Solving High-Dimensional PDEs

Deep Galerkin Method with Timestepping

Ray Wu and Christina C. Christara

University of Toronto
Department of Computer Science

May 27, 2022

Overview

Overview of the talk:

- Introduction and motivation, problem description.
- One- and multi-dimensional Black-Scholes PDEs.
- Deep Galerkin Method (DGM).
- Deep Galerkin Method with Timestepping (DGMT).
- Numerical results.
- Error analysis.
- Comparison of DGM and DGMT.
- Convergence discussion.
- Conclusions and future work.

Introduction

Why are we interested in solving high-dimensional PDEs?

- Many examples, but we are specifically interested in pricing multi-asset options accurately.
- Some options can have hundreds of underlying assets, each asset giving rise to a spatial dimension.

Curse of Dimensionality

The *curse of dimensionality* refers to the problem that the complexity of the numerical method scales exponentially with the dimension. With N gridpoints per dimension, there are N^d unknowns in total. Traditional PDE methods such as Finite Difference Methods (FDMs) and Finite Element Methods (FEMs) suffer from this problem.

Dimensions	1	2	3	4
Unknowns	64	4096	262144	16777216
Execution Time	<0.01 sec	0.1 sec	15 sec	28 min

Table 1: Exponential increase in runtime of an ADI method as dimensions increase.

One-dimensional Black-Scholes PDE

Before showing high-dimensional [Black and Scholes, 1973] PDEs, we first show the one-dimensional case, given by

$$V_\tau = \mathcal{L}V \equiv \frac{\sigma^2 S^2}{2} V_{SS} + (r-q)SV_S - rV. \tag{1}$$

Note that subscripts denote partial derivatives, and

- S denotes the stock price,
- τ denotes the reverse time counted from expiry time T ($\tau = T - t$, t forward time),
- σ denotes the volatility of the stock,
- r denotes the risk-free interest rate,
- q denotes dividend yield of the stock,
- V denotes the unknown option price we are solving for.

We are interested in the option values at $\tau = T$. Payoffs denoted by $V^*(S)$ correspond to initial conditions:

- Call payoff: $V(0,S) = V^*_{\text{call}}(S) \equiv \max(S-K,0)$,
- Put payoff: $V(0,S) = V^*_{\text{put}}(S) \equiv \max(K-S,0)$.

Multi-dimensional Black-Scholes

In d-dimensions, the analogous PDE to Equation (1) is

$$V_\tau = \mathcal{L}V \equiv \frac{1}{2}\sum_{i,j=1}^{d} \rho_{i,j}\sigma_i\sigma_j S_i S_j V_{S_i,S_j} + \sum_{i=1}^{d}(r - q_i)S_i V_{S_i} - rV. \tag{2}$$

Note that

- S_i, σ_i, q_i denote the stock price, volatility, and dividend yield of the i-th stock
- $\rho_{i,j}$ denotes the correlation between S_i and S_j. Must be 1 if $i = j$, and less than 1 in absolute value otherwise.

Many different payoffs, but we use Geometric Average Put, given by

$$V(0, S_1, S_2, \ldots, S_d) = \max(K - (\prod_{i=1}^{d} S_i)^{1/d}, 0) \tag{3}$$

Useful because a corresponding one-dimensional problem exists.

Geometric Average Put

The Geometric Average Put problem has the identical option price as a corresponding one-dimensional Put problem [Birge and Linetsky, 2007], with the interest rate r remaining unchanged and adjusted parameters $\hat{\sigma}$ and $\hat{q}$ given by

$$\hat{\sigma} = \frac{1}{d}\sqrt{\sum_{i,j=1}^{d} \rho_{i,j}\sigma_i\sigma_j}$$

$$\hat{q} = \frac{1}{d}\sum_{i=1}^{d}(q_i + \frac{1}{2}\sigma_i^2) - \frac{1}{2}\hat{\sigma}^2$$

In our examples we do not have dividend yield in multidimensional problems. In other words, $q_i = 0$.

