le voir sur les graphes du prix de l'électricité, en fonction du temps, celui de la production ponctuelle en prix et sur celui des transferts appliqués à la seconde retenue.

On peut noter que les entrées d'eaux sont maintenant plus difficiles à observer, car elles proviennent maintenant de deux sources simulées différemment les intempéries et les rivières.

Pour la retenue principale, on observe qu'elle se remplit rapidement et ne se vide jamais, car son volume de 10 et sa capacité maximale de contrôles de 2 sont relativement faibles.

Elle reçoit non seulement les transferts de la retenue secondaire, mais en plus l'entrée d'eau des rivières et l'entrée d'eau des intempéries, simulée par la chaîne de Markov, qui suffirait déjà à elle seule pour la faire déborder.

Pour la retenue secondaire, c'est plus intéressant. La retenue met au début du temps à se remplir entièrement, car elle aide la retenue principale à se remplir aussi.

Ensuite, on observe que son entrée d'eau des intempéries s'effondre au milieu du temps et que son entrée d'eau des rivières ne tarde pas à également redescendre, son entrée d'eau s'annule.

On observe que l'algorithme décide alors de vider progressivement la retenue, au profit de la première retenue, mais toujours en heures pleines, afin de maximiser la production.

Le rendement de la seconde retenue, étant toujours très important par rapport à celui de la retenue principale, on observe que les pics de transferts de la retenue secondaire apparaissent dans le graphe de la production ponctuelle, en prix, en fonction du temps.

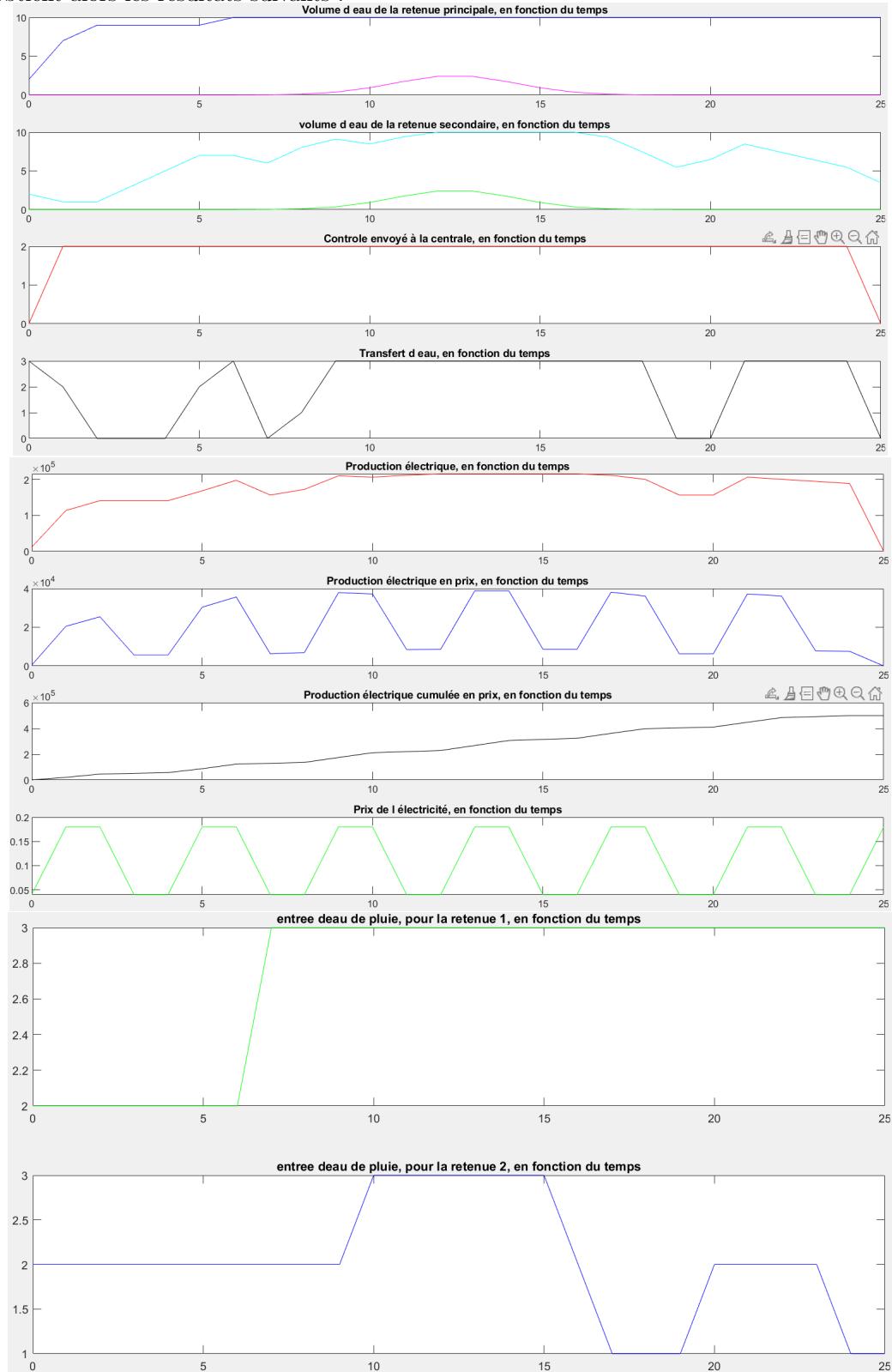
Ce premier test semble montrer que l'algorithme effectue bien la résolution souhaitée.

On effectue maintenant un dernier test, pour obtenir une simulation avec d'autres chaînes aléatoire de

Markov simulant la pluie, pour mieux en voir l'influence sur la résolution de l'algorithme :
`[production,prog1,prog2]=Optimise_Production(100,50,10,15,100,"periodique","periodique",5,20,0.18,0.14,10)`

Pour ce dernier test, le rendement μ_1 de la retenue principale est de 0.2 et le rendement μ_2 de la retenue secondaire est de 0.8.

On obtient alors les résultats suivants :



Ce nouveau test donne des chaînes de Markov aléatoires, qui sont très similaires à celles du test précédent sur la première partie du temps.

On observe alors que l'algorithme fait des choix similaires et cohérents par rapport à ceux qu'il a fait précédemment.

Ensuite, dans la seconde partie du temps, l'entrée d'eau de pluie de la retenue secondaires ne s'annule pas, mais varie un peu et l'entrée d'eau de pluie de la retenue principale reste bloquée à 3.

On remarque l'algorithme fait alors des choix légèrement différents des précédents, mais toujours cohérents. Ils adaptent ainsi sa résolution à l'évolution de la situation.

Concernant les tarifs du prix de l'électricité, les contrôles et les transferts appliqués en fonction du temps, les observations du premier test s'appliquent toujours dans celui-ci.

5 Résolution pour un barrage à 3 retenues

On passe maintenant à la résolution de notre problème d'optimisation de la production électrique, pour un barrage comportant 3 retenues d'eaux différentes.

5.1 Résolution avec des entrées d'eaux constantes

Pour cette première résolution de notre problème, avec un barrage à 3 retenues, on se donne maintenant trois valeurs d'entrées d'eaux journalières W_1 , W_2 et W_3 constantes sur toute la durée de production N .

5.1.1 Présentation du code et de la résolution numérique en Matlab

Voici l'ensemble des paramètres, de notre nouveau problème :

```
% L = taille maximale de la retenue principale du barrage.  
% L2 = taille maximale de la retenue secondaire du barrage.  
% L3 = taille maximale de la retenue tertiaire du barrage.  
% o = taille maximale de l'ouverture des conduites forcées, de la retenue principale.  
% T = taille maximale de l'ouverture du canal de transfert, de la retenue secondaire.  
% S = taille maximale de l'ouverture du canal des apports, de la retenue tertiaire.  
% N = durée totale de la période de production d'électricité.  
% W1 = entrée d'eau journalière, dans la retenue principale.  
% W2 = entrée d'eau journalière, dans la retenue secondaire.  
% W3 = entrée d'eau journalière, dans la retenue tertiaire.  
% type_graphe = sélectionne le type de liaison entre les 3 retenues, les valeurs possibles sont "arbre" ou "serie".  
% vol_depart = valeur arbitraire du volume présent dans le barrage au départ, pour pouvoir construire le graphe.  
% vol_reserve_depart = valeur arbitraire du volume au départ, dans la seconde retenue du barrage.  
% vol_reserve2_depart = valeur arbitraire du volume au départ, dans la troisième retenue du barrage.
```

Pour résoudre ce nouveau problème, on rajoute les variables L_3 et W_3 , qui donneront respectivement la taille et l'entrée d'eau dans la troisième retenue.

