

Deep learning methods for high dimensional partial differential equations

An application to control problems

Carlos Daniel Contreras Quiroz

Tesis de maestría
Departamento de matemáticas
Universidad de los Andes

30 de Junio 2023

Las ecuaciones en derivadas parciales son útiles para modelar todo tipo de fenómenos

1. En finanzas la ecuación de Black-Scholes

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0 \quad (1)$$

2. En control estocástico la ecuación HJB

$$\begin{aligned} \frac{\partial V}{\partial t} + H(t, x, D_x V, D_{xx} V) &= 0 \\ V(T, x) &= g(x), \end{aligned} \quad (2)$$

3. En mecánica cuántica la ecuación de Schrödinger

$$i\hbar \frac{\partial}{\partial t} \Psi(x, t) = \left[-\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + V(x, t) \right] \Psi(x, t) \quad (3)$$

Usualmente no tenemos acceso a soluciones analíticas, por lo cual usamos métodos numéricos como

1. Diferencias finitas
2. Elementos finitos
3. Métodos espectrales

Todos estos sufren la "*la maldición de la dimensionalidad*" ("*the curse of dimensionality*"). Esto es, la complejidad computacional incrementa exponencialmente con la dimensión del problema que se quiere resolver $O(N^d)$.

Pero hay problemas interesantes en dimensiones altas

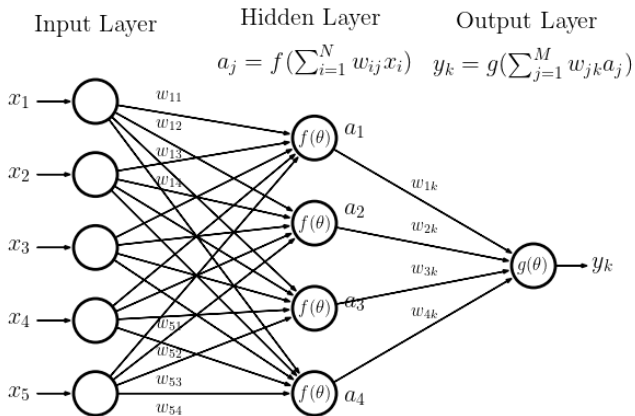
1. Evaluar el precio justo de una opción que tiene muchos activos subyacentes
2. Encontrar controles óptimos para problemas con espacio de estados grande
3. Encontrar la dinámica cuántica de un problema de muchos cuerpos para diseñar chips o nuevos fármacos

Entonces ¡necesitamos métodos para resolverlas!

Un problema similar surge en la modelación estadística de datos, para lo cual se han propuesto técnicas de *machine learning*. En particular, el uso de *redes neuronales artificiales* (ANN) ha mostrado ser útil para resolver cierto tipo de problemas.

Queremos usar estas mismas herramientas para resolver este problema para ecuaciones en derivadas parciales

¿Qué es una red neuronal?



Son una forma **expresiva** de parametrizar funciones $\varphi_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^m$, mediante composición de operaciones sencillas

$$x_{k+1} = \sigma(Wx_k + b)$$

Y podemos calcular $\nabla_\theta \varphi$ usando la regla de la cadena: diferenciación automática(AD).

La idea es usar este tipo de parametrizaciones para resolver PDE's. Para esto se han planteado varias formas

1. Deep Galerkin Method (DGM) y Physics Informed Neural Networks(PINNs)
2. Fourier Neural Operators
3. **Métodos basados en BSDEs**

¿Que es una BSDE?

Una SDE standard en $[0, T]$ es de la forma

$$\begin{aligned}X_t &= x_0 + \int_0^t b(s, X_s) ds + \int_0^t \sigma(s, X_s) dW_s \\X_0 &= x_0,\end{aligned}\tag{4}$$

¿Qué pasa si la consideramos hacia atrás, con una condición terminal?

$$\begin{aligned}X_t &= \xi + \int_t^T b(s, X_s) ds + \int_t^T \sigma(s, X_s) dW_s \\X(T) &= \xi\end{aligned}\tag{5}$$

Nada asegura que X_t sea \mathcal{F}_t -medible, por lo que el problema no está bien puesto.

¿Que es una BSDE?

