BERERD Mats
Machine learning in finance :
Theoretical foundations

ENSAE 3ème année
Master SFA
2022-2023

Deep Deterministic Portfolio Optimization

Table des matières

1 Introduction

La finance quantitative est un domaine qui s'appuie sur des modèles mathématiques et statistiques pour analyser les marchés financiers et prendre des décisions d'investissement. Dans ce contexte, le *trading* d'actions est une activité qui nécessite la prise de décisions dynamiques, c'est-à-dire des choix qui doivent être faits en temps réel, tels que le choix des actions à trader, leur prix et leur quantité, dans un environnement complexe et extrêmement volatil.

Le *Deep Reinforcement Learning* (DRL), ou apprentissage profond par renforcement, est une technique d'apprentissage automatique qui permet aux machines d'apprendre à travers l'interaction avec leur environnement. Dans le cadre du *trading* d'actions, cette méthode offre un cadre idéal pour développer des stratégies qui permettent d'équilibrer l'exploration de nouvelles pistes et l'exploitation des connaissances déjà acquises au fil du temps.

Cependant, dans quelle mesure ce type d'algorithme peut-il être employé pour obtenir des stratégies optimales de *trading* ? Bien que ces modèles aient montré des performances impressionnantes dans la pratique, leur reproductibilité est souvent remise en question. Afin de répondre à cette préoccupation, les auteurs de cette étude [1] cherchent à comparer les performances de l'algorithme plébiscité *Deep Deterministic Policy Gradient* (DDPG) à la solution optimale connue d'un environnement de *trading* conceptuellement simple.

2 Cadre théorique

2.1 Optimisation dynamique pour l'allocation de portefeuille

Le *trading* d'actions est caractérisé par une nature stochastique et hautement interactive, ce qui permet de considérer la prise de décision comme un processus de décision Markovien (MDP). À chaque instant t, l'agent récupère toutes les informations disponibles telles que les changements de prix, puis prend une action et calcule son gain. Formellement, le problème d'optimisation dynamique s'écrit :

$$\max_{\{a_t \in \mathcal{A}\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \mathrm{rwd}_t(s_t, a_t, s_{t+1}, \xi_t)\right] \tag{1}$$

$$\text{s.t. } s_{t+1} = f_t(s_t, a_t, \eta_t)$$

avec $a_t \in \mathcal{A}$ l'action choisie dans l'espace des actions (on appelle $\{a_t\}$ la politique/le contrôle), $s_t \in \mathcal{S}$ l'état de l'environnement à la date t, ξ_t et η_t des bruits et rwd_t est la fonction de gain perçu.

Dans le cadre de l'allocation de portefeuille (que l'on restreint au cas unidimensionnel pour simplifier), l'investisseur alloue une portion $\pi_t \in \mathbb{R}$ sur un actif, qui donne un rendement r_t, pour maximiser une utilité $U(\cdot)$ sur la richesse future :

$$\max_{\{\pi_t\}} \mathbb{E}\left[U\left(\sum_{t=0}^{T-1} \mathrm{PnL}_{t,t+1}\right)\right] \tag{2}$$

avec $\mathrm{PnL}_{t,t+1} = \mathrm{gain}_{t+1} - \mathrm{cost}_{t,t+1}$ avec $\mathrm{gain}_t = \pi_t r_t$ le profit du portefeuille à chaque date t et $\mathrm{cost}_{t,t+1}$ une fonction de coûts incluant notamment les coûts de *trading*.

Nous réduisons le problème au cas où la fonction d'utilité peut se ramener à la forme :

$$\max_{\{\pi_t\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1} r_{t+1} - \mathrm{cost}(|\pi_{t+1} - \pi_t|) - \mathrm{risk}(\pi_{t+1})\right] \tag{3}$$

avec $\mathrm{risk}(\pi_t)$ une fonction de régularisation des portefeuilles risqués (*e.g.* $\mathrm{risk}(\pi_t) = \pi_t^2$) suivient de la fonction d'utilité. Cela permet notamment d'intégrer les modèles moyenne-variance de Marcowitz.

Pour se placer dans le cadre de l'optimisation dynamique, les actions peuvent se voir comme les trades entre deux pas de temps : $a_t = \pi_{t+1} - \pi_t$. L'espace des états est plus complexe à modéliser. Une hypothèse standard est la décomposition du revenu r_t en un terme prévisible p_t et un bruit $\eta_t^{(r)}$. De plus, on suppose que p_t est un processus AR(1) de paramètre ρ, de bruit $\eta_t^{(p)}$. L'optimisation dynamique pour allocation de portefeuille s'écrit finalement :

$$\max_{\{\pi_t\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1} r_{t+1} - \mathrm{cost}(|a_t|) - \mathrm{risk}(\pi_{t+1})\right] \tag{4}$$

$$\pi_{t+1} = \pi_t + a_t \tag{5}$$

$$p_{t+1} = \rho p_t + \eta_t^{(p)} \tag{6}$$

$$r_{t+1} = p_t + \eta_t^{(r)} \tag{7}$$

L'état $s_t = (\pi_t, p_t)$ contient toute l'information disponible à la date t. Deux cas de figure sont étudiés :

1. le cas où $r_{t+1} \mapsto p_t$ ($\eta_t^{(r)} = 0$ p.s.) : l'agent a une observation parfaite de l'espace des états en t. En effet, le gain perçu est fonction des paramètres observables pi_t, p_t et a_t.
2. le cas où le revenu est perturbé par le bruit $\eta_t^{(r)}$ qui ne vaut pas 0 p.s., ce qui constitue un cadre d'évaluation plus complexe.

Trois environnements d'étude, où la politique optimale est connue, sont retenus :

1. Coût et risque quadratique
2. Coût proportionnel et risque quadratique
3. Coût proportionnel et risque *maxpos*

2.2 Coût et risque quadratique

Sous ces hypothèses, le problème devient une Commande LQ (*Linear-Quadratic Regulator*, LQR) en temps discret et horizon fini :

$$\max_{\{\pi_t\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1} p_{t+1} - \Gamma a_t^2 - \lambda \pi_{t+1}^2\right] \tag{8}$$

L'expression explicite s'obtient en prenant le gain moyen à horizon infini (*i.e.* en divisant le gain total par T, puis en prenant la limite $T \to \infty$) :

$$a_t = -K s_t \tag{9}$$

avec K unique solution d'un système d'Équations Différentielles Ordinaires (EDO) de Riccatti, qui dépend linéairement de l'état.

Sans le terme de coût, la solution suit une allocation de Marcowitz :

$$\pi_{t+1}^* = \pi_{t+1}^{(M)} = \frac{p_t}{2\lambda} \tag{10}$$

La composante coût modifie l'allocation de Marcowitz qui s'exprime s'écrit alors comme un lissage exponentiel :

$$\pi_{t+1}^* = (1-\omega)\pi_t + \omega\psi\pi_{t+1}^{(M)} \tag{11}$$

avec $\psi = \frac{\omega}{1-(1-\omega)\rho}$, $\omega = f_c\left(\sqrt{\frac{\lambda}{T}}\right)$ et $f_c(x) = \frac{2}{1+\sqrt{1+\frac{4}{x}}} = \frac{x}{2}(\sqrt{x^2+4} - x)$. Dans le cadre de l'étude $\Gamma = 1, \lambda = 0.3$ et $\rho = 0.9$.