On ajoute également la variable S , qui représentera la valeur maximale de contrôles applicables au volume d'eau contenu dans la troisième retenue du barrage.

Pour différencier ces contrôles de ceux de la retenue principale et des transferts de la retenue secondaire, on les appellera les "apports".

On ajoute aussi un volume de départ dans la troisième retenue, pour la simulation avec la variable *vol_reserve_depart*.

Enfin, on ajoute une nouvelle variable nommée "*type_graphe*". Ce sera une chaîne de caractère, pouvant prendre soit la valeur "arbre" ou bien "serie". Elle permettra de choisir l'organisation ou plutôt le graphe de la hiérarchisation des retenues du barrages entre elles.

Avec "serie", la troisième retenue envoie ses apports à la seconde retenue, qui envoie elle-même ses transferts à la première retenue.

Avec "arbre", la troisième et la seconde retenue envoient toutes deux l'ensemble de leurs contrôles à la retenue principale du barrage.

Voici le code de la fonction qui effectue la résolution basée sur la nouvelle adaptation de l'algorithme de programmation dynamique et qui réalise aussi une simulation à partir des paramètres de la situation qui sont donnés en entrée, avant de retourner tous les résultats voulus et d'afficher les graphes de la simulation :

```

function [prod_max,Programme1,Programme2,Programme3]=Optimise_Production(L,L2,L3,o,T,S,N,W1,W2,W3,type_graphe,vol_depart,vol_reserve_depart,vol_reserve2,
% Données
rho=1000;
g=9.80665;
mu1=0;
mu2=0;
mu3=0.8;
%x=1;
%y=1;
%z=1;
% Définition des matrices
V=zeros(N+1,L+1,L+2+1,L3+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1,L+2+1,L3+1); % Matrices des valeurs des contrôles, pour la retenue principale.
U2=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs des transferts, de la retenue secondaire.
U3=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs des transferts, de la retenue tertiaire.
Volume=[0:L]; % Vecteur des valeurs des niveaux de remplissages, de la retenue principale du barrage.
Volume2=[0:L2]; % Vecteur des valeurs des niveaux de remplissages, de la retenue secondaire du barrage.
Volume3=[0:L3]; % Vecteur des valeurs des niveaux de remplissages, de la retenue tertiaire du barrage.
Controle=[0:o]; % Vecteur des tailles d'ouvertures de la conduite forcée.
Transfert=[0:T]; % Vecteur des valeurs possibles de transferts de la retenue secondaire.
Apport=[0:S]; % Vecteur des valeurs possibles de transferts de la retenue tertiaire.
if(type_graphe=="serie")
    % Construction des Matrices U et V, par remontée.
    for i=N:-1:1 % Boucle sur le temps
        for j=1:L+1 % Boucle sur le volume du barrage
            for k=1:L2+1 % Boucle sur le volume de la retenue secondaire
                for l=1:L3+1 % Boucle sur le volume de la retenue tertiaire
                    optimal2=0; % Contiendra le maximum par rapport aux transferts et apports, pour chaque contrôle.
                    for s=1:S+1
                        if(Volume3(l)+W3-Apport(s)>=0)
                            for t=1:T+1
                                if(Volume2(k)+W2+Apport(s)-Transfert(t)>=0)
                                    for u=1:+1
                                        if((Volume(j)+W1-Controle(u)+Transfert(t)>=0))
                                            if(optimal2<=rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(l)*mu3*Apport(s))+V
                                                optimal2=rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(l)*mu3*Apport(s))+V
                                                x=u;
                                                y=t;
                                                z=s;
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
    V(i,j,k,l)=optimal2;
    U(i,j,k,l)=Controle(x);
    U2(i,j,k,l)=Transfert(y);
    U3(i,j,k,l)=Apport(z);
end
end
end
[prod_max,rang]=max(V(1,:,:));
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U,U2 et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
Transfert_opt=zeros(1,N+1);
Transfert_opt(1)=U2(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
Apport_opt=zeros(1,N+1);
Apport_opt(1)=U3(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
vol_reserve=zeros(1,N+1);
vol_reserve(1)=vol_reserve_depart;
vol_reserve2=zeros(1,N+1);
vol_reserve2(1)=vol_reserve2_depart;
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+W1-Controle_opt(i-1)+Transfert_opt(i-1)),0);
    vol_reserve(i)=max(min(L2,vol_reserve(i-1)+W2+Apport_opt(i-1)-Transfert_opt(i-1)),0);
    vol_reserve2(i)=max(min(L3,vol_reserve2(i-1)+W3-Apport_opt(i-1)),0);
    Controle_opt(i)=U(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
    Transfert_opt(i)=U2(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
    Apport_opt(i)=U3(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
end
elseif(type_graphe=="arbre")
    % Construction des Matrices U et V, par remontée.
    for i=N:-1:1 % Boucle sur le temps
        for j=1:L+1 % Boucle sur le volume du barrage
            for k=1:L2+1 % Boucle sur le volume de la retenue secondaire
                for l=1:L3+1 % Boucle sur le volume de la retenue tertiaire
                    optimal2=0; % Contiendra le maximum par rapport aux transferts et apports, pour chaque contrôle.
                    for s=1:S+1

```

```

if(Volume3(1)+W3-Apport(s)>=0)
    for t=1:T+1
        if(Volume2(k)+W2-Transfert(t)>=0)
            for u=1:o+1
                if((Volume(j)+W1-Controle(u)+Transfert(t)+Apport(s)>=0))
                    if(optimal2<rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(1)*mu3*Apport(s))+x=u;
                    optimal2=rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(1)*mu3*Apport(s))+V(i+
                    x=u;
                    y=t;
                    z=s;
                    end
                end
            end
        end
    end
    V(i,j,k,l)=optimal2;
    U(i,j,k,l)=Controle(x);
    U2(i,j,k,l)=Transfert(y);
    U3(i,j,k,l)=Apport(z);
end
end
end
[prod_max,rang]=max(V(1,:,:));
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U,U2 et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
Transfert_opt=zeros(1,N+1);
Transfert_opt(1)=U2(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
Apport_opt=zeros(1,N+1);
Apport_opt(1)=U3(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
vol_reserve=zeros(1,N+1);
vol_reserve(1)=vol_reserve_depart;
vol_reserve2=zeros(1,N+1);
vol_reserve2(1)=vol_reserve2_depart;
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+W1-Controle_opt(i-1)+Transfert_opt(i-1)+Apport_opt(i-1)),0);
    vol_reserve(i)=max(min(L2,vol_reserve(i-1)+W2-Transfert_opt(i-1)),0);
    vol_reserve2(i)=max(min(L3,vol_reserve2(i-1)+W3-Apport_opt(i-1)),0);
    Controle_opt(i)=U(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
    Transfert_opt(i)=U2(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
    Apport_opt(i)=U3(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
end
% Récupération des résultats
Programme1=Controle_opt(1:N);
Programme2=Transfert_opt(1:N);
Programme3=Apport_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
% Test fonction sinusoïdale:
x=[0:N]; % Vecteur abscisses du temps.
clf
hold on
subplot(3,1,1);
plot(x,vol_reserve2,'green')
title('volume d eau de la retenue tertiaire, en fonction du temps')
subplot(3,1,2);
plot(x,vol_reserve,'cyan');
title('volume d eau de la retenue secondaire, en fonction du temps')
subplot(3,1,3);
plot(x,vol_courant,'magenta'); % Graphe du volume d'eau, du barrage.
title('Volume d eau de la retenue principale, en fonction du temps')
hold off
figure(2)
clf
hold on
subplot(3,1,1);
plot(x,Apport_opt,'black');
title('Transfert d eau depuis la retenue 3, en fonction du temps')
subplot(3,1,2);
plot(x,Transfert_opt,'blue');
title('Transfert d eau depuis la retenue 2, en fonction du temps')
subplot(3,1,3);
plot(x,Controle_opt,'red');
title('Controle envoyé à la centrale, en fonction du temps')
hold off
end

```

Dans ce nouveau code, on effectue des modifications similaires à celles qui ont été effectuées pour passer d'un barrage de 1 à 2 retenues.