Podemos forzar este proceso para ser \mathcal{F}_t -medible añadiendo un proceso Z que le "restará" estocasticidad a ξ para que la solución sea adaptada.

$$Y_t = \xi + \int_t^T b(s, Y_s, Z_s) ds - \int_t^T Z_s dW_s \quad (6)$$

Que se puede escribir hacia adelante como

$$\begin{aligned} dY_t &= -f(t, Y_t, Z_t)dt + Z_t \cdot dW_t \\ Y(T) &= \xi \end{aligned} \quad (7)$$

Noten el cambio de notación $X_t := Y_t$ y $b := f$

También podemos acoplar una SDE con una BSDE para obtener una ecuación Forward-Backward

.

$$\begin{aligned}dX_s &= \mu(t, X_s)ds + \sigma(s, X_s)dW_s \\X_t &= x \\dY_s &= -f(s, X_s, Y_s, Z_s)ds + Z_sdW_s \\Y_T &= g(X_T),\end{aligned}\tag{8}$$

Lo importante de estos procesos es que los podemos relacionar con ecuaciones en derivadas parciales de la forma

$$\begin{aligned}\frac{\partial u}{\partial t}(t, x) + \mathcal{L}_t u(t, x) + f(t, x, u(t, x), \sigma(t, x)' D_x u(t, x)) &= 0 \\ u(T, x) &= g(x),\end{aligned}\tag{9}$$

donde

$$\mathcal{L}_t v = \mu(t, x) \cdot D_x u(t, x) + \frac{1}{2} \text{Tr}(\sigma(t, x) \sigma'(t, x) D_{xx}^2 u(t, x))\tag{10}$$

es el generador del proceso X_t .

Podemos asociar la solución (X_t, Y_t, Z_t) de una FBSDE con una PDE de en la forma anterior mediante el siguiente teorema

Theorem (Pham 2009)

Bajo ciertas hipótesis de regularidad, si v es una solución clásica de (9) entonces los procesos $dX_t = \mu(t, X_t)dt + \sigma(t, X_t) \cdot dW_t$ $X(0) = x$ $Y_t = u(t, X_t)$ y $Z_t = \sigma(t, X_t)'D_x u(t, X_t)$ son solución de la FBSDE (8).

Recíprocamente, si (X_t, Y_t, Z_t) es una solución de (8), entonces $u(t, x) = Y_t^{t,x}$ es continua y es una solución viscosa a (9).

Vamos a explotar esta conexión para resolver la PDE usando simulaciones de la FBSDE de la siguiente forma (Han, Jentzen, and E 2018)

- Discretizamos la FBSDE usando el esquema de Euler-Maruyama

$$X_{t_{n+1}} \approx X_{t_n} + \mu(t_n, X_{t_n}) \Delta t_n + \sigma(t_n, X_{t_n}) \Delta W_{t_n} \quad (11)$$

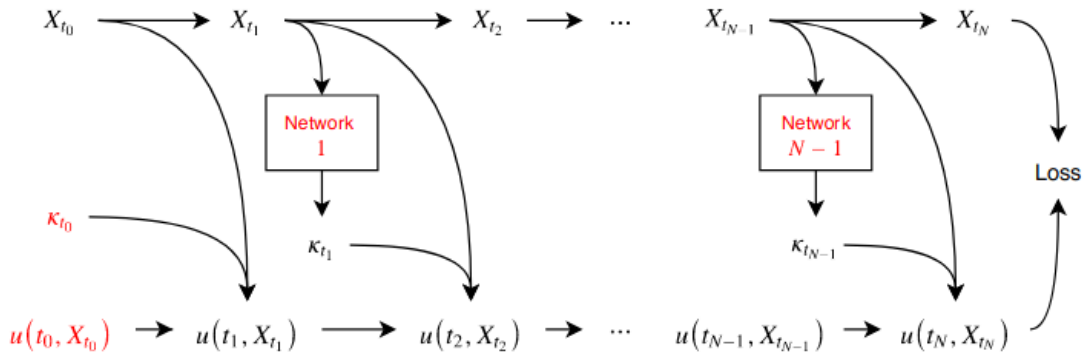
y

$$\begin{aligned} u(t_{n+1}, X_{t_{n+1}}) \approx & u(t_n, X_{t_n}) - f(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma'(t_n, X_{t_n}) D_x u(t_n, X_{t_n})) \Delta t_n \\ & + \sigma(t_n, X_{t_n})' D_x u(t_n, X_{t_n}) \Delta W_{t_n}, \end{aligned} \quad (12)$$