2.3 Coût proportionnel et risque quadratique

Le deuxième cadre d'étude prend en compte un coût linéaire :

$$\max_{\{\pi_t\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1} p_{t+1} - \Gamma |a_t| - \lambda \pi_{t+1}^2\right] \tag{12}$$

La politique optimale est un système de bande sans achat autour de l'allocation de Marcowitz :

$$\pi^*_{t+1} = \begin{cases} u(\pi^{(M)}_{t+1}) & \text{si } \pi_t > u(\pi^{(M)}_{t+1}) \\ l(\pi^{(M)}_{t+1}) & \text{si } \pi_t < l(\pi^{(M)}_{t+1}) \\ \pi_t & \text{sinon} \end{cases} \tag{13}$$

Bien que les bornes $u(\cdot)$ et $l(\cdot)$ ne sont pas triviales, une bonne approximation est de considérer la bande symétrique autour de l'allocation de Marcowitz, avec b obtenue par une *grid search* :

$$u(\pi^{(M)}_{t+1}) = \pi^{(M)}_{t+1} + b \quad \text{et} \quad l(\pi^{(M)}_{t+1}) = \pi^{(M)}_{t+1} - b \tag{14}$$

Dans ce cadre d'étude $\Gamma = 4, \lambda = 0.3$ et $\rho = 0.9$.

2.4 Coût proportionnel et risque *maxpos*

La contrainte de risque *maxpos* impose une limite maximum aux positions prises : $|\pi_t| \leq M$. Ce qui donne :

$$\max_{\{|\pi_t| \leq M\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \pi_{t+1} p_{t+1} - \Gamma |a_t|\right] \tag{15}$$

La stratégie optimale est alors de trader le maximum autorisé quand le prédicteur p_t dépasse un seuil :

$$\pi^*_{t+1} = \begin{cases} M & \text{si } \pi_t > q \\ -M & \text{si } \pi_t < -q \\ \pi_t & \text{sinon} \end{cases} \tag{16}$$

La valeur q est aussi approchée par *grid search*. Les expérimentations sont faites avec $\Gamma = 4, M = 2$ et $\rho = 0.9$.

3 L'algorithme DDPG

Deep Deterministic Policy Gradient est un algorithme acteur-critique. La partie acteur permet d'estimer une politique déterministe $\phi_\Theta : \mathcal{S} \to \mathcal{A}$ (Θ désignant les paramètres du réseau de neurones), lorsque la partie critique donne une approximation de la fonction valeur $\mathcal{Q}$. Pour cela les hypothèses fondamentales sont que l'espace des actions est considéré continu et $\mathcal{Q}(s, a)$ est supposée différentiable en a.

De plus, DDPG s'entraîne *off-policy*, autrement dit, l'agent apprend à partir d'une

politique différente de celle qu'il va suivre lorsqu'il agit avec l'environnement. Il va donc utiliser une politique « cible »(*target policy*), alors que les données d'apprentissage sont générées selon une *behavorial policy* $\beta(a|s)$. On note ρ^β la distribution de l'espace des états sous β. L'équation de Bellman associée à l'équation (1) est donnée par, avec $\mathcal{Q}_\Theta = \mathcal{Q}^{\phi_\Theta}$, la fonction valeur associée à la politique ϕ_Θ :

$$\begin{aligned}\mathcal{Q}_\Theta(s_t, a) &= \mathbb{E}_{s_{t+1}\sim\rho^\beta}[\mathrm{rwd}_t(s_t, a) + \gamma \max_{a'} \mathcal{Q}_\Theta(s_{t+1}, a')] \\ &= \mathbb{E}_{s_{t+1}\sim\rho^\beta}[\mathrm{rwd}_t(s_t, a) + \gamma \mathcal{Q}_\Theta(s_{t+1}, \phi_\Theta(s_{t+1})]\end{aligned}$$

La fonctionnelle de gain s'écrit alors :

$$\mathcal{J}^\beta(\phi_\Theta) = \int_{s\in\mathcal{S}} \rho^\beta(s)\mathcal{Q}_\Theta(s, \phi_\Theta(s))ds$$

Et l'on a, selon [4] :

$$\nabla_\Theta \mathcal{J}(\phi_\Theta) \approx \mathbb{E}_{s\sim\rho^\beta}[\nabla_\Theta \phi_\Theta(s) \nabla_a \mathcal{Q}_\Theta(s, a)|_{a=\phi_\Theta(s)}] \tag{17}$$

Cette approximation permet d'actualiser Θ par montée de gradient, en enlevant la dépendance à Θ de $\mathcal{Q}_\Theta$.

L'entraînement *off-policy* laisse l'agent intéragir avec l'environnement pour récupérer de l'expérience sous forme de tuples $(s_t, a_t, \mathrm{rwd}_t, s_{t+1})$ dans un *replay buffer* (composée de données d'entraînement déjà utilisée), puis échantillonne des nouvelles données d'entraînement pour mettre à jour la partie critique puis la partie acteur. Voir 6.1.

3.1 *Replay Buffer*

Pour réduire le biais d'estimation du gradient, un *batch* de tuples $(s_{t_j}, a_{t_j}, \mathrm{rwd}_{t_j}, s_{t_j+1})_{1\leq j\leq b}$ de taille b est échantillonné du *replay buffer*. Chaque tuple représentant de « l'expérience », ils sont pondérés selon leurs erreurs TD (*Temporal Difference*) :

$$\delta(s, a, \mathrm{rwd}, s') = \mathrm{rwd} + \gamma \max_{a'} \mathcal{Q}(s', a') - \mathcal{Q}(s, a)$$

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha},\ 1 \leq i \leq N,\ 0 \leq \alpha \leq 1$$

avec N la taille du *replay buffer*, $p_i = |\delta_i| + \varepsilon$, ε un petit nombre positif pour assurer une probabilité non nulle de tirage. L'idée est que les expériences avec une erreur élevée sont celles mal jugées par la critique, d'où le besoin se se ré-entraîner sur ces valeurs. Cette méthode est appelée *Prioritized Experience Replay* (PER) [3].