On commence notamment par ajouter la variable μ_3 dans les données, elle représente le rendement de la troisième retenue du barrage.

On ajoute de même, une nouvelle dimension supplémentaire à chacune des matrices de résolutions V, U

et $U2$, pour stocker les valeurs de production et des contrôles maximaux, également par rapport au volume de la troisième retenue.

On ajoute aussi une nouvelle matrice de résolution nommée $U3$, de la même taille que les autres et qui permettra de stocker les valeurs des contrôles appliqués au volume de la troisième retenue du barrage.

Ensuite, on crée deux nouveaux vecteurs de discrétilisations, nommés respectivement $Volume3$ et $Apport$. $Volume3$ permettra de discrétiliser les valeurs des différents niveaux de remplissage de la troisième retenue, et $Apport$ permettra de discrétiliser les valeurs des différents contrôles applicables au volume de la troisième retenue.

Maintenant, pour connaître l'organisation entre les retenues du barrages, on utilise une structure conditionnelle *if* et *elsif*, dans laquelle le prédicat vérifie la valeur de la chaîne de caractère *type_graphe* donnée en paramètre de notre fonction *Optimise_production*.

Dans chacune des deux parties du *if*, on a une version différente de l'algorithme de programmation dynamique, qui adapte sa résolution du problème à trois retenues pour chacun des deux types d'organisations entre les retenues.

Le principal changement intervient pour la mise à jour des volumes des trois retenues, dans l'appel récursif à la matrice de résolution V de l'algorithme de programmation dynamique.(cf partie 1.2.3)

Dans les deux versions de l'algorithme, on ajoute deux nouvelles boucles for.
La première permet d'itérer sur les valeurs du volume de la troisième retenue, afin de pouvoir stocker les valeurs de résolutions optimales dans les bonnes cases des matrices de résolutions V , U , $U2$ et $U3$.
La seconde boucle for permet d'itérer sur les valeurs des contrôles applicables au volume de la troisième retenue, afin de tester les valeurs de production, pour tous les triplets de contrôles possibles.

Avec ces nouvelles boucles for, on ajoute également deux nouveaux tests *if* qui vont permettre de vérifier suivant la situation, que ces triplets de contrôles en question sont bien physiquement applicables aux volumes des trois volumes du barrage.

En revanche, on ne peut plus simplement stocker dans une matrice nommée *optimal2*, dimensionnée sur toutes les valeurs de contrôles possibles pour les deux retenues.
Puis juste récupérer les valeurs du premier maximum de l'ensemble des solutions optimales, à l'aide des commandes Matlab *max* et *find*, car il se trouve que pour trois dimensions de contrôles possibles, ça ne fonctionne plus.

Pour contourner ce problème, on décide de simplement créer une variable nommée *optimal2* et de lui stocker le maximum entre la valeur de production, issue du triplets de contrôles précédents et celle issue du triplet de contrôles itérés actuellement.
On récupère ce maximum, en utilisant une structure conditionnelle *if*, dont le prédicat compare les deux valeurs. On récupère également les trois valeurs de contrôles associées.

Comme *Optimal2* est initialisée à 0, on lui affecte obligatoirement la valeur issue du premier triplet de contrôles itéré.

Mais puisque d'après l'ordre croissant d'itération des valeurs de contrôles, il s'agit du triplet de contrôles nulles, la valeur de production associée est également nulle, donc cela ne change rien.

Ensuite, dans les itérations suivantes, on calcule la nouvelle la valeur de production du nouveau triplet de contrôles itéré. On la compare, à celle du triplet précédents et on écrase l'ancienne si elle est supérieure ou égale.

Comme on écrase les anciennes valeurs de contrôles optimaux, même si la valeur de production est égale, on peut en déduire que les valeurs de contrôles optimaux que l'on va obtenir à la fin des itérations, seront celles qui correspondent à la dernière combinaisons de valeurs de contrôles optimaux, dans l'ensemble de ces combinaisons.

Puisque les itérations sur les valeurs de ces contrôles se font de manière croissante, on en déduit que l'on obtiendra les valeurs de contrôles optimaux les plus grandes possibles, dans l'ensemble des combinaisons optimales.

Dans la version précédente de l'algorithme, on récupérait le premier triplet de contrôles optimaux, dans l'ordre croissant, donc les mêmes de ces contrôles étaient minimales, parmi l'ensemble des triplets optimaux.

On pourra plus tard observer cela sur les graphes de cette version de la résolution du problème, et aussi sur les graphes de la version suivantes. (cf Partie 5.2.2)

Si l'on souhaitait récupérer de nouveau le premier triplet de valeurs contrôles optimaux, on aurait juste à mettre une inégalité stricte dans le prédictat qui compare la valeur de production précédente et la nouvelle dans l'algorithme.

De même, dans la version précédente du code, si l'on souhaitait récupérer les valeurs du derniers couples de contrôles optimaux, on aurait seulement à remplacer, 1 par $length(x)$, dans la récupération des contrôles associées à la valeur du maximum de production.(cf partie 4.4.1)

Dans la partie simulation, on construit les anciens et les nouveaux vecteurs des graphes, exactement comme dans les codes précédents, en adaptant bien sur la mise à jour des volumes des retenues au type de graphes *type_graphe*, correspondant à la copie du code et en ajoutant les coefficients d'appels aux nouvelles dimensions des matrices de résolutions.

Pour finir, on construit les graphes sur plusieurs fenêtres, avec la commande Matlab *subplot*, comme précédemment.

5.1.2 Quelques exemples de tests du code

On effectue quelques simulations, avec différentes données en paramètres et on observe principalement les graphes des volumes des retenues et des différents contrôles appliquées, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité à la réalité physique.

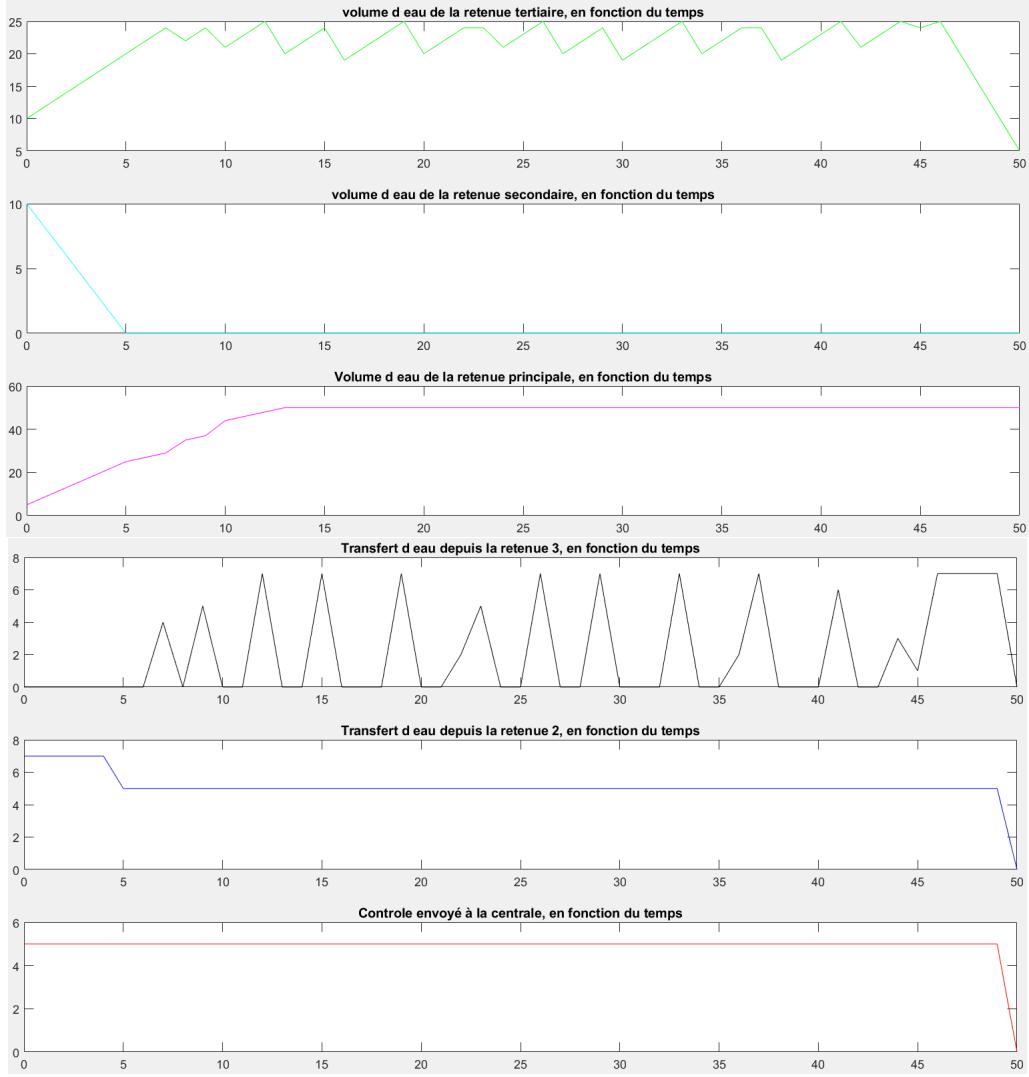
On ne peut malheureusement pas tout tester dans ce même rapport, au vu du nombre croissant de paramètres qui peuvent influencer les résolutions et les simulations, au fil des situations.
De plus, avec ce nouveau code, le temps de calcul a été encore très fortement rallongé, principalement à cause de l'augmentation du nombre de dimensions des matrices de résolutions, ce qui fait également augmenter le nombre de boucle for de l'algorithme.