- Parametrizamos la funciones $X_0 \rightarrow u(0, Y_0)$ y $X_t \rightarrow \sigma'(t, X_t) D_x u(t, X_t)$ con redes neuronales
- Optimizamos los parámetros de la red intentando llegar a la condición terminal usando el costo

$$\ell(\theta) = \mathbb{E}[|g(X_{t_N}) - \hat{u}(\{X_{t_n}\}_0^N, \{W_{t_n}\}_0^N)|^2], \quad (13)$$

El método Deep BSDE



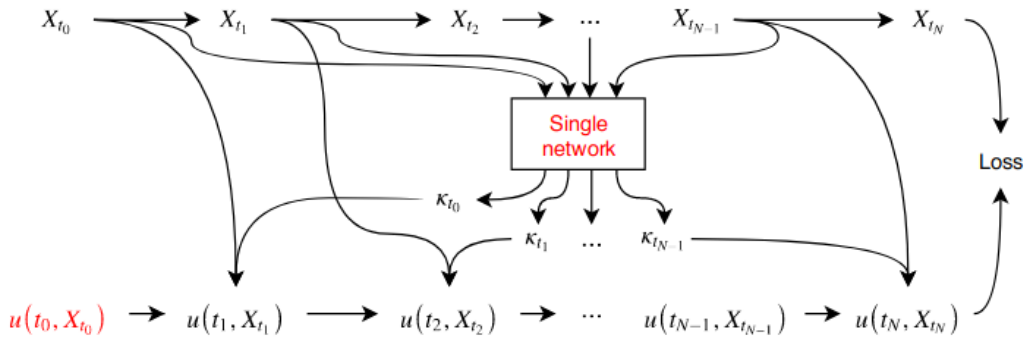
La estructura y método de optimización de la red que escojamos influye mucho en la calidad de la aproximación y velocidad de convergencia. En el artículo original se usó

- Una red completamente acoplada con 2 capas intermedias y $d+10$ neuronas en cada capa
- La función de activación *ReLU*
- Capas intermedias de normalización
- Recorte de gradientes
- El método ADAM para optimizar la red

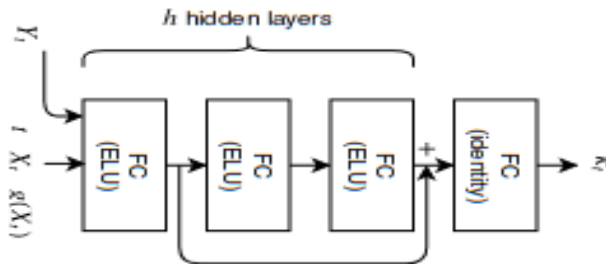
Variantes: Merged Deep BSDE

Desde entonces se han planteado varias variantes de este método

- Merged Deep BSDE (Chan-Wai-Nam, Mikael, and Warin 2018): Aproximar el gradiente con una sola red neuronal que incluya el tiempo en sus entradas



- Merged Residual Deep BSDE (Chan-Wai-Nam, Mikael, and Warin 2018): Aproximar el gradiente con una sola red neuronal que incluya el tiempo en sus entradas y usar una arquitectura con shortcuts



- Raissi's method(Raissi 2018): Aproximar directamente la solución de la PDE con una red neuronal, calcular las derivadas espaciales con diferenciación automática y optimizar la red con un costo que modele la diferencias locales de la evolución temporal

$$\ell(\theta) := \mathbb{E} \left[\sum_{i=0}^{N-1} \Phi(t_i, X_{t_i}, Y_{t_i}, Y_{t_{i+1}}, \Delta W_{t_i}) + (g(X_{t_N}) - Y_{t_N})^2 \right] \quad (14)$$

donde

$$\begin{aligned} \Phi(t_i, X_{t_i}, Y_{t_i}, Y_{t_{i+1}}, \Delta W_{t_i}) = & (Y_{t_{i+1}} - Y_{t_i} + f(t_i, X_{t_i}, Y_{t_i}, \sigma'(t_i, X_{t_i}) \hat{Z}_{t_i}) (\Delta t_i) \\ & - \hat{Z}'_{t_i} \sigma(t_i, X_{t_i}) (\Delta W_{t_i}))^2, \end{aligned} \quad (15)$$