Comme l'échantillonnage non-uniforme implique un biais sur l'estimation du gradient,

nous le corrigeons en utilisant $\alpha_i \delta_i$ au lieu de δ_i pour l'actualisation de la partie critique, avec :

$$\alpha_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^{\beta}, \ 0 \leq \beta \leq 1$$

3.2 Exploration

Est nommée épisode une suite d'interaction que l'agent a face à l'environnement, initié par une position $\pi_0 = 0$. Par la suite, l'agent effectue l'action $a_t = a_t^{pred} + \eta_t^{(a)}$, étant donnée sa position actuelle π_t et le revenu prévisible p_t :

$$\begin{cases} a_t^{pred} = \phi_\Theta(p_t, \pi_t) \\ \eta_t^{(a)} = (1 - \rho^{expl}) \eta_{t-1}^{(a)} + \sigma^{expl} \epsilon_t \\ \eta_0^{(a)} = 0 \end{cases}$$

$\eta_t^{(a)}$ est le bruit d'exploration, qui suit un processus AR(1) avec $\epsilon_t \sim_{iid} \mathcal{N}(0,1)$, $\sigma^{explo} > 0$.

3.3 Actualisation de la partie critique actuelle

On note $\mathcal{Q}, \tilde{\mathcal{Q}}, \phi$ et $\tilde{\phi}$ la critique actuelle, la critique cible, l'acteur actuel et l'acteur cible, de paramètre respectifs $\omega, \tilde{\omega}, \Theta, \tilde{\Theta}$. L'actualisation des critiques se fait à partir de :

$$\tilde{Q}_i = \text{rwd}_{t_i} + \gamma \tilde{\mathcal{Q}} \left(s_{t_i+1}, \tilde{\phi}(s_{t_i+1}) \right)$$

$$Q_i = \mathcal{Q}(s_{t_i}, a_{t_i})$$

$$\delta_i = Q_i - \tilde{Q}_i$$

Puis ω est actualisé par descente de gradient sur la fonction de perte suivante, avec b la taille du *batch* :

$$\mathcal{L}_{critique}(\omega) = \frac{1}{2b} \sum_{i=1}^{b} \alpha_i \delta_i^2$$

$$\omega_{t+1} = \omega_t + \frac{1}{b} \sum_{i=1}^{b} \alpha_i \delta_i \nabla_\omega \tilde{\mathcal{Q}}(s_i, a_i)$$

3.4 Actualisation de la partie acteur actuelle

En utilisant l'équation (17), Θ est actualisé par montée de gradient sur :

$$\mathcal{L}_{acteur}(\Theta) = -\frac{1}{b}\sum_{i=1}^{b} \mathcal{Q}(s_i, \phi_\Theta(s_i))$$

$$\Theta_{t+1} = \Theta_t + \frac{1}{b}\sum_{i=1}^{b} \nabla_\Theta \phi_\Theta(s_i) \nabla_a \mathcal{Q}(s_i, a_i)|_{a=\phi_\Theta(s)}$$

3.5 Actualisation des parties cibles

Les paramètres des parties cibles sont actualisés par moyennise de Polyak, ce qui rend l'entraînement plus stable :

$$\tilde{\omega} \leftarrow \tau_c\, \omega + (1-\tau_c)\, \tilde{\omega},\ 0 < \tau_c < 1$$

$$\tilde{\Theta} \leftarrow \tau_a\, \Theta + (1-\tau_a)\, \tilde{\Theta},\ 0 < \tau_a < 1$$

3.6 Subtilité du cas *maxpos*

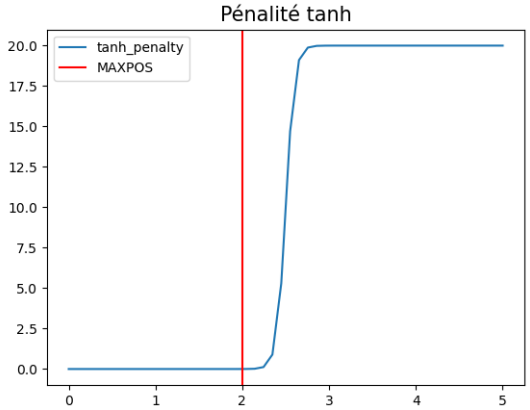
La contrainte *maxpos* viole l'hypothèse de différentiabilité de la fonction $\mathcal{Q}$. Pour contourner ce problème, deux astuces sont utilisées :

— les positions de l'acteur sont rognées pour respecter la contrainte.

— un coût supplémentaire est ajouté à la fonction de gain,pour pénaliser les positions ne respectant pas la contrainte :

$$\text{rwd}(p, \pi, a) = p\pi - \Gamma|a| - \beta\{\tanh[\alpha(|\pi + a| + (1+\gamma)M] + 1\}$$

Les expériences utilisent $\beta = 10, \alpha = 20$ et $\gamma = 0.25$



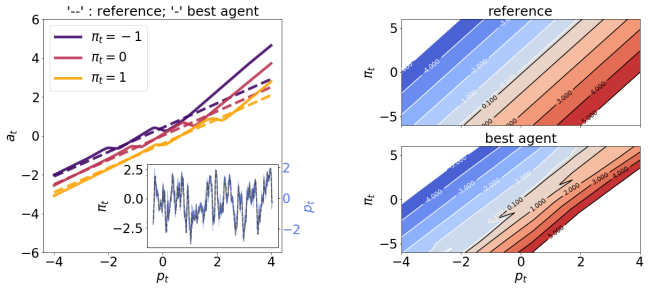


FIGURE 1 – Coût et risque quadratique, sans bruit dans le gain

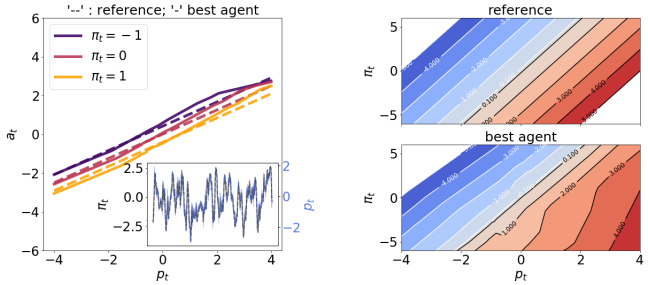


FIGURE 2 – Coût et risque quadratique, avec bruit dans le gain

4 Résultats

Les différents graphiques présentés ci-après comparent l'algorithme présentant les meilleures performances à la solution optimale théorique (ou approchée par *grid search*).

Les graphiques sur la gauche comparent les différentes actions a_t en fonction du revenu prévisible p_t pour trois positions $\pi_t \in \{-1, 01\}$, le graphique interne compare la politique apprise (en bleue) par rapport à la politique optimale de référence (en noire) pour une trajectoire de revenu prévisible simulée. Les graphiques sur la droite comparent la politique de référence à la politique apprise, quand $|a_t| \leq 5$ sur le plan (π_t, p_t). Les résultats détaillés sont dans 6.2.

4.1 Coût et risque quadratique

Bien que la politique apprise ne dévie que très peu de la stratégie optimale, on voit que cette dernière présente parfois un caractère sous-optimal. Dans les faits, l'algorithme converge très vite lors de l'entraînement vers la forme optimale puis dévie légèrement. Cela est notamment dû au caractère relativement plat de la fonction de gain autour de sa valeur maximale. De plus, les politiques plus extrêmes sont moins fréquentes, ce qui permet une plus grande variation dans ces régions sans pour autant avoir un impact trop lourd sur la fonction de gain finale.