On commence par ce premier test :

```
[production,prog1,prog2,prog3]=Optimise_Production(50,25,25,5,7,7,50,2,5,2,"arbre",5,10,10)
```

Pour ce test, le rendement μ_1 de la retenue principale est de 0, le rendement μ_2 de la retenue secondaire est de 0 et le rendement μ_3 de la troisième retenue est 0.8.

Après un temps de compilation d'environ 1m30s, on obtient alors les résultats suivants :



On observe que l'algorithme fait bien en sorte de maintenir la troisième retenue à un volume d'eau maximal, tout en exerçant des contrôles ponctuels les plus importants possibles, en accord avec l'entrée d'eaux constante $W3$.

La retenue principale reçoit les contrôles des deux autres retenues car le type de graphe choisi est "*arbre*". La retenue principale se remplit rapidement et déborde, jusqu'à la fin de la période de production N .

On peut également observer que, malgré des rendements μ_1 et μ_2 nuls, les retenues 1 et 2 appliquent quand même des contrôles non nuls à leurs volumes respectifs.

Ceci se justifie, par le fait que l'algorithme choisit de récupérer le dernier triplet de contrôles associé à la valeur de production optimale, pour chaque situation de volume et pour tous les temps, comme expliqué plus haut.

5.2 Résolution matricielle

Pour cette première résolution de notre problème avec un barrage à 3 retenues, on se donne toujours trois valeurs d'entrées d'eaux journalières W_1 , W_2 et W_3 constantes, sur toute la durée de production N .

On a souhaité écrire la mise à jour des volumes des retenues sous forme matricielle dans l'équation de programmation dynamique, afin de simplifier l'algorithme et de réduire la taille du code.

On obtient malheureusement, pour cette version de la résolution de notre problème, un temps de compilation environ 4 fois plus long que celui du code précédent, et ceci malgré un code deux fois moins long et bien plus simple à lire.

On a vu au cours des TD que l'on pouvait associer une matrice à chaque graphe de l'organisation des retenues entre elles.[5]

Voici la matrice correspondant au graphe de type "*Série*" :

$$\begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{pmatrix}$$

Voici la matrice correspondant au graphe de type "*Arbre*" :

$$\begin{pmatrix} -1 & 1 & 1 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

On les obtient par le raisonnement suivant.

Le nombre de dimensions de la matrice correspond au nombre de retenues du barrage. On a une dimension pour chaque retenue dans l'ordre croissant.

Les lignes représentent les retenues dans l'ordre croissant et les colonnes représente le contrôle associé au volume de chaque retenue.

Par exemple, pour "*Arbre*", la retenue 1 reçoit les contrôles issues de la retenue 2 et 3, donc on met des 1 sur les coefficients des colonnes 2 et 3 de la ligne 1.

On met un 1 quand la retenue de la ligne correspondante reçoit l'eau du contrôle de la colonne correspondante.

Toujours pour "*Arbre*", la retenue 2 en ligne 2 perd l'eau de son contrôle en colonne 2, au profit de la retenue en ligne 1. On place donc un -1 , à la colonne 2 de la ligne 2.

On fait le même raisonnement pour la ligne 3. Pour la ligne 1, l'eau du contrôle est envoyée à la centrale hydraulique, elle sort donc définitivement du circuit du barrage.

À l'aide d'une des matrices de graphe précédentes, on applique la formule suivante pour mettre à jour les volumes des trois retenues à chaque nouveau temps i :

$$New_Vol = Matrice * Instruction + Entree + Volume$$

Où New_Vol contient les valeurs des trois volumes, mise à jour au nouveau temps i .

$Instruction$ contient les valeurs de chacun des trois contrôles, appliquées à chaque retenue.

$Entree$ contient les valeurs des entrées d'eaux journalières constantes, ajoutées aux volume de chaque retenue du barrage.

$Volume$ contient les valeurs des volumes des trois retenues, avant la mise à jour.

5.2.1 Présentation du code et de la résolution numérique en Matlab

Voici l'ensemble des paramètres, de notre nouveau problème :

```
% L = taille maximale de la retenue principale du barrage.
% L2 = taille maximale de la retenue secondaire du barrage.
% L3 = taille maximale de la retenue tertiaire du barrage.
% o = taille maximale de l'ouverture des conduites forcées, de la retenue principale.
% T = taille maximale de l'ouverture du canal de transfert, de la retenue secondaire.
% S = taille maximale de l'ouverture du canal des apports, de la retenue tertiaire.
% N = durée totale de la période de production d'électricité.
% W1 = entrée d'eau journalière, dans la retenue principale.
% W2 = entrée d'eau journalière, dans la retenue secondaire.
% W3 = entrée d'eau journalière, dans la retenue tertiaire.
% type_graphe = sélectionne le type de liaison entre les 3 retenues, les valeurs possibles sont "arbre" ou "serie".
% vol_depart = valeur arbitraire du volume présent dans le barrage au départ, pour pouvoir construire le graphe.
% vol_reserve_depart = valeur arbitraire du volume au départ, dans la seconde retenue du barrage.
% vol_reserve2_depart = valeur arbitraire du volume au départ, dans la troisième retenue du barrage.
```

On résout exactement le même problème que dans la partie précédente et avec les mêmes données, donc les paramètres de la résolution et de la simulation ne changent pas.