Supongamos que tenemos 3 partículas en \mathbb{R}^2 cuyo estado conjunto describimos con un vector $\vec{X} = (x_1, y_1, x_2, y_2, x_3, y_3) \in \mathbb{R}^6$, que se mueven satisfaciendo la dinámica

$$\begin{aligned}dX_t &= 2\sqrt{\lambda}\alpha_t dt + \sqrt{2\nu}dW_t \\ X_0 &= x_0.\end{aligned}\tag{16}$$

Queremos elegir el control α_t para llevar a todas las partículas hacia un punto dado usando la menor cantidad de combustible posible y evitando que estén cerca entre sí.

Esto es, queremos elegir el control α_t que minimiza el costo dado por

$$J(\alpha_t) = \mathbb{E} \left[\int_0^T (|\alpha_t|^2 + F(t, X_t)) dt + g(X_T) \right], \quad (17)$$

donde modelamos la aversión entre partículas con

$$F(t, X_t) = F(t, (\vec{x}_1, \vec{x}_2, \vec{x}_3)) = C \left(e^{-\frac{|\vec{x}_1 - \vec{x}_2|^2}{\sigma}} + e^{-\frac{|\vec{x}_1 - \vec{x}_3|^2}{\sigma}} + e^{-\frac{|\vec{x}_2 - \vec{x}_3|^2}{\sigma}} \right), \quad (18)$$

y su deseo para llegar al estado terminal z_T por

$$g(x) = |x - z_T|^2. \quad (19)$$

¿Cómo elegimos el mejor control?

Definimos el costo empezando en un estado x y tiempo t usando el control α como

$$J(t, x, \alpha) = \mathbb{E} \left[\int_t^T f(s, X_s^{t,x,\alpha}, \alpha_s) ds + g(X_T^{t,x,\alpha}) \right]. \quad (20)$$

Y la función de valor como

$$V(t, x) = \sup_{\alpha \in \mathcal{A}[t, T]} J(t, x, \alpha). \quad (21)$$

Entonces, del principio de programación dinámica se establece que esta debe cumplir la ecuación de Hamilton-Jacobi-Bellman

$$\begin{aligned} \frac{\partial V}{\partial t} + \inf_{a \in A} \{ \mathcal{L}^a[V](t, x) + f(t, x, a) \} &= 0 \\ V(T, x) &= g(x), \end{aligned} \quad (22)$$

En este caso la ecuación a resolver es

$$\frac{\partial V}{\partial t} + \nu \Delta V - \lambda |\nabla V|^2 + F(t, x) = 0, \quad (23)$$

sujeto a la condición terminal

$$V(T, x) = g(x). \quad (24)$$

Si la resolvemos, podemos acceder a controles óptimos a través de $\hat{\alpha}(t, x) = -\sqrt{\lambda} D_x V(t, x)$.

Sin embargo, esta ecuación es no lineal y está definida en \mathbb{R}^6 , por lo que no podemos usar métodos como diferencias finitas. Por otro lado, existe una representación probabilística de la solución

$$V(t, x) = -\frac{\nu}{\lambda} \ln \left(\mathbb{E} \left[\exp \left(-\frac{\lambda}{\nu} g(x + \sqrt{2\nu} W_{T-t}) - \frac{\lambda}{\nu} \int_t^T F(s, x + \sqrt{2\nu} W_{s-t}) ds \right) \right] \right). \quad (25)$$

Usemos el método Deep BSDE. En este marco el proceso hacia adelante tendría drift y volatilidad dadas por

$$\mu(t, x) = 0 \quad \sigma(t, x) = \sqrt{2\nu} \mathbb{I}_{6 \times 6} \quad (26)$$

y la parte no lineal es

$$f(t, x, V(t, x), \sigma(t, x)' D_x V(t, x)) = -\lambda |D_x V|^2 + F(t, x). \quad (27)$$