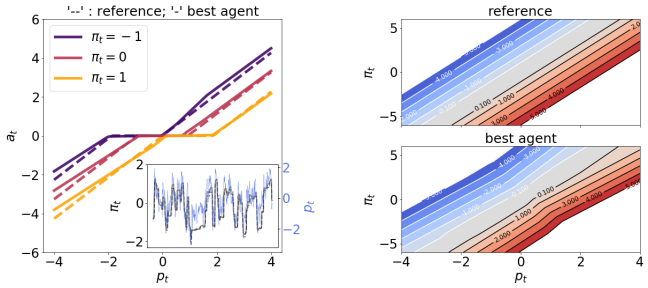


FIGURE 3 – Coût proportionnel et risque quadratique, sans bruit dans le gain

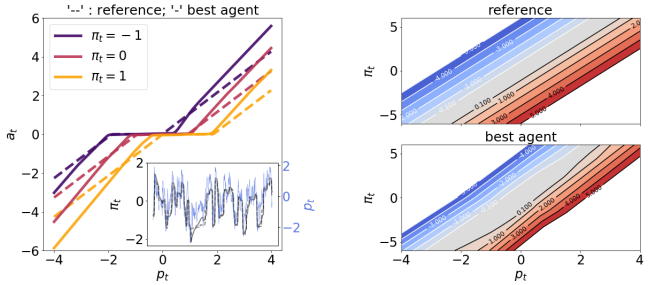


FIGURE 4 – Coût proportionnel et risque quadratique, avec bruit dans le gain

4.2 Coût proportionnel et risque quadratique

De manière similaire, DDPG arrive à retrouver les composantes principales de la stratégie optimale : les zones sans *trading* sont bien restituées et les pentes en dehors de cette bande sont correctes (presque linéaire). Pendant l'entraînement, l'algorithme apprend d'abord les zones où il peut agir puis perfectionne ensuite la stratégie dans la bande autorisée pour améliorer la gestion du risque.

4.3 Coût proportionnel et risque *maxpos*

Si la solution optimale est conceptuellement simple (mais discontinue), cet environnement est le plus difficile à gérer pour un algorithme *model-free* comme DDPG qui suppose

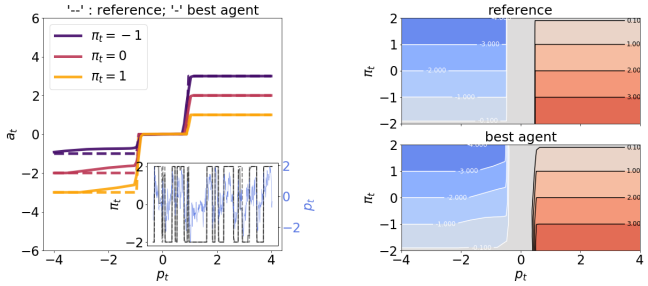


FIGURE 5 – Coût proportionnel et risque *maxpos*, sans bruit dans le gain

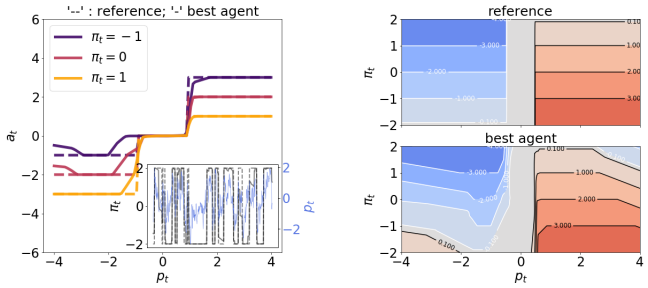


FIGURE 6 – Coût proportionnel et risque *maxpos*, avec bruit dans le gain

justement la continuité de l'espace des actions. Il en résulte que la variance du portefeuille n'est pas du tout contrôlée : la stratégie optimale est de conserver la position constante la plupart du temps et de la changer un nombre limité de fois, ce qui implique une variabilité élevée de la fonction de gain.

5 Discussion

Voici les principales conclusions que nous pouvons déduire des résultats :

— De manière générale, DDPG parvient à apprendre des stratégies de *trading* proche des solutions optimales en terme de fonction de gain et de *PnL*. DDPG arrive donc à approcher correctement l'auto-corrélation du revenu prévisible et à trouver un bon compromis entre gain, coût et gestion du risque.

— Un fait intéressant est que les performances au niveau *PnL* sont atteintes plus rapidement qu'au niveau fonction de gain. Cela suggère que l'algorithme apprend d'abord à repérer les signaux de *trading* et maîtriser les coûts avant de s'intéresser à optimiser le contrôle du risque.

— Ajouter du bruit à la fonction de gain est plus complexe à gérer : les résultats sont plus variables.

— Le troisième environnement est le plus difficile : plusieurs agents n'ont pas convergé.

Quelques remarques peuvent cependant être faites :

— DDPG suppose un espace d'actions continu, il est donc indispensable d'avoir une connaissance fine de cet espace pour pouvoir modifier l'algorithme en conséquence (*c.f.* le cas *maxpos*).

— D'autres ajustements pourront sûrement améliorer la convergence et la précision de l'algorithme, *e.g.* changer la méthode d'exploration (ici ϵ-greedy) [2] [5].

— DDPG est *model-free*, *i.e.* la fonction de gain est supposée non-connue au moment de l'entraînement. Ajouter ces informations de modèle peut aussi améliorer les performances.

