Notas

Carlos Daniel Contreras Quiroz *March 12, 2023*

To do list

1. Entender BSDE
2. Escribir conexión PDE-BSDE
3. Programar reflexión en frontera
4. Deep Fictious Play
5. Escribir Crowd motion
6. Simular diferencias
7. Escribir simulaciones
8. Escribir apendice Neural Networks

Problemas

1. El repositorio del review corre muy lento ✔
2. Las condiciones de frontera no son iguales
   - Reflejar los caminos puede funcionar ✗
3. Parece que DeepBSDE no funciona con dimensiones de más de 100
   - Podría usarse el deep Backward de Pham [1], se puede entrenar basado en modelos anteriores. No hay código
   - O tambien deep splitting de Beck [2]. Si hay código

Ideas

- Hacerlo en flux
- Hacerlo en equinox/jax
- Buscar un modelo de Cucker Smale abierto

Preguntas

1. £En que sentido converge a la solución?
2. £Qué pasaría si se cambia la discretización de Euler por una mejor?£Afecta el modo de convergencia?

Probabilidad

Preguntas

- Es $E[\int_0^T |f|^2] < \infty$ equivalente a que $\int_0^T |f|^2 < \infty$ a.s?

Para recordar

Recordar: Filtración aumentada

A una filtración $\mathbb{F} = (\mathcal{F}_t)_t$ en un espacio de probabilidad $(\Omega, \mathcal{A}, \mathbb{P})$ le corresponde una filtración continua a la derecha $\mathbb{F}^+ = (\mathcal{F}_t)_t = \cap_{t<s}(\mathcal{F}_s)$. $\mathbb{F}$ se dice continua a la derecha si $\mathbb{F}^+ = \mathbb{F}$.
Sea $\mathcal{N}_P = \{A \subseteq \omega | A \subseteq B$ para algún $B \in \mathbb{F}$ con $P(B) = 0\}$. $\mathbb{F}$ se dice completa si $\mathcal{F}_t$ contiene $\mathcal{N}_P$ para todo t.
Una filtración continua a la derecha y completa se llama filtración aumentada.

Deep BSDE

Vamos a intentar resolver ecuaciones del tipo

$$\begin{aligned} &\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\operatorname{Tr}\left(\sigma\sigma^{\mathrm{T}}(t,x)\left(\operatorname{Hess}_x u\right)(t,x)\right) + \nabla u(t,x)\cdot\mu(t,x) \\ &\quad + f\left(t,x,u(t,x),\sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\right) = 0 \end{aligned} \tag{1}$$

con la condición final $u(T,x) = g(x)$.

Vamos a realizar una aproximación con la fórmula de Feynman-Kac. Esto es, la solución de la ecuación anterior viene dada por

$$\begin{aligned} & u\left(t, X_t\right) - u\left(0, X_0\right) \\ = & -\int_0^t f\left(s, X_s, u\left(s, X_s\right), \sigma^{\mathrm{T}}\left(s, X_s\right)\nabla u\left(s, X_s\right)\right) ds \\ & + \int_0^t \left[\nabla u\left(s, X_s\right)\right]^{\mathrm{T}} \sigma\left(s, X_s\right) dW_s. \end{aligned} \tag{2}$$

donde X resuelve la ecuación diferencial estocástica

$$X_t = \xi + \int_0^t \mu\left(s, X_s\right) ds + \int_0^t \sigma\left(s, X_s\right) dW_s \tag{3}$$

Vamos a estudiar primero el caso de HJB del repositorio. Acá intentamos controlar el proceso 100-dimensional

$$dX_t = 2\sqrt{\lambda} m_t dt + \sqrt{2} dt \quad X(0) = x \quad t \in (0,T) \tag{4}$$

a través del control m_t, con el funcional de costo

$$J(m_t) = \mathbb{E}\left[\int_0^T ||m_t||^2 dt + g(X_T)\right]. \tag{5}$$

Recordar: Hamilton-Jacobi-Bellman

Para un proceso controlado

$$\begin{aligned} &dX_x = \mu(X,u)dt + \sigma(X,u)dW \\ &X(0) = x \end{aligned}$$

con función de costo

$$J(t,x,u) = \mathbb{E}\left[\int_t^T f(X_x,u)dt + g(X_x)\right].$$

la ecuación de HJB para la función de valor

$$V(t,x) = \sup_{u\in\mathcal{A}} J(t,x,u)$$

es

$$\frac{\partial V}{\partial t} + \sup_{u\in A}\{\mathcal{L}^u(V) + f(x,u)\} = 0$$

donde

$$\mathcal{L}^u(V)(t,x) = \mu(x,u)\cdot\nabla V(t,x) + \frac{1}{2}Tr(\sigma(t,x)\sigma^T(t,x)\nabla^2 V(t,x))$$

con la condición final

$$u(T,x) = g(x)$$

Esto también puede escribirse con el Hamiltoniano

$$H(t,x,p,M) = \sup u \in \mathcal{A}\{\mu(x,u)\cdot p + \frac{1}{2}Tr[\sigma(t,x)\sigma^T(t,x)M] + f(x,u)\}$$

y asumiendo que el control no se aplica a la volatilidad se escribe

$$\frac{\partial u}{\partial t} + H(t,x,\nabla u) + \frac{1}{2}Tr[\sigma(t,x)\sigma^T(t,x)\nabla^2 u] = 0$$

```
import numpy as np

def incmatrix(genl1,genl2):
m = len(genl1)
n = len(genl2)
M = None #to become the incidence matrix
VT = np.zeros((n*m,1), int)  #dummy variable

#compute the bitwise xor matrix
M1 = bitxormatrix(genl1)
M2 = np.triu(bitxormatrix(genl2),1)

for i in range(m-1):
for j in range(i+1, m):
[r,c] = np.where(M2 == M1[i,j])
for k in range(len(r)):
VT[(i)*n + r[k]] = 1;
VT[(i)*n + c[k]] = 1;
VT[(j)*n + r[k]] = 1;
VT[(j)*n + c[k]] = 1;

if M is None:
M = np.copy(VT)
else:
M = np.concatenate((M, VT), 1)

VT = np.zeros((n*m,1), int)

return M
```

References

[1] Côme Huré, Huyên Pham, and Xavier Warin. "Deep backward schemes for high-dimensional nonlinear PDEs". In: *Mathematics of Computation* 89.324 (Jan. 31, 2020), pp. 1547–1579. ISSN: 0025-5718, 1088-6842. DOI: 10.1090/mcom/3514. URL: https://www.ams.org/mcom/2020-89-324/S0025-5718-2020-03514-5/ (visited on 02/28/2023).

[2] Christian Beck et al. "Deep Splitting Method for Parabolic PDEs". In: *SIAM Journal on Scientific Computing* 43.5 (Jan. 2021), A3135–A3154. ISSN: 1064-8275, 1095-7197. DOI: 10.1137/19M1297919. URL: https://epubs.siam.org/doi/10.1137/19M1297919 (visited on 02/14/2023).