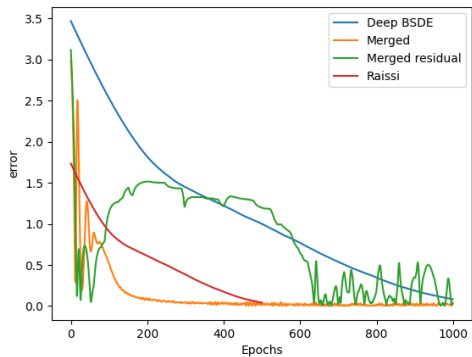


Figure: Curvas de entrenamiento

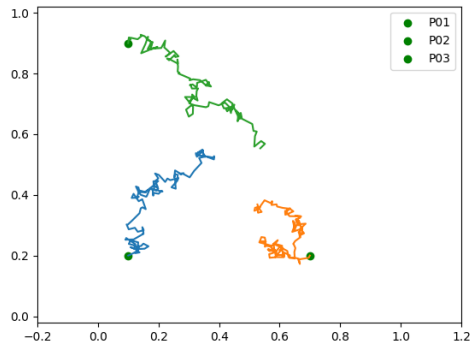


Figure: Ejemplo de trayectoria óptima

Ahora, consideremos problemas en dominio acotados Ω

$$\begin{aligned}\frac{\partial u}{\partial t}(t, x) + \mathcal{L}_t u(t, x) + f(t, x, u(t, x), \sigma(t, x)' D_x u(t, x)) &= 0 \\ u(T, x) &= g(x),\end{aligned}\tag{28}$$

con condiciones de frontera de Dirichlet y Neumann

$$u(t, x) = h_d(x) \quad \forall x \in \Gamma_D \quad \forall t \in [0, T] \tag{29}$$

$$\frac{\partial u}{\partial n}(t, x) = h_n(x) \quad \forall x \in \Gamma_N \quad \forall t \in [0, T], \tag{30}$$

Una forma muy general de resolver PDE's es forzando todo en el costo de la red. Esto se conoce como el método de Deep Galerkin o Physics Informed Neural Networks (Sirignano and Spiliopoulos 2018). En este caso, aproximamos la solución v con una red neuronal φ y la entrenamos con el costo

$$\begin{aligned}\mathcal{L}(\varphi) &:= \alpha_{int} \mathcal{L}_{\text{DGM,int}}(\varphi) + \alpha_T \mathcal{L}_{\text{DGM,T}}(\varphi) + \alpha_d \mathcal{L}_{\text{DGM,d}}(\varphi) + \alpha_n \mathcal{L}_{\text{DGM,n}}(\varphi) \\ &:= \alpha_{int} \left\| \frac{\partial \varphi}{\partial t} + \mathcal{L}\varphi + f(t, x, \varphi(t, x), \sigma(t, x)' D_x \varphi(t, x)) \right\|_{[0,T] \times \Omega, \nu_1}^2 \\ &\quad + \alpha_T \|\varphi(T, x) - g(x)\|_{\Omega, \nu_2}^2, \\ &\quad + \alpha_d \|\varphi(t, x) - h_d(x)\|_{[0,T] \times \partial \Gamma_D, \nu_3}^2 \\ &\quad + \alpha_n \|\hat{n} \cdot D_x \varphi(t, x) - h_n(x)\|_{[0,T] \times \partial \Gamma_N, \nu_4}^2\end{aligned}\tag{31}$$

donde todas las derivadas se calculan con diferenciación automática.

Podemos combinarlo con el método Deep BSDE usando el **costo de difusión** (Nüsken and Richter 2023)

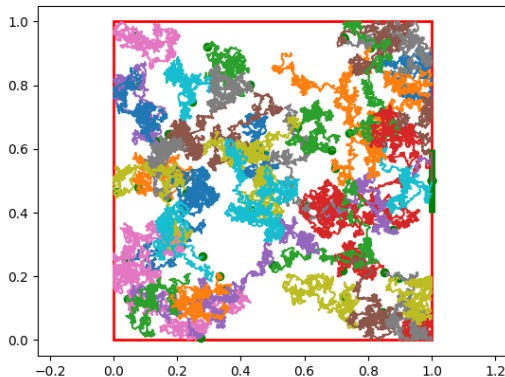
$$\mathcal{L}_{\text{diff}}^t(\varphi) = \alpha_{\text{int}} \mathcal{L}_{\text{diff,int}}^t(\varphi) + \alpha_T \mathcal{L}_{\text{diff,T}}^t(\varphi) + \alpha_d \mathcal{L}_{\text{diff,d}}^t(\varphi) + \alpha_n \mathcal{L}_{\text{diff,n}}^t(\varphi) \quad (32)$$