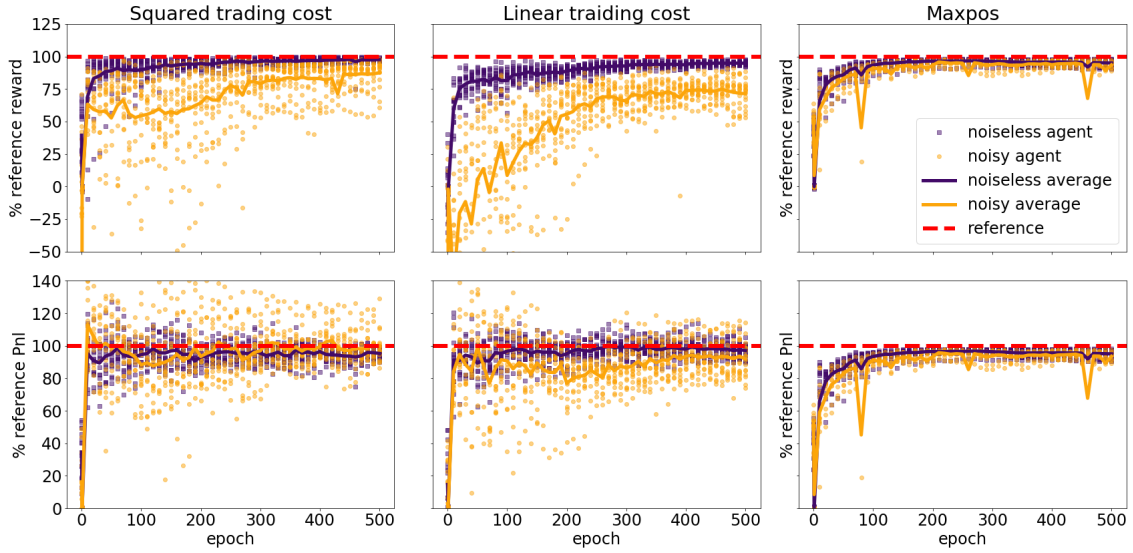


FIGURE 7 – Résultats d'entraînements

6 Annexe

6.1 Pseudo-code DDPG

Voir Algorithme 6.1

6.2 Résultats supplémentaires

La figure 7 montre les performances de 16 agents initialisés sous des *seeds* différentes, en violet quand les agents ont accès à toute l'information et en jaune lorsque du bruit est présent dans le gain. La ligne rouge présente la fonction de gain ou le PnL théorique sous information complète. Les performances des agents sont mesurées selon les résultats sur un jeu de données test composées de 10 environnements de pas $T = 5000$. Les courbes colorées représentent la moyenne des performances des 16 agents. Pour le cas *maxpos*, seul les agents qui ont convergé sont reportés, ce qui crée un biais positif.

Les tableaux (1-3) donnent les performances des 16 agents. La colonne *Diff* est calculée comme la norme l_1 par la position prise par l'agent et celle prise par la référence. '-' signifie que certains agents n'ont pas convergé.

6.2.1 Coût et risque quadratique

La *heat map* 8 montre le résultat de la *GridSearch* effectuée pour trouver les valeurs approchées de Γ et λ. On remarque le caractère relativement plat de cette fonction de gain.

Algorithm 1 DDPG avec PER

Définir le processus AR du revenu prévisible
Initialiser les réseaux $\mathcal{Q}^{\omega}$ et ϕ^{Θ} et la taille b du *replay buffer*
Initialiser les réseaux cibles $\tilde{\mathcal{Q}}^{\tilde{\omega}}$ et $\tilde{\phi}^{\tilde{\Theta}}$
Initialiser l'environnement
Initialiser le bruit d'exploration $(\eta_t^{(a)})_{1\leq t\leq T_{pretrain}}$
for $t = 1$ à $T_{pretrain}$ **do**
  Observer s_t
  Prendre l'action $a_t = \phi^{\Theta}(s_t) + \eta_t^{(a)}$
  Observer rwd_t et l'état suivant s_{t+1}
  Ajouter $(s_t, a_t, \text{rwd}_t, s_{t+1})$ avec priorité $p = |\text{rwd}_t|$ au *replay buffer*
end for
for episode= 1 à n **do**
  Initialiser l'environnement
  Initialiser le bruit d'exploration $(\eta_t^{(a)})_{1\leq t\leq T}$
  for $t = 1$ à T **do**
    Observer s_t
    Prendre l'action $a_t = \phi^{\Theta}(s_t) + \eta_t^{(a)}$
    Observer rwd_t et l'état suivant s_{t+1}
    Ajouter $(s_t, a_t, \text{rwd}_t, s_{t+1})$ avec priorité maximale au *replay buffer*
    if $t \equiv 0[\tau]$ **then**
      Échantillonner un *batch* $(s_{t_i}, a_{t_i}, \text{rwd}_{t_i}, s_{t_i+1})_{1\leq i\leq b}$ via PER selon la distribution $P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}, 0 \leq \alpha \leq 1$
      Calculer $\forall i \in \{1, ..., b\} : \tilde{\mathcal{Q}}_i = \text{rwd}_{t_i} + \gamma \tilde{\mathcal{Q}}\left(s_{t_i+1}, \tilde{\phi}(s_{t_i+1})\right)$
      Calculer $\forall i \in \{1, ..., b\} : \delta_i = |\mathcal{Q}(s_i, a_i) - \tilde{\mathcal{Q}}_i|$
      Calculer les poids $\forall i \in \{1, ..., b\} : \alpha_i = \left(\frac{1}{N \times P(i)}\right)^{\beta}$
      Normaliser les poids $\forall i \in \{1, ..., b\} : \alpha_i \leftarrow \frac{\alpha_i}{\max_k(\alpha_k)}$
      Actualiser les priorités $\forall i \in \{1, ..., b\} : p_i = |\delta_i|$
      Actualiser le gradient $\mathcal{L}_{critique}(\omega) = \frac{1}{2b}\sum_{i=1}^{b} \alpha_i \delta_i^2$
      Actualiser le gradient $\mathcal{L}_{acteur}(\omega) = -\frac{1}{b}\sum_{i=1}^{b} \mathcal{Q}\left(s_{t_i}, \phi^{\Theta}(s_{t_i})\right)$
      Actualiser les réseaux cibles :

$$\tilde{\omega} \leftarrow \tau_c\, \omega + (1 - \tau_c)\, \tilde{\omega}$$
$$\tilde{\Theta} \leftarrow \tau_a\, \Theta + (1 - \tau_a)\, \tilde{\Theta}$$

    end if
  end for
end for

	Reward		PnL		Diff	
	no noise	noise	no noise	noise	no noise	noise
reference	0.681		1.298		0	
best	0.677	0.671	1.291	1.674	0.081	0.128
mean	0.665	0.596	1.237	1.295	0.140	0.383
worst	0.655	0.415	1.170	1.119	0.218	0.781
75%-tile	0.668	0.640	1.258	1.316	0.161	0.491
50%-tile	0.666	0.624	1.239	1.276	0.132	0.349
25%-tile	0.660	0.588	1.215	1.215	0.106	0.256

TABLE 1 – Résultats Coût et risque quadratique

	Reward		PnL		Diff	
	no noise	noise	no noise	noise	no noise	noise
reference	0.254		0.492		0	
best	0.248	0.225	0.518	0.562	0.063	0.131
mean	0.241	0.181	0.478	0.452	0.093	0.250
worst	0.234	0.135	0.442	0.364	0.126	0.441
75%-tile	0.244	0.196	0.491	0.486	0.101	0.301
50%-tile	0.239	0.188	0.482	0.455	0.090	0.244
25%-tile	0.238	0.167	0.464	0.414	0.080	0.181

TABLE 2 – Résultats Coût proportionnel et risque quadratique

	Reward		PnL		Diff	
	no noise	noise	no noise	noise	no noise	noise
reference	0.901		0.901		0	
best	0.884	0.876	0.884	0.876	0.101	0.143
mean	0.856	0.842	0.856	0.842	0.198	0.246
worst	0.815	0.803	0.815	0.803	0.321	0.346
75%-tile	0.849	0.849	0.849	0.849	0.148	0.210
50%-tile	0.862	-	0.862	-	0.184	-
25%-tile	0.826	-	0.826	-	0.239	-

