Numerical Methods for Mathematical Finance

Matteo Garbelli

November 2024

1 Course Outline

Lecture 1: Brownian Motions and Geometric BMs

- Brownian Motion (BMs) and Geometric Brownian Motion (GBMs)
- Simulations of BMs, GBMs and Correlated GBMs

Lecture 2 : Numerical Methods for SDEs

1. Discretization Methods for SDEs:
   - Euler-Maruyama Method
   - Milstein Method
2. SDE Simulations of:
   - Ornstein–Uhlenbeck process
   - Cox–Ingersoll–Ross (CIR) process
   - Heston Model

Lecture 3: Pricing

1. Real-World Measure $\mathbb{P}$ vs Risk-Free Measure $\mathbb{P}$
2. Black Scholes Equation and Feynman–Kac Theorem
3. Monte Carlo Simulation:
   - Generation of Stochastic Paths
   - Applications in Option Valuation

Lecture 4: Advanced Topics and Assignment of Projects

1. Energy markets
2. Machine Learning for Finance
3. Reinforcement Learning
4. Example Project: Application of Reinforcement Learning in Energy Markets.
5. Assignment of Projects

List of Potential Projects

1. Neural SDE
2. Deep Learning Methods for solving BSDEs
3. Fourier Series Cosine Expansion (COS method) for Pricing (Section 6 of [1])
4. Entropic Risk Minimization for Multi-asset Portfolio Optimization via Neural Networks
5. Stochastic Volatility Models besides Heston Model (e.g. Local Volatility Model)
6. Simulation of Jump Processes (Lévy Processes) and Feynman-Kac theorem for jump diffusion process (Section 5 of [1])
7. Energy Markets: Modeling and Pricing Carbon Credits with Jump Diffusion
8. FinTech: Reinforcement Learning for Optimal Trading Strategies

References

[1] C.W. Oosterlee and L.A. Grzelak - Mathematical Modeling and Computation in Finance: With Exercises and Python and MATLAB Computer Codes", World Scientific Publishing, 2019.

2 Projects Details

1. **Calibration of Neural SDEs**

   - **Objective**: Explore Neural SDEs for modeling financial data, integrating deep learning into SDEs to model time series with uncertainty.
   - **Mathematical Framework**: Start with basic SDEs of the form

     $$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dW_t$$

     and extend to Neural SDEs by parameterizing μ and σ with neural networks.
   - **Coding Reference**: Use PyTorch or TensorFlow for neural networks and stochastic processes;

     PyTorch-based packages for NeuralSDE.

     Repository: `https://github.com/google-research/torchsde`
   - **Paper Reference**:

     - Neural SDEs as Infinite-Dimensional GANs by Patrick Kidger, James Foster, Xuechen Li, Terry J Lyons Proceedings of the 38th International Conference on Machine Learning, PMLR 139:5453-5463, 2021 `https://proceedings.mlr.press/v139/kidger21b.html`

     - 'Neural Ordinary Differential Equations" by Chen et al. (2018).

2. **Deep Learning Methods for SDEs and Backward SDEs (BSDEs)**

- **Objective**: Implement deep learning models for solving BSDEs, used to represent complex financial derivatives.
- **Mathematical Framework**: Given a BSDE of the form
$$Y_t = \xi + \int_t^T f(s, Y_s, Z_s) ds - \int_t^T Z_s dW_s$$
approximate the solution (Y, Z) with neural networks.
- **Coding Reference**: PyTorch or TensorFlow for implementing the deep BSDE solver, with forward-backward neural networks.
- **Paper Reference**:
"Deep BSDEs for High-Dimensional PDEs and Their Applications" by E et al. (2017)
"Backward Stochastic Differential Equations and Optimal Control" by Pham (2009).
"Backward Stochastic Differential Equations in Finance" by N. El Karoui, S. Peng, M. C. Quenez (2002)

3. **Fourier Series Cosine Expansion (COS) Method for Pricing**

- **Objective**: Use the COS method, a Fourier-based approach, for pricing options in models where characteristic functions are known.
- **Mathematical Framework**: The COS method decomposes the option price as a sum of cosine basis functions. For an option price V, we compute it as
$$V = e^{-rT} \sum_{k=0}^{N-1} Re \left[\varphi \left(\frac{k\pi}{b-a} \right) \right] f_k$$
, where f_k are Fourier coefficients.
- **Coding Reference**: Implement in Python, using libraries like SciPy for numerical integration.
- **Paper Reference**: "The COS Method for Option Pricing" by Fang and Oosterlee (2008).