Voici le code de la fonction qui effectue la résolution basée sur la nouvelle adaptation matricielle de l'algorithme de programmation dynamique et qui réalise aussi une simulation à partir des paramètres de la situation qui sont données en entrée avant de retourner tous les résultats voulus et d'afficher les graphes de la simulation :

```
function [prod_max,Programme1,Programme2,Programme3]=Optimise_Production(L,L2,L3,o,T,S,N,W1,W2,W3,type_graphe,vol_depart,vol_reserve_depart,vol_reserve2,
% Données
rho=1000;
g=9.80665;
mu1=0;
mu2=0;
mu3=0.8;
%x=1;
%y=1;
%z=1;
% Définition des matrices
V=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs des contrôles, pour la retenue principale.
U2=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs des transferts, de la retenue secondaire.
U3=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs des transferts, de la retenue tertiaire.
Volume1=[0:L]; % Vecteur des valeurs des niveaux de remplissages, de la retenue principale du barrage.
Volume2=[0:L2]; % Vecteur des valeurs des niveaux de remplissages, de la retenue secondaire du barrage.
Volume3=[0:L3]; % Vecteur des valeurs des niveaux de remplissages, de la retenue tertiaire du barrage.
Controle=[0:o];
Transfert=[0:T];
Apport=[0:S];
Entree=[W1;W2;W3];
Volume=zeros(3,1);
Instruction=zeros(3,1);
% Création des matrices Arbre et Serie:
Serie=eye(3);
Serie=Serie+diag(ones(1,2),1);
Arbre=eye(3);
Arbre(1,2)=1;
Arbre(1,3)=1;
% Choix de l'organisation des retenues:
if(type_graphe=="serie")
    Matrice=Serie;
elseif(type_graphe=="arbre")
    Matrice=Arbre;
end
% Construction des Matrices U et V, par remontée.
for i=N:-1:1 % Boucle sur le temps
    for j=1:L+1 % Boucle sur le volume du barrage
        Volume1(1)=Volume1(j);
        for k=1:L2+1 % Boucle sur le volume de la retenue secondaire
            Volume2(k)=Volume2(k);
            for l=1:L3+1 % Boucle sur le volume de la retenue tertiaire
                Volume3(l)=Volume3(l);
                optimal12=0; % Contiendra le maximum par rapport aux transferts et apports, pour chaque contrôle.
                for s=1:S+1
                    if(Volume3(l)+W3-Apport(s)>=0)
                        Instruction(3)=Apport(s);
                        for t=1:T+1
                            if(Volume2(k)+W2+Apport(s)-Transfert(t)>=0)
                                Instruction(2)=Transfert(t);
                                for u=1:o+1
                                    if((Volume1(j)+W1-Controle(u)+Transfert(t))>=0))
                                        Instruction(1)=Controle(u);
                                        New_Vol=Matrice*Instruction+Entree+Volume;
                                        if(optimal12<rho*g*(Volume1(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(l)*mu3*Apport(s))+V(i)
```

Dans la partie définition des matrices de résolutions et des vecteurs de discrétilisations, on crée les vecteurs *volume*, *Entree* et *Instruction*, présentée plus haut.

On crée ensuite les matrices correspondant aux graphes de types *serie* et *arbre*, qui sont également présentées plus haut.

À l'aide d'une structure conditionnelle *if*, on teste la valeur de la chaîne de caractère *type_graphe*, donnée en paramètre de la fonction et on lui associe la matrice du graphe correspondant, que l'on va utiliser

dans l'algorithme et dans la partie Simulation.

Dans l'algorithme, on remplit les coefficient des vecteur *Volume* et *Instruction* au fur et à mesure des boucles for à l'aide des nouvelles itérations.

Le vecteur *Entree* reste quant à lui constant sur toute la durée de production N .

On remplace aussi la mise à jour des volumes des retenues par l'expression du produit matriciel présenté plus haut et on le stocke dans le vecteur *New_Vol*, que l'on utilise ensuite pour calculer la valeur de production associée dans l'appel récursif au coefficient de la matrice de résolution V , écrite par remontée.

Dans la partie Simulation, on crée de nouveau les vecteurs *Volume* et *Instruction* et on les réutilise avec *Entrée* et la matrice du graphe, pour mettre à jour les volumes des trois retenues, que l'on stocke dans *New_Vol*.

On utilise ensuite *New_Vol*, pour construire les vecteurs des graphes de la résolution et de la simulation.

La construction des graphes reste identique à celle du code précédent.

5.2.2 Quelques exemples de tests du code

On effectue quelques simulations avec différentes données en paramètres et on observe principalement les graphes des volumes des retenues et des différents contrôles appliquées, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité à la réalité physique.

On ne peut malheureusement pas tout tester dans ce même rapport, au vu du nombre croissant de paramètres, qui peuvent influencer les résolutions et les simulations au fil des situations.

De plus, avec ce nouveau code, le temps de calcul a été encore encore plus fortement rallongé, principalement à cause de toutes les affectations des vecteurs *Volume* et *Instruction* du produit matriciel, pour chaque itération, de chacune des nombreuse boucles for, ce qui fait augmenter rapidement le temps de calcul de l'algorithme et de la construction des vecteurs des graphes.

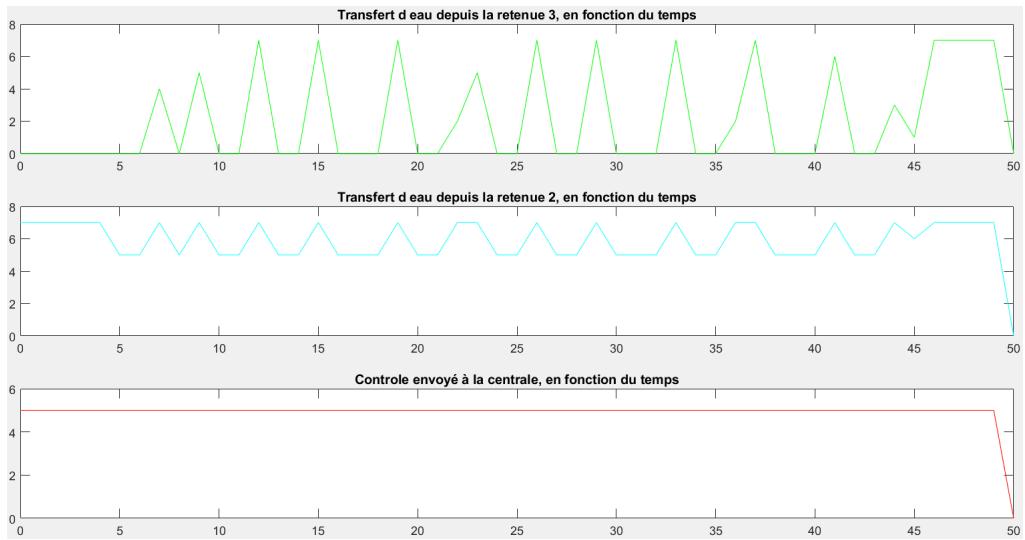
On commence par ce premier test :

```
[production,prog1,prog2,prog3]=Optimise_Production(50,25,25,5,7,7,50,2,5,2,"arbre",5,10,10)
```

Pour ce test, le rendement μ_1 de la retenue principale est de 0, le rendement μ_2 de la retenue secondaire est de 0 et le rendement μ_3 de la troisième retenue est 0.8.

Après un temps de compilation d'environ 5 minutes, on obtient alors les résultats suivants :





On a réalisé ce test avec exactement les mêmes paramètres que pour le code précédent.
 On remarque que les résultats obtenus sont bien conformes à ceux obtenus précédemment, pour l'autre version du code.

On observe que, même si ce code est deux fois plus court et bien moins complexe que le code précédent, il est cependant environ 4 fois plus long à compiler que celui-ci, ce qui le rend bien moins pratique que l'autre.

5.3 Résolution avec ajout des entrées d'eaux non constantes et des tarifs horaires de l'électricité

Pour cette dernière résolution de notre problème avec un barrage à 3 retenues, on simule maintenant les entrées d'eaux journalières W_1 et W_2 , au choix par une fonction périodique ou gaussienne sur toute la durée de production N et on considère aussi les différents tarifs horaires du prix de l'électricité.

On repart de la première version du code, de la résolution du problème à trois retenues, avec des entrées d'eaux constantes.

5.3.1 Présentation du code et de la résolution numérique en Matlab

Voici l'ensemble des paramètres de notre dernier problème :

```
% L = taille maximale de la retenue principale du barrage.
% L2 = taille maximale de la retenue secondaire du barrage.
% L3 = taille maximale de la retenue tertiaire du barrage.
% o = taille maximale de l'ouverture des conduites forcées, de la retenue principale.
% T = taille maximale de l'ouverture du canal de transfert, de la retenue secondaire.
% S = taille maximale de l'ouverture du canal des apports, de la retenue tertiaire.
% N = durée totale de la période de production d'électricité.
% type_retenue1 = sélectionne le type d'entrée d'eau dans la retenue1, les valeurs possibles sont "periodique" ou "crue".
% type_retenue2 = sélectionne le type d'entrée d'eau dans la retenue2, les valeurs possibles sont "periodique" ou "crue".
% type_retenue3 = sélectionne le type d'entrée d'eau dans la retenue3, les valeurs possibles sont "periodique" ou "crue".
% type_graphe = sélectionne le type de liaison entre les 3 retenues, les valeurs possibles sont "arbre" ou "serie".
% vol_depart = valeur arbitraire du volume présent dans le barrage au départ, pour pouvoir construire le graphe.
% vol_reserve_depart = valeur arbitraire du volume au départ, dans la seconde retenue du barrage.
% vol_reserve2_depart = valeur arbitraire du volume au départ, dans la troisième retenue du barrage.
% P_H_Pleine = Prix de l'électricité, en heure pleine.
% P_H_Creuse = Prix de l'électricité, en heure creuse.
% Periode_H = Période entre les changements de tarifs, heures pleines et heures creuses.
```

Comme précédemment, on ajoute les trois variables $type_retenue1$, $type_retenue2$ et $type_retenue3$, pour générer les trois vecteurs des entrées d'eaux, à partir d'une fonction périodique ou gaussienne dépendant de la valeur de chacune des chaînes de caractères correspondantes.