TABLE 3 – Résultats Coût proportionnel et risque *maxpos*

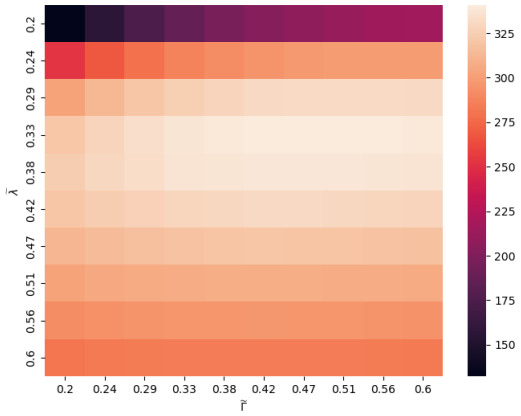


FIGURE 8 – *Grid Search* de la fonction de gain pour coût et risque quadratique

La figure 9 montre le comportement de DDPG, selon le nombre d'épisodes dans l'apprentissage, la figure 10 montre le comportement de DDPG et de la solution optimale face à un processus d'Ornstein-Uhlenbeck qui représente p_t.

6.2.2 Coût proportionnel et risque quadratique

La *heat map* 11 montre le résultat de la *GridSearch* effectuée pour trouver les valeurs approchées de Γ et λ.

La figure 12 montre le comportement de DDPG, selon le nombre d'épisodes dans l'apprentissage, la figure 13 montre le comportement de DDPG et de la solution optimale face à un processus d'Ornstein-Uhlenbeck qui représente p_t.

6.2.3 Coût proportionnel et risque *maxpos*

La figure 14 montre le comportement de DDPG, selon le nombre d'épisodes dans l'apprentissage, la figure 15 montre le comportement de DDPG et de la solution optimale face à un processus d'Ornstein-Uhlenbeck qui représente p_t.

6.3 Lien *Github*

`https://github.com/mdlbm/ML4F-DDPO`

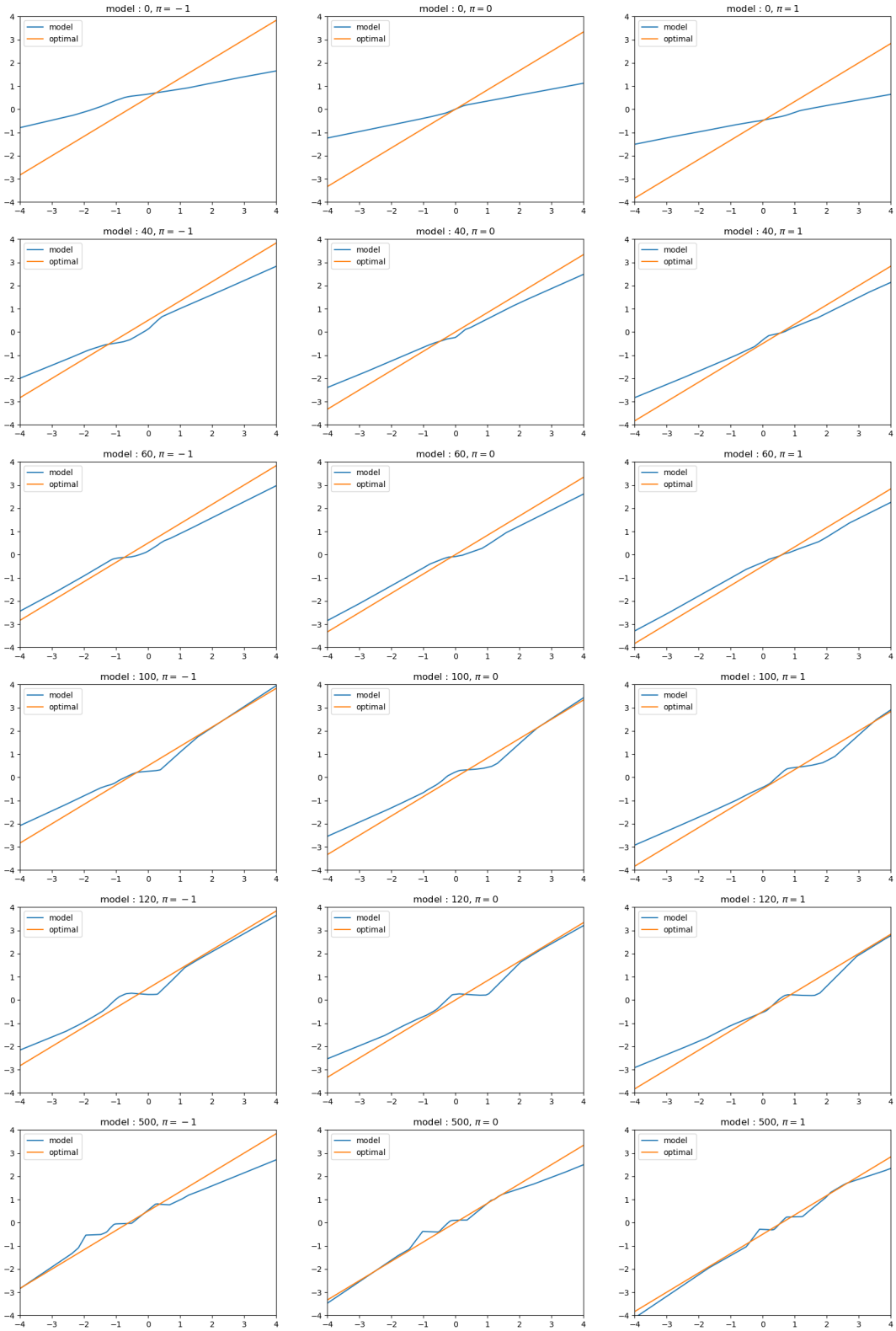


FIGURE 9 – Comparaison des différentes actions prises selon p_t pour des positions $\pi_t \in \{-1, 0, 1\}$ selon le nombre d'épisodes d'entraînement, coût et risque quadratique

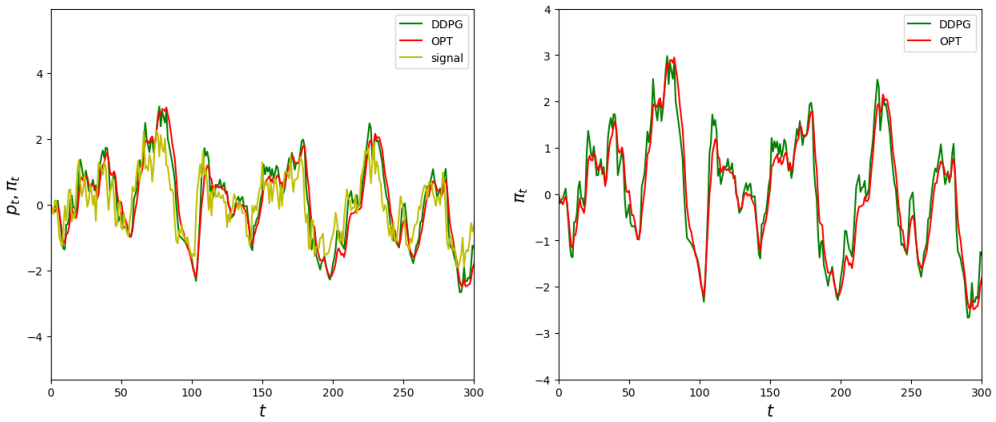


FIGURE 10 – Comparaison des actions prises par DDPG et la solution optimale étant donné p_t (processus d'Ornstein-Uhlenbeck), coût et risque quadratique