donde

$$\begin{aligned} \mathcal{L}_{\text{diff,int}}^t(\varphi) &= \mathbb{E} \left[\left(\varphi(T, X_T) - \varphi(t_0, X_{t_0}) - \int_{t_0}^T \sigma^\top \nabla \varphi(s, X_s) \cdot dW_s \right. \right. \\ &\quad \left. \left. + \int_{t_0}^T f(s, X_s, \varphi(s, X_s), \sigma^\top \nabla \varphi(s, X_s)) ds \right)^2 \right], \\ \mathcal{L}_{\text{diff,T}}^t(\varphi) &= \mathbb{E} \left[\left(\varphi(T, X^{(T)}) - g(X^{(T)}) \right)^2 \right], \mathcal{L}_{\text{diff,d}}^t(\varphi) = \mathbb{E} \left[\left(\varphi(t^d, X^d) - h_d(t^d, X^d) \right)^2 \right], \\ \mathcal{L}_{\text{diff,n}}^t(\varphi) &= \mathbb{E} \left[\left(D_x \varphi(t^n, X^n) - h_n(t^n, X^n) \right)^2 \right]. \end{aligned} \quad (33)$$

Ejemplo: Control estocástico

Este problema modela una partícula que quiere salir de una habitación por la puerta



Consideremos un caso de control LQR con ecuación HJB

$$\frac{\partial V}{\partial t} + \nu \Delta V - \lambda |\nabla V|^2 + 1 = 0, \quad (34)$$

sujeto a las condiciones de Diriclet y Neumann

$$V(t, x) = 0 \quad \text{for } x \in \partial\Omega_d, \quad (35)$$

$$\frac{\partial V}{\partial \vec{n}}(t, x) = 0 \quad \text{for } x \in \partial\Omega_n, \quad (36)$$

con condición terminal

$$V(1.0, x) = 0 \quad \text{for } x \in \Omega. \quad (37)$$

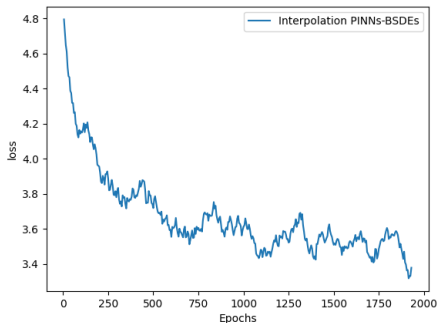


Figure: Curva de entrenamiento Interpolación

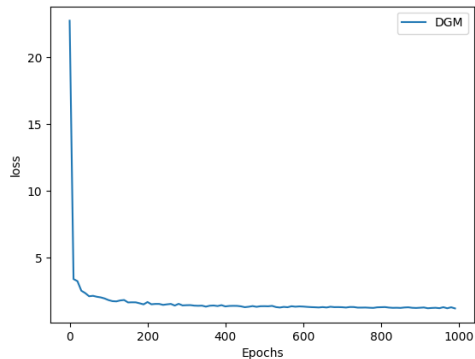
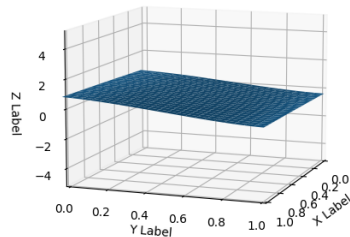
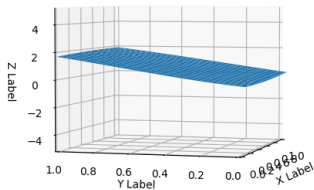


Figure: Curva de entrenamiento DGM



¿Porqué no para 3 partículas?

No se puede aplicar en el caso que estudiamos ya que

1. Samplear en el interior del dominio en \mathbb{R}^6 no captura completamente las interacciones entre partículas.
2. Los gradientes calculados con diferenciación automática no son confiables si no se aproximan directamente.
3. En varias dimensiones el tiempo de parada en el método de interpolación no es práctico de calcular.
4. Las redes de cada método son bastante inestables en su entrenamiento.