On ajoute également les trois variables P_H_Pleine , P_H_Creuse et $Periode_H$, pour la prise en compte des tarifs du prix de l'électricité, en fonction du temps.

Voici le code de la fonction qui effectue la résolution basée sur la nouvelle adaptation de l'algorithme de programmation dynamique, et qui réalise aussi une simulation à partir des paramètres de la situation qui sont donnés en entrée, avant de retourner tous les résultats voulus et d'afficher les graphes de la simulation :

```
function [prod_max,Programme1,Programme2,Programme3]=Optimise_Production(L,L2,L3,o,T,S,N,type_retenue1,type_retenue2,type_retenue3,type_graphe,vol_depart)
% Données
rho=1000;
g=9.80665;
mu1=0.75;
mu2=0.8;
mu3=0.7;
%x=1;
%y=1;
%z=1;
% Définition des matrices
V=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs de production d'électricité entre les instants n et N.
U=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs des contrôles, pour la retenue principale.
U2=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs des transferts, de la retenue secondaire.
U3=zeros(N+1,L+1,L2+1,L3+1); % Matrices des valeurs des transferts, de la retenue tertiaire.
Volume=[0:L]; % Vecteur des valeurs des niveaux de remplissages, de la retenue principale du barrage.
Volume2=[0:L2]; % Vecteur des valeurs des niveaux de remplissages, de la retenue secondaire du barrage.
Volume3=[0:L3]; % Vecteur des valeurs des niveaux de remplissages, de la retenue tertiaire du barrage.
Controle=[0:o]; % Vecteur des tailles d'ouvertures de la conduite forcée.
Transfert=[0:T]; % Vecteur des valeurs possibles de transferts de la retenue secondaire.
Apport=[0:S]; % Vecteur des valeurs possibles de transferts de la retenue tertiaire.
Production_elec_Prix=ones(1,N+1); % Vecteur des valeurs de production d'électricité, en prix, en fonction du temps.
Production_elec_cumulee=ones(1,N+1); % Vecteur des valeurs des gains de production cumulés, en prix, en fonction du temps.
% Construction des vecteurs des entrées d'eau:
W1=zeros(1,N+1); % Vecteurs des entrées d'eaux.
W2=zeros(1,N+1);
W3=zeros(1,N+1);
if(type_retenue1=="crue")
```

```

for i=0:N
    W1(i+1)=0+(L/4)*exp(-(i-(N/2))^2/(N/4));
end
elseif(type_retenue1=="periodique")
    for i=0:N
        W1(i+1)=(L/8)+(L/8)*cos((6*N)*i);
    end
else
    for i=0:N
        W1(i+1)=0;
    end
end
if(type_retenue2=="crue")
    for i=0:N
        W2(i+1)=0+(L2/4)*exp(-(i-(N/2))^2/(N/4));
    end
elseif(type_retenue2=="periodique")
    for i=0:N
        W2(i+1)=(L2/8)+(L2/8)*cos((6*N)*i);
    end
else
    for i=0:N
        W2(i+1)=0;
    end
end
if(type_retenue3=="crue")
    for i=0:N
        W3(i+1)=0+(L3/4)*exp(-(i-(N/2))^2/(N/4));
    end
elseif(type_retenue3=="periodique")
    for i=0:N
        W3(i+1)=(L3/8)+(L3/8)*cos((6*N)*i);
    end
else
    for i=0:N
        W3(i+1)=0;
    end
end
% Construction du vecteur des prix de l'électricité, en fonction du temps:
Prix=[];
            % Vecteur des prix.
Compteur=1;
Changement=false;
for i=0:N
    if Compteur==Periode_H
        Compteur=0;
        Changement=not(Changement);
    end
    if Changement
        Prix(i+1)=P_H_Pleine;
    else
        Prix(i+1)=P_H_Creuse;
    end
    Compteur=Compteur+1;
end
if(type_graphe=="serie")
    % Construction des Matrices U et V, par remontée.
    for i=N:-1:1      % Boucle sur le temps
        for j=1:L+1    % Boucle sur le volume du barrage
            for k=1:L2+1 % Boucle sur le volume de la retenue secondaire
                for l=1:L3+1 % Boucle sur le volume de la retenue tertiaire
                    optimal2=0; % Contiendra le maximum par rapport aux transferts et apports, pour chaque contrôle.
                    for s=1:S+1
                        if(Volume3(l)+W3(i)-Apport(s)>=0)
                            for t=1:T+1
                                if(Volume2(k)+W2(i)+Apport(s)-Transfert(t)>=0)
                                    for u=1:o+1
                                        if((Volume(j)+W1(i)-Controle(u)+Transfert(t)>=0))
                                            if(optimal2<=Prix(i)*rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(l)*mu3*Apport(z))
                                                optimal2=Prix(i)*rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(l)*mu3*Apport(z));
                                                x=u;
                                                y=t;
                                                z=s;
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
    V(i,j,k,l)=optimal2;
    U(i,j,k,l)=Controle(x);
    U2(i,j,k,l)=Transfert(y);
    U3(i,j,k,l)=Apport(z);
end