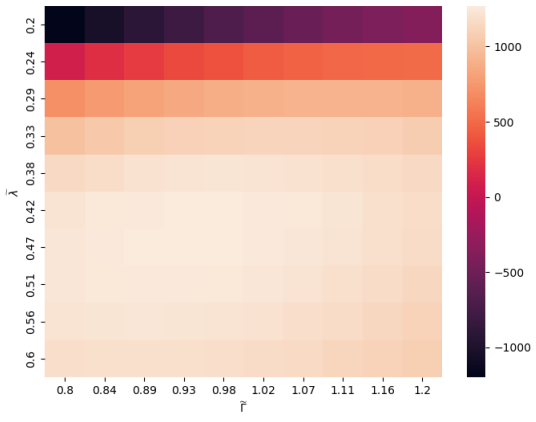


FIGURE 11 – *Grid Search* de la fonction de gain pour coût proportionnel et risque quadratique

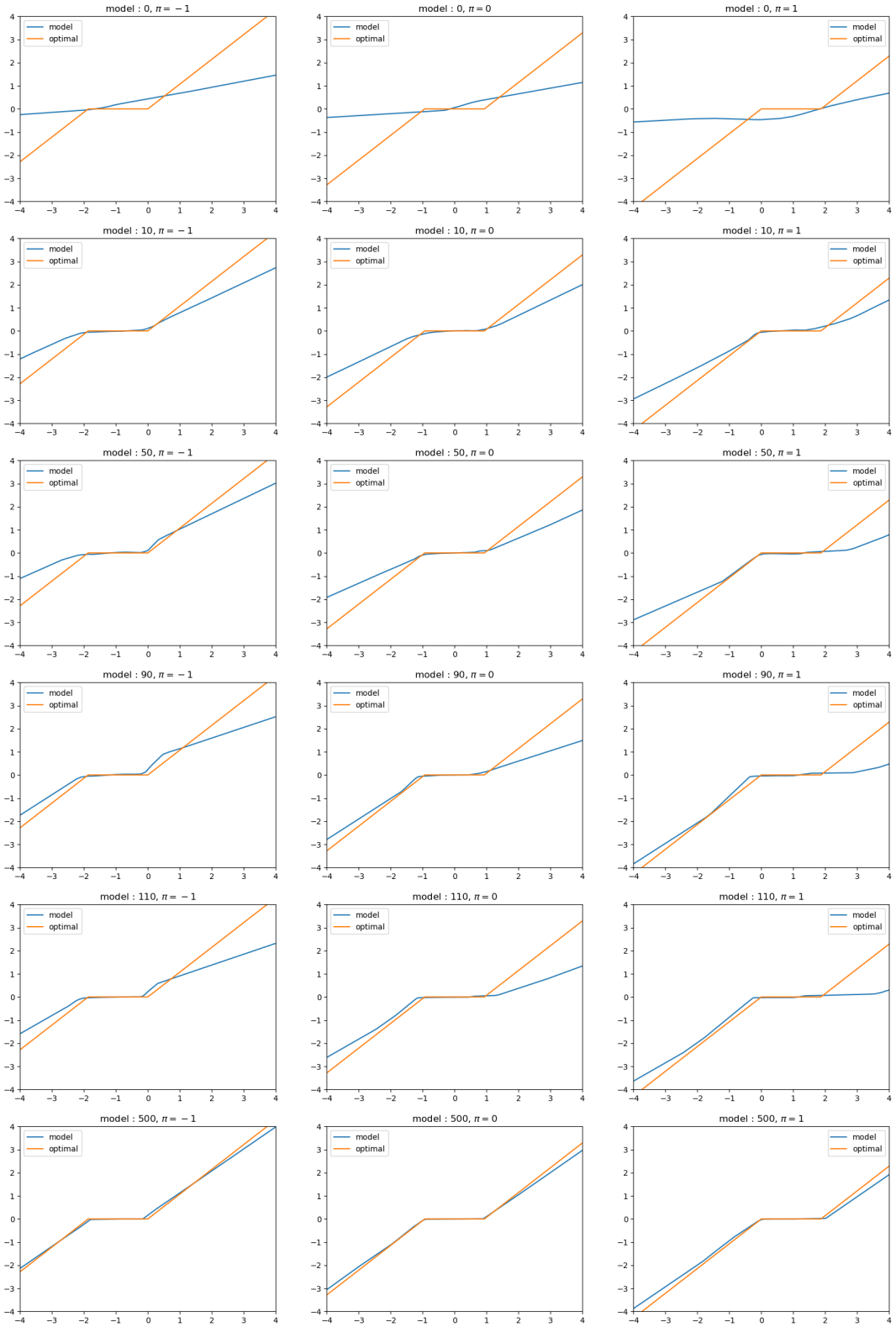


FIGURE 12 – Comparaison des différentes actions prises selon p_t pour des positions $\pi_t \in \{-1, 0, 1\}$ selon le nombre d'épisodes d'entraînement, coût proportionnel et risque quadratique

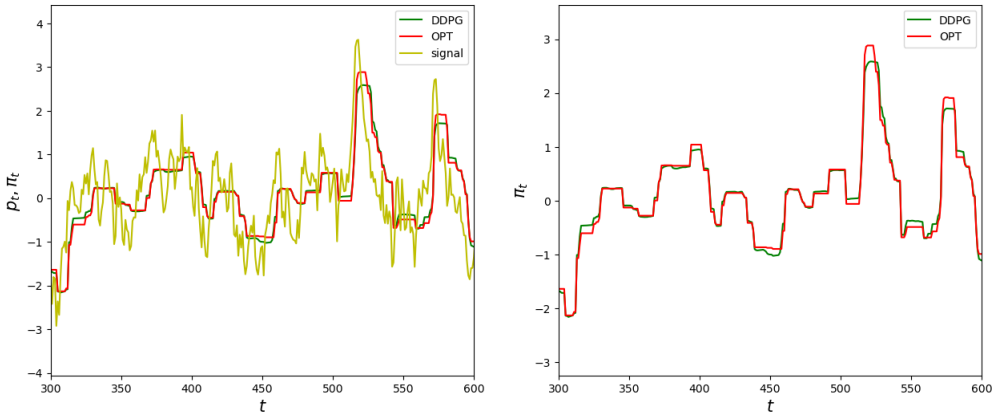


FIGURE 13 – Comparaison des actions prises par DDPG et la solution optimale étant donné p_t (processus d'Ornstein-Uhlenbeck), coût proportionnel et risque quadratique

Références

[1] Ayman CHAOUKI et al. *Deep Deterministic Portfolio Optimization*. 2020. arXiv : 2003.06497 [q-fin.MF].