4. **Entropic Risk Minimization for Multi-asset Portfolio Optimization via Neural Networks**

- **Objective**: Investigate entropic risk minimization techniques for optimizing a multi-asset portfolio by leveraging neural networks. The project aims to integrate deep learning methodologies to enhance portfolio management under risk constraints.
- **Mathematical Framework**: Formulate the portfolio optimization problem with an entropic risk measure, defined as:
$$\rho(X) = \frac{1}{\eta} \log \mathbb{E}\left[e^{-\eta X} \right],$$
where η represents the risk aversion parameter, and X is the portfolio return. Use neural networks to approximate optimal portfolio weights by learning the underlying distributional properties and minimizing the entropic risk measure across multiple assets.

- **Coding Reference**: Employ PyTorch or TensorFlow for developing neural network architectures to model entropic risk measures and perform portfolio optimization. Consider leveraging libraries such as PyPortfolioOpt for basic portfolio calculations and build custom neural architectures to address multi-asset optimization challenges.

  Repository: `https://github.com/robertmartin8/PyPortfolioOpt`

- **Paper Reference**:

  - Entropic Risk Minimization and Portfolio Selection by *Hirschberger, Steuer, and Utz.* European Journal of Operational Research, 2007. `https://doi.org/10.1016/j.ejor.2005.12.017`

  - Deep Learning for Finance: Deep Portfolios by *Huang, Zhong, and Chen.* Journal of Financial Data Science, 2020. `https://jfds.pm-research.com/content/2/2/28`

  - "Neural Networks for Portfolio Management" by Heaton et al. (2017).

5. **Stochastic Volatility Models (Beyond Heston Model)**

- **Objective**: Implement models such as the Local Volatility Model

$$\begin{aligned} dS_t &= rS_t\,dt + b(V_t)\,\sigma_{LSV}(S_t,t)S_t\,dW_t, \\ dV_t &= a(V_t,t)\,dt + c(V_t,t)\,dZ_t, \\ dW_t\,dZ_t &= \rho\,dt. \end{aligned}$$

  or the Stochastic Alpha Beta Rho (SABR) model, which are alternatives to Heston for stochastic volatility.

- **Mathematical Framework**: The Local Volatility Model derives volatility as a deterministic function of stock price and time, while the SABR model represents volatility with another SDE.

- **Coding Reference**: QuantLib for stochastic modeling or custom simulations.

- **Paper Reference**: "A Local Volatility Model" by Dupire (1994) and "The SABR Model in Illiquid Markets" by Hagan et al. (2002).

6. **Simulation of Jump Processes (Lévy Processes) and Pricing**

- **Objective**: Simulate jump-diffusion processes (e.g., Merton model) and price derivatives in models incorporating jumps, like the Kou and Variance Gamma models.

- **Mathematical Framework**: Start with jump-diffusion SDEs,

$$dX_t = \mu dt + \sigma dW_t + dJ_t$$

  , where J_t is a jump process (e.g., Poisson-driven).

- **Coding Reference**: Python packages like NumPy and SciPy for simulations; PyMC for stochastic processes.

- **Paper Reference**: "Option Pricing when Jump-Diffusion Processes are Used" by Merton (1976).

7. **Energy Markets: Modeling and Pricing Carbon Credits with Jump Diffusion**

    - **Objective**: Model and price carbon credits in energy markets, incorporating jumps due to policy changes or regulatory shifts.
    - **Mathematical Framework**: Use jump-diffusion models or Markov-modulated jump processes to simulate carbon price dynamics.
    - **Coding Reference**: Python libraries like QuantLib or custom implementations for pricing.
    - **Paper Reference**: "The Pricing of Emission Allowances in the Presence of Market Frictions" by Carmona et al. (2009).

8. **FinTech: Reinforcement Learning for Optimal Trading Strategies**

    - **Objective**: Develop reinforcement learning (RL) algorithms for optimal trading strategies.
    - **Mathematical Framework**: Formulate as a Markov Decision Process, where the RL agent maximizes reward by learning an optimal trading policy.
    - **Coding Reference**: Use RL libraries such as Stable-Baselines3 (Python) for implementing trading strategies.
    - **Paper Reference**: "Deep Reinforcement Learning in High Frequency Trading" by Zhang et al. (2020).