```

```

        end
    end
end
[prod_max,rang]=max(V(1,:,:));
% Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U,U2 et V.
Controle_opt=zeros(1,N+1);
Controle_opt(1)=U(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
Transfert_opt=zeros(1,N+1);
Transfert_opt(1)=U2(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
Apport_opt=zeros(1,N+1);
Apport_opt(1)=U3(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
vol_courant=zeros(1,N+1);
vol_courant(1)=vol_depart;
vol_reserve=zeros(1,N+1);
vol_reserve(1)=vol_reserve_depart;
vol_reserve2=zeros(1,N+1);
vol_reserve2(1)=vol_reserve2_depart;
Production_elec_Prix(1)=rho*g*(vol_courant(1)*mu1*Controle_opt(1)+vol_reserve(1)*mu2*Transfert_opt(1)+vol_reserve2(1)*mu3*Apport_opt(1))*Prix(1);
Production_elec_cumulee(1)=Production_elec_Prix(1);
for i=2:N+1
    vol_courant(i)=max(min(L,vol_courant(i-1)+W1(i-1)-Controle_opt(i-1)+Transfert_opt(i-1)),0);
    vol_reserve(i)=max(min(L2,vol_reserve(i-1)+W2(i-1)+Apport_opt(i-1)-Transfert_opt(i-1)),0);
    vol_reserve2(i)=max(min(L3,vol_reserve2(i-1)+W3(i-1)-Apport_opt(i-1)),0);
    Controle_opt(i)=U(1,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
    Transfert_opt(i)=U2(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
    Apport_opt(i)=U3(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
    Production_elec_Prix(i)=rho*g*(vol_courant(i)*mu1*Controle_opt(i)+vol_reserve(i)*mu2*Transfert_opt(i)+vol_reserve2(i)*mu3*Apport_opt(i))*Prix(i);
    for j=i:N+1
        Production_elec_cumulee(j)=Production_elec_cumulee(j)+Production_elec_Prix(i);
    end
end
elseif(type_graphe=="arbre")
    % Construction des Matrices U et V, par remontée.
    for i=N:-1:1 % Boucle sur le temps
        for j=1:L+1 % Boucle sur le volume du barrage
            for k=1:L2+1 % Boucle sur le volume de la retenue secondaire
                for l=1:L3+1 % Boucle sur le volume de la retenue tertiaire
                    optimal2=0; % Contiendra le maximum par rapport aux transferts et apports, pour chaque contrôle.
                    for s=1:S+1
                        if(Volume3(l)+W3(i)-Apport(s)>=0)
                            for t=1:T+1
                                if(Volume2(k)+W2(i)-Transfert(t)>=0)
                                    for u=1:o+1
                                        if((Volume(j)+W1(i)-Controle(u)+Transfert(t)+Apport(s)>=0))
                                            if(optimal2<=Prix(i)*rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(l)*mu3*Apport(x,u))
                                                optimal2=Prix(i)*rho*g*(Volume(j)*mu1*Controle(u)+Volume2(k)*mu2*Transfert(t)+Volume3(l)*mu3*Apport(x,u));
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                    V(i,j,k,l)=optimal2;
                    U(i,j,k,l)=Controle(x);
                    U2(i,j,k,l)=Transfert(y);
                    U3(i,j,k,l)=Apport(z);
                end
            end
        end
    end
    [prod_max,rang]=max(V(1,:,:));
    % Construction des vecteurs des CONTROLES OPTIMAUX Controle_opt et des VOLUMES OPTIMAUX vol_courant, grâce aux matrices U,U2 et V.
    Controle_opt=zeros(1,N+1);
    Controle_opt(1)=U(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
    Transfert_opt=zeros(1,N+1);
    Transfert_opt(1)=U2(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
    Apport_opt=zeros(1,N+1);
    Apport_opt(1)=U3(1,vol_depart+1,vol_reserve_depart+1,vol_reserve2_depart+1);
    vol_courant=zeros(1,N+1);
    vol_courant(1)=vol_depart;
    vol_reserve=zeros(1,N+1);
    vol_reserve(1)=vol_reserve_depart;
    vol_reserve2=zeros(1,N+1);
    vol_reserve2(1)=vol_reserve2_depart;
    Production_elec_Prix(1)=rho*g*(vol_courant(1)*mu1*Controle_opt(1)+vol_reserve(1)*mu2*Transfert_opt(1)+vol_reserve2(1)*mu3*Apport_opt(1))*Prix(1);
    Production_elec_cumulee(1)=Production_elec_Prix(1);
    for i=2:N+1
        vol_courant(i)=max(min(L,vol_courant(i-1)+W1(i-1)-Controle_opt(i-1)+Transfert_opt(i-1)+Apport_opt(i-1)),0);
        vol_reserve(i)=max(min(L2,vol_reserve(i-1)+W2(i-1)-Transfert_opt(i-1)),0);
        vol_reserve2(i)=max(min(L3,vol_reserve2(i-1)+W3(i-1)-Apport_opt(i-1)),0);
    end
end

```

```

Contrôle_opt(i)=U(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
Transfert_opt(i)=U2(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
Apport_opt(i)=U3(i,round(vol_courant(i))+1,round(vol_reserve(i))+1,round(vol_reserve2(i))+1);
Production_elec_Prix(i)=rho*g*(vol_courant(i)*mu1*Contrôle_opt(i)+vol_reserve(i)*mu2*Transfert_opt(i)+vol_reserve2(i)*mu3*Apport_opt(i))*P
for j=1:N+1
    Production_elec_cumulee(j)=Production_elec_cumulee(j)+Production_elec_Prix(i);
end
end
% Récupération des résultats
Programme1=Contrôle_opt(1:N);
Programme2=Transfert_opt(1:N);
Programme3=Apport_opt(1:N);
% Construction du graphe, du volume de la situation de production optimale, en fonction du temps.
% Test fonction sinusoïdale:
x=[0:N]; % Vecteur abscisses du temps.
clf
hold on
subplot(3,1,1);
plot(x,vol_reserve2,'green');
title('Volume d eau de la retenue tertiaire, en fonction du temps')
hold on
plot(x,W3,'black');
hold off
subplot(3,1,2);
plot(x,vol_reserve,'cyan');
title('Volume d eau de la retenue secondaire, en fonction du temps')
hold on
plot(x,W2,'red');
hold off
subplot(3,1,3);
plot(x,vol_courant,'blue'); % Graphe du volume d'eau, du barrage.
title('Volume d eau de la retenue principale, en fonction du temps')
hold on
plot(x,W1,'magenta')
hold off
hold off
figure(2)
clf
hold on
subplot(3,1,1);
plot(x,Apport_opt,'green');
title('Transfert d eau depuis la retenue 3, en fonction du temps')
subplot(3,1,2);
plot(x,Transfert_opt,'cyan');
title('Transfert d eau depuis la retenue 2, en fonction du temps')
subplot(3,1,3);
plot(x,Contrôle_opt,'red');
title('Contrôle envoyé à la centrale, en fonction du temps')
hold off
figure(3)
clf
hold on
subplot(3,1,1);
plot(x,Prix,'green');
title('Tarifs horaires de l électricité, en fonction du temps')
hold off
figure(2)
clf
hold on
subplot(3,1,1);
plot(x,Apport_opt,'green');
title('Transfert d eau depuis la retenue 3, en fonction du temps')
subplot(3,1,2);
plot(x,Transfert_opt,'cyan');
title('Transfert d eau depuis la retenue 2, en fonction du temps')
subplot(3,1,3);
plot(x,Contrôle_opt,'red');
title('Contrôle envoyé à la centrale, en fonction du temps')
hold off
figure(3)
clf
hold on
subplot(3,1,1);
plot(x,Prix,'green');
title('Tarifs horaires de l électricité, en fonction du temps')
subplot(3,1,2);
plot(x,Production_elec_Prix,'cyan');
title('Production électrique journalières en Prix, en fonction du temps')
subplot(3,1,3);
plot(x,Production_elec_cumulee,'red');
title('Gain total cumulée, en fonction du temps')
hold off
end

```

Dans la partie définition des Matrices de résolutions et des vecteurs de discrétilisations, on ajoute les vecteurs *Production_elec_Prix* et *Production_elec_cumulee*.

Ensuite, on construit les vecteurs des entrées d'eaux $W1$, $W2$ et $W3$, de la même manière que dans le code pour un barrage à deux retenues.(cf partie 4.3.1)

Après cela, on construit le vecteur du prix de l'électricité, en fonction du temps, toujours de la même manière que dans la partie 4.3.1 de ce rapport.

Dans les algorithmes de programmation dynamique des deux types de graphes, on multiplie encore le prix de l'électricité, en fonction du temps $Prix(i)$, dans l'expression de la fonction valeur de production instantanée $L(x, u)$.(cf Partie 1.2.1)

Dans les parties Simulation, on ajoute la construction des vecteurs $Production_elec_Prix$ et $Production_elec_cumulee$, en fonction du temps.

Enfin, pour finir, on ajoute les nouveaux graphes correspondant à ces vecteurs, dans de nouvelles fenêtres graphiques.

5.3.2 Quelques exemples de tests du code

On effectue quelques simulations avec différentes données en paramètres et on observe principalement les graphes des volumes des retenues et des différents contrôles appliquées, en fonction du temps. On tente d'interpréter la pertinence des résultats obtenus et leurs niveaux de fidélité à la réalité physique.

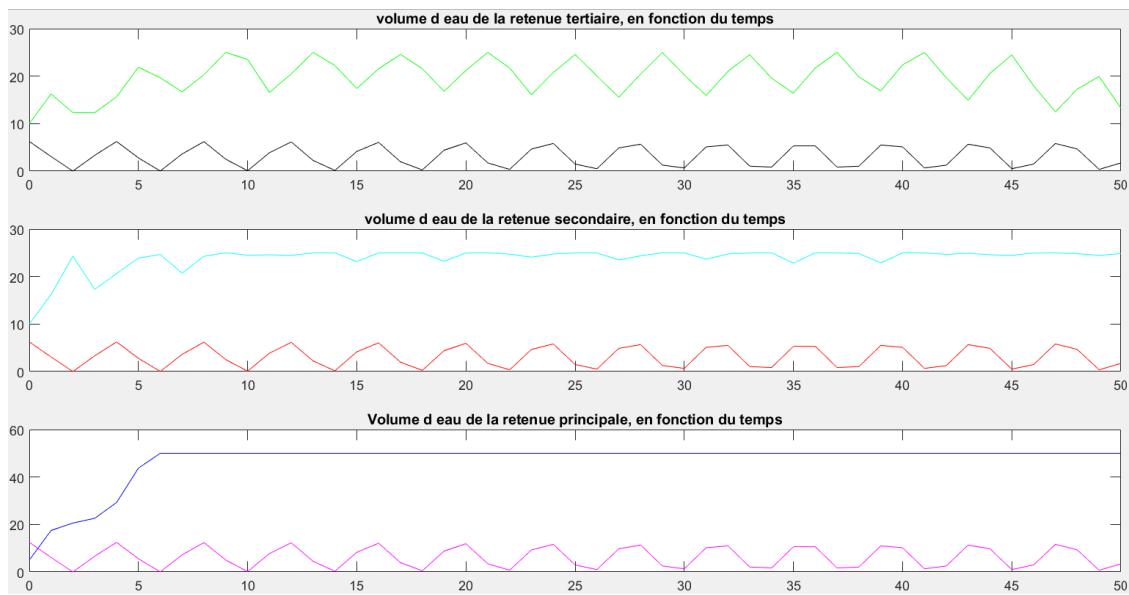
On ne peut malheureusement pas tout tester dans ce même rapport, au vu du nombre croissant de paramètres, qui peuvent influencer les résolutions et les simulations au fil des situations.
On se concentre donc ici sur l'ajout complémentaire de la nouvelle résolution.

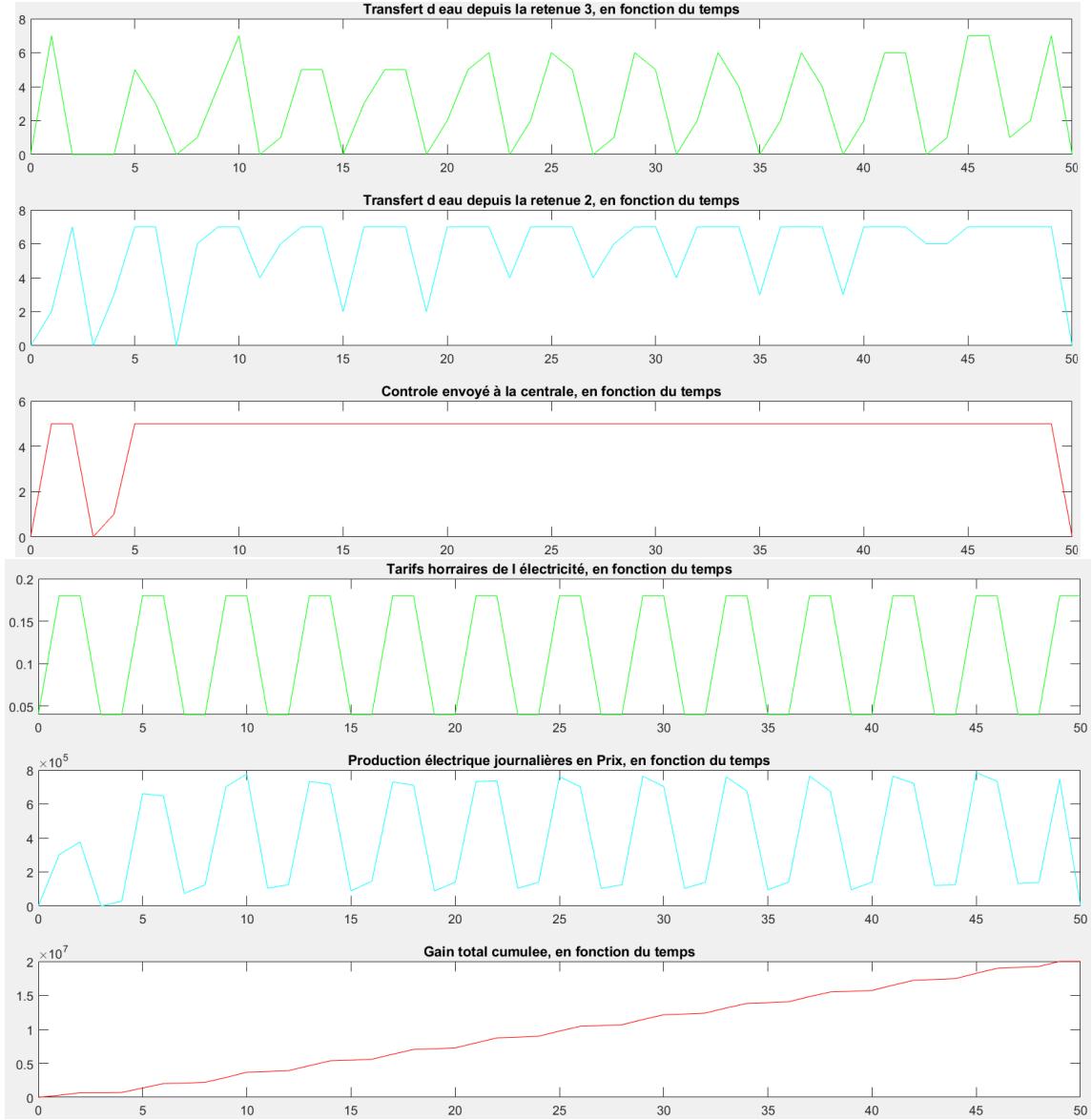
On commence par ce premier test :

```
[production,prog1,prog2,prog3]=Optimise_Production(50,25,25,5,7,7,50,"periodique","periodique","periodique","serie",5,10,10,0.18,0.04,2)
```

Pour ce test, le rendement μ_1 de la retenue principale est de 0.75, le rendement μ_2 de la retenue secondaire est de 0.8 et le rendement μ_3 de la troisième retenue est 0.7.

Après un temps de compilation d'environ 1m30s, on obtient alors les résultats suivants :





On observe que les retenues 1 et 2 se remplissent très rapidement et sont presque toujours pleines, puisque le graphe du barrage est ici en série.

Les contrôles des retenues 1 et 2 sont d'ailleurs presque toujours maximaux. Leurs pics sont bien alignés, avec les périodes temps en tarifs heures pleines.

On peut remarquer également que le volume de la retenue 3 oscille aussi sur le rythme des périodes de temps, en tarifs heures pleines.

Pour finir, les graphes des productions d'électricité ponctuelles et cumulées sont également bien synchronisées avec les oscillations des tarifs du prix de l'électricité, en fonction du temps.

Tout cela montre que le tarif horaire du prix de l'électricité est bien pris en compte dans la résolution du problème et dans la simulation de la situation correspondante.

5.3.3 Remarques

On ne traitera pas le cas, pour un barrage à trois retenues, avec les chaînes de Markov, car cela ajoute-rait trois nouvelles dimensions supplémentaires aux matrices V , U , $U2$ et $U3$, ce qui nous donneraient des

matrices à 7 dimensions.

De plus, cela ajouterait aussi 6 nouvelles boucles *for* et 3 nouveaux test *if*, à l'algorithme de programmation dynamique, qui possède déjà actuellement 7 boucle *for* et 4 test *if*.
L'algorithme deviendrait alors non compilable en temps raisonnable.

L'algorithme peut encore être adapté pour résoudre le problème pour un barrage avec N retenues, mais le nombres de boucles *for*, de matrices, de vecteurs, de dimensions et de graphes, augmentent alors avec le nombre de retenues et le code Matlab devient vite non compilable en temps raisonnable.

6 Bibliographie

Références

- [1] Programmation dynamique, pyramide des nombres. *wikipedia*.
- [2] Richard ernest bellman, programmation dynamique. *wikipedia*.
- [3] La production d'énergie en auvergne-rhône-alpes. *Observatoire Régional Climat Air énergie Auvergne Rhône-Alpes*, 2019.
- [4] Mikael Falconnet. Introduction aux chaînes de markov. pages 24–32, 2013-2014.
- [5] Zephyr Luckny. Thèse sur l'optimisation stochastique des systèmes multi-réservoirs par l'agrégation de scénarios et la programmation dynamique approximative. *Université Laval, Québec, Canada*, pages 3–11, 2015.
- [6] Olga Mula. Optimisation et programmation dynamique. *Université Paris Dauphine*, pages 33–34, 2019.