Como alternativa podríamos usar la versión del teorema de Feynmann-Kac con caminos reflejados y parados en la condición de Dirichlet, modificando el costo terminal respectivo.

Consideremos un juego con N agentes que se mueven según la dinámica (Han and Hu 2020)

$$\begin{aligned} dX_t^i &= \mu^i(t, \mathbf{X}_t, \alpha_t)dt + \sigma^i(t, \mathbf{X}_t, \alpha_t)dW_t^i + \sigma^0(t, \mathbf{X}_t, \alpha_t)dW_t^0 \\ X_0^i &= x_0^i \quad i \in \mathcal{I}, \end{aligned} \tag{38}$$

En donde cada uno quiere minimizar su propio costo

$$J^i(t, x, \alpha_t) := \mathbb{E} \left[\int_t^T f^i(t, \mathbf{X}_t, \alpha_t)dt + g^i(\mathbf{X}_T) \middle| X_t^i = x \right]. \tag{39}$$

¿Qué debería hacer cada agente para optimizar su costo? Se vuelve un problema de teoría de juegos. Definimos un equilibrio de Nash para el problema como un conjunto de estrategias α tal que para todo $i \in \mathcal{I}$ y cualquier $\beta^i \in \mathcal{A}^i$ tenemos

$$J^i(\alpha) \leq J^i(\alpha^{1,*}, \dots, \alpha^{i-1,*}, \beta^i, \alpha^{i+1,*}, \dots, \alpha^{N,*})$$

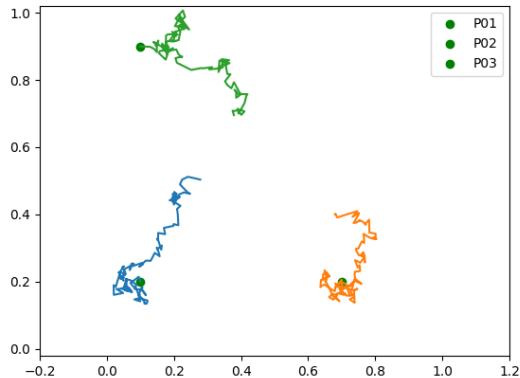
Para hallar este equilibrio vamos a optimizar cada agente conociendo lo mejor que harían los demás. Vamos a hacer esto iterativamente hasta llegar a una aproximación del equilibrio. En cada paso vamos a resolver la HJB

$$\frac{\partial V^{i,m+1}}{\partial t} - \lambda |\nabla_i V^{i,m+1}|^2 + 2\sqrt{\lambda} \alpha^{-i,m} \cdot \nabla_{-i} V^{i,m+1} + F(t, \mathbf{X}) + \nu \Delta V^{i,m+1} = 0, \quad (40)$$

Es la misma ecuación que resolvimos con Deep BSDE!

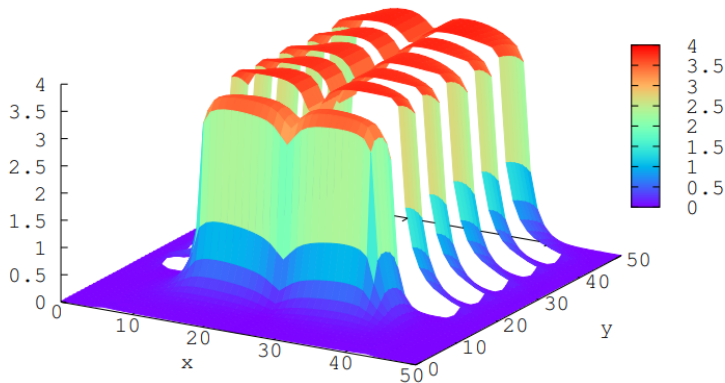
$$\mu^i(t, \mathbf{X}) = 2\sqrt{\lambda} \begin{bmatrix} \alpha^1(t, \mathbf{X}) \\ \vdots \\ \alpha^{i-1}(t, \mathbf{X}) \\ 0 \\ \alpha^{i+1}(t, \mathbf{X}) \\ \vdots \\ \alpha^N(t, \mathbf{X}) \end{bmatrix} \quad \sigma^i(t, \mathbf{X}) = \sqrt{2\nu} \mathbb{I}_{2N \times 2N} \quad (41)$$