Introduction to Deep Learning Methods

Deep Learning addresses the curse of dimensionality in PDEs, for example

- Black-Scholes equations [Grohs et al., 2018], and
- Semilinear heat equations [Hutzenthaler et al., 2020]

have been proven to be able to be approximated by neural networks to arbitrary accuracy, with the complexity of the neural network being a polynomial function of both the dimension and the inverse of a prespecified accuracy.

- Current research studies how to pose the problem and how to design the neural network such that it is easy for the neural network to solve.
- We mainly consider [Sirignano and Spiliopoulos, 2018]'s Deep Galerkin Method (DGM).
- A related and popular method is [Han et al., 2018]'s Deep BSDE method.
  - Compared to DGM, solves a narrower range of problems.
  - Some finance problems that have nonlinearity in the V_{SS} (diffusion) term cannot be solved by Deep BSDE.

The Deep Galerkin Method (DGM)

The DGM uses a neural network $f(\tau, x; \theta)$ to approximate the unknown function over the entire time and space domain.

The problems DGM solves are of the form

$$\begin{aligned} &u_\tau(\tau, x) = \mathcal{L}u(\tau, x) \text{ where } (\tau, x) \in [0, T] \times \Omega \\ &u(\tau = 0, x) = u_0(x) \\ &u(\tau, x) = g(\tau, x) \text{ for } x \in \partial\Omega. \end{aligned} \tag{4}$$

This is a generalization of the Black-Scholes equation, since $\mathcal{L}$ here is possibly a nonlinear operator.

[Sirignano and Spiliopoulos, 2018] solve the optimization problem

$$\begin{aligned} \theta^* = \arg\min_\theta \{ G(\tau, x; \theta) \equiv &\|f_\tau(\tau, x; \theta) - \mathcal{L}f(\tau, x; \theta)\|^2_{[0,T]\times\Omega,\nu_1} \\ &+ \|f(\tau, x; \theta) - g(\tau, x)\|^2_{[0,T]\times\partial\Omega,\nu_2} \\ &+ \|f(0, x; \theta) - u_0(x)\|^2_{\Omega,\nu_3} \} \end{aligned} \tag{5}$$

using Adaptive Moment Estimation (Adam) [Kingma and Ba, 2014].

Deep Galerkin Method with Timestepping (DGMT)

- In all parabolic problems, by nature, the values at one timestep depend on values on previous timesteps.
- We present an extension of the Deep Galerkin Method that incorporates timestepping (DGMT) and therefore abides by this property.
- Consider a time-discretization of the PDE (1) or (2):

$$(\mathcal{I} - \vartheta \Delta\tau \mathcal{L}) v_j = (\mathcal{I} + (1-\vartheta)\Delta\tau\mathcal{L}) v_{j-1} \tag{6}$$

- Key idea: Instead of using a neural network to approximate the PDE along the *entire* domain, approximate it at only one point in time, that is,

$$f_j(x;\theta) \equiv f(x;\theta_j) \approx V(\tau_j, S = x). \tag{7}$$

- Then, partition the time domain into subintervals. Using Equation (6) as an objective function, solve optimization problems until last timestep is reached.
- Another key idea: The parameters of the neural network at the previous timestep are a good initial guess for the parameters at the current timestep.

DGMT Algorithm

Algorithm 1 DGMT with ϑ-timestepping

1: Pick time points τ_j, $j = 0, ..., M$, with $\tau_0 = 0$ and $\tau_M = T$. Let $\Delta\tau_j = \tau_j - \tau_{j-1}$.
2: Initialize a neural network $f(x;\theta)$
3: Solve the optimization problem

$$\theta_1 = \arg\min_{\theta} \left(f(x;\theta) - \Delta\tau_1 \mathcal{L} f(x;\theta) - u_0(x) \right)^2 \tag{8}$$

where u_0 is the initial condition of the PDE problem.
4: **for** $j = 2, \ldots, M$ **do**
5: Solve the optimization problem

$$\theta_j = \arg\min_{\theta} \left(\left[f(x;\theta) - \vartheta \Delta\tau_j \mathcal{L} f(x;\theta) \right] - \left[f(x;\theta_{j-1}) + (1-\vartheta)\Delta\tau_j \mathcal{L} f(x;\theta_{j-1}) \right] \right)^2 \tag{9}$$

with θ_{j-1} as the initial guess for θ.
6: **end for**
7: $f(x;\theta_M)$ now approximates $V(\tau = T, S)$

Details about DGMT Algorithm

Choices of ϑ correspond to timestepping schemes:

- Crank-Nicolson (CN) timestepping: $\vartheta = 1/2$. Second-order convergent.
- Backwards Euler timestepping: $\vartheta = 1$. First-order convergent.
- We mostly use CN timestepping, but for the initial timestep we use Backwards Euler to avoid taking derivatives of a nonsmooth payoff function.

Number of Epochs:

- For the first optimization problem (8), we have a cold start, and a large number of epochs (4000) are used for convergence to a reasonable error.

$$\theta_1 = \arg\min_{\theta} \left(f(x;\theta) - \Delta\tau_1 \mathcal{L} f(x;\theta) - u_0(x) \right)^2$$

- For subsequent problems (9), θ_{j-1} at the previous timestep is already a good approximation (warm start), and much fewer iterations (500) are required.

$$\theta_j = \arg\min_{\theta} \left(\left[f(x;\theta) - \vartheta \Delta\tau_j \mathcal{L} f(x;\theta) \right] - \left[f(x;\theta_{j-1}) + (1-\vartheta)\Delta\tau_j \mathcal{L} f(x;\theta_{j-1}) \right] \right)^2$$

Timesteps: Uniform timesteps taken, but easily generalizable to variable timesteps.

Computational Results (One-dimensional problem)

M	Computed Value	Relative Error	time(s)
Exact solution	10.802266	—	—
4 timesteps	10.636006	1.54×10^{-2}	7.34×10^{2}
8 timesteps	10.742462	5.53×10^{-3}	1.07×10^{3}
16 timesteps	10.794114	7.50×10^{-4}	1.74×10^{3}
32 timesteps	10.803133	8.53×10^{-5}	3.02×10^{3}
64 timesteps	10.811709	8.79×10^{-4}	5.60×10^{3}
128 timesteps	10.800688	1.41×10^{-4}	1.09×10^{4}
original DGM	10.792760	8.75×10^{-4}	1.09×10^{4}

Table 2: Comparison of DGMT method with 4, 8, 16, 32, 64, and 128 timesteps. $S = K$, with $T = 1$, $\sigma = 0.4$, $K = 100$, $r = 0.1$. Neural network size held constant. DGMT computes a more accurate result than DGM for the same computational work.

Some similar properties with Finite Difference Methods:

- Error decreases with number of timesteps.
- Limitations to the accuracy as number of timesteps increase:
  - Approximation error of the neural network to the true function dominates the truncation error from timestepping.
  - Similar to only increasing the number of timesteps but not changing the size of the grid in a FDM.

Neural Network approximation limitations

The DGMT algorithm is limited by the ability of the neural network to approximate the solution function.

$$\text{total error} = \text{time-discretization error} + \text{neural network approximation error}. \quad (10)$$

Time-discretization error is the familiar second-order $\mathcal{O}(\Delta\tau^2)$ error of [Crank and Nicolson, 1947, Rannacher, 1984] timestepping.

Neural network approximation error, given by (11), is the ability of the neural network to approximate the solution function $V(\tau = T, \cdot)$:

$$|f(x = K; \theta) - V(\tau = T, S = K)| \quad (11)$$

For one-dimensional European options, we can measure the magnitude by directly solving the minimization problem

$$\theta^* = \arg\min_{\theta}(f(x; \theta) - V(\tau = T, S))^2. \quad (12)$$

Our experiments indicate that the quantity in (11) can be reduced to $\approx 10^{-4}$.

Time-convergence of DGMT

DGMT has $\mathcal{O}(\Delta\tau^2)$ time-discretization error, limited by the ability of the neural network to approximate the payoff function.