[2] Matthias PLAPPERT et al. *Parameter Space Noise for Exploration*. 2018. arXiv : 1706.01905 [cs.LG].

[3] Tom SCHAUL et al. *Prioritized Experience Replay*. 2016. arXiv : 1511.05952 [cs.LG].

[4] David SILVER et al. "Deterministic Policy Gradient Algorithms." In : *ICML*. T. 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, p. 387-395. URL : http://dblp.uni-trier.de/db/conf/icml/icml2014.html#SilverLHDWR14.

[5] Yan YAN et Quan LIU. "Policy Space Noise in Deep Deterministic Policy Gradient". In : *Neural Information Processing*. Sous la dir. de Long CHENG, Andrew Chi Sing LEUNG et Seiichi OZAWA. Cham : Springer International Publishing, 2018, p. 624-634. ISBN : 978-3-030-04179-3.

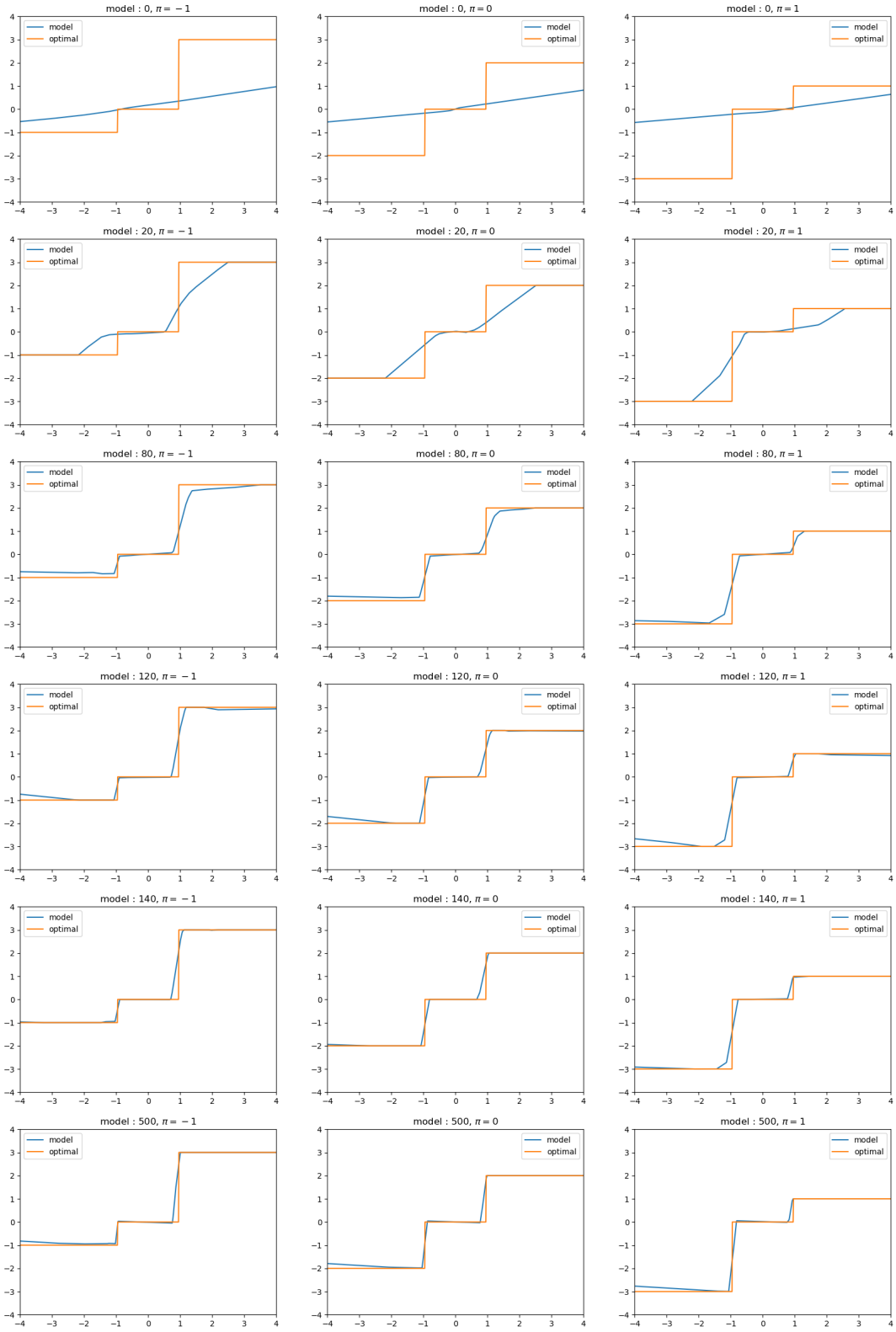


FIGURE 14 – Comparaison des différentes actions prises selon p_t pour des positions $\pi_t \in \{-1, 0, 1\}$ selon le nombre d'épisode d'entraînement, coût proportionnel et risque *maxpos*

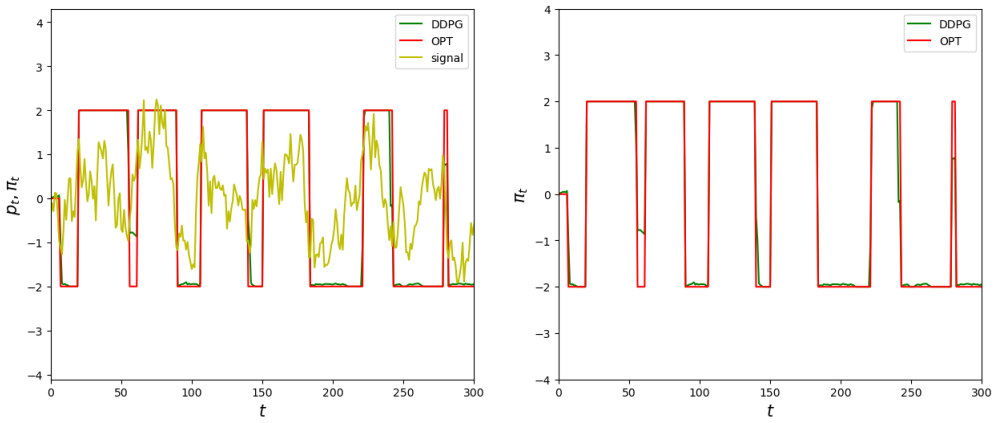


FIGURE 15 – Comparaison des actions prises par DDPG et la solution optimale étant donné p_t (processus d'Ornstein-Uhlenbeck), coût proportionnel et risque *maxpos*