Después de 50 iteraciones del algoritmo obtenemos la siguiente trayectoria óptima



- Las redes neuronales permiten parametrizar funciones en espacios de alta dimensión. Esto las hace buenas candidatas para resolver ecuaciones diferenciales con estas características.
- Lamentablemente , estas no son muy estables ni robustas

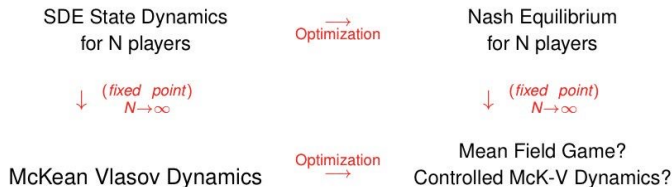
Nuestra idea original era usar estos métodos para resolver problemas de muchos agentes en dominios acotados para examinar las diferencias con soluciones de campo medio. (Achdou and Laurière 2020)



Y eventualmente estudiar numéricamente este cuadro


(INFORMAL) NATURAL QUESTION




Is the diagram



commutative?

-  Achdou, Yves and Mathieu Laurière (2020). “Mean Field Games and Applications: Numerical Aspects”. In: Achdou, Yves et al. *Mean Field Games*. Ed. by Pierre Cardaliaguet and Alessio Porretta. Vol. 2281. Series Title: Lecture Notes in Mathematics. Cham: Springer International Publishing, pp. 249–307. ISBN: 978-3-030-59836-5 978-3-030-59837-2. DOI: 10.1007/978-3-030-59837-2_4. URL: http://link.springer.com/10.1007/978-3-030-59837-2_4 (visited on 02/17/2023).
-  Chan-Wai-Nam, Quentin, Joseph Mikael, and Xavier Warin (Dec. 10, 2018). *Machine Learning for semi linear PDEs*. arXiv: 1809.07609 [cs, math, stat]. URL: <http://arxiv.org/abs/1809.07609> (visited on 03/26/2023).

-  Han, Jiequn and Ruimeng Hu (Aug. 16, 2020). “Deep Fictitious Play for Finding Markovian Nash Equilibrium in Multi-Agent Games”. In: *Proceedings of The First Mathematical and Scientific Machine Learning Conference*. Mathematical and Scientific Machine Learning. ISSN: 2640-3498. PMLR, pp. 221-245. URL: <https://proceedings.mlr.press/v107/han20a.html> (visited on 02/28/2023).
-  Han, Jiequn, Arnulf Jentzen, and Weinan E (Aug. 21, 2018). “Solving high-dimensional partial differential equations using deep learning”. In: *Proceedings of the National Academy of Sciences* 115.34, pp. 8505-8510. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1718942115. URL: <https://pnas.org/doi/full/10.1073/pnas.1718942115> (visited on 09/26/2022).

-  Nüsken, Nikolas and Lorenz Richter (Jan. 29, 2023). *Interpolating between BSDEs and PINNs: deep learning for elliptic and parabolic boundary value problems*. arXiv: 2112.03749 [cs, math, stat]. URL: <http://arxiv.org/abs/2112.03749> (visited on 03/26/2023).
-  Pham, Huyền (2009). *Continuous-time Stochastic Control and Optimization with Financial Applications*. Vol. 61. Stochastic Modelling and Applied Probability. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-89499-5 978-3-540-89500-8. DOI: 10.1007/978-3-540-89500-8. URL: <https://link.springer.com/10.1007/978-3-540-89500-8> (visited on 02/28/2023).
-  Raissi, Maziar (Apr. 19, 2018). *Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations*. arXiv: 1804.07010 [cs, math, stat]. URL: <http://arxiv.org/abs/1804.07010> (visited on 03/22/2023).



Sirignano, Justin and Konstantinos Spiliopoulos (Dec. 2018). “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of Computational Physics* 375, pp. 1339–1364. ISSN: 00219991. DOI: 10.1016/j.jcp.2018.08.029. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0021999118305527> (visited on 10/10/2022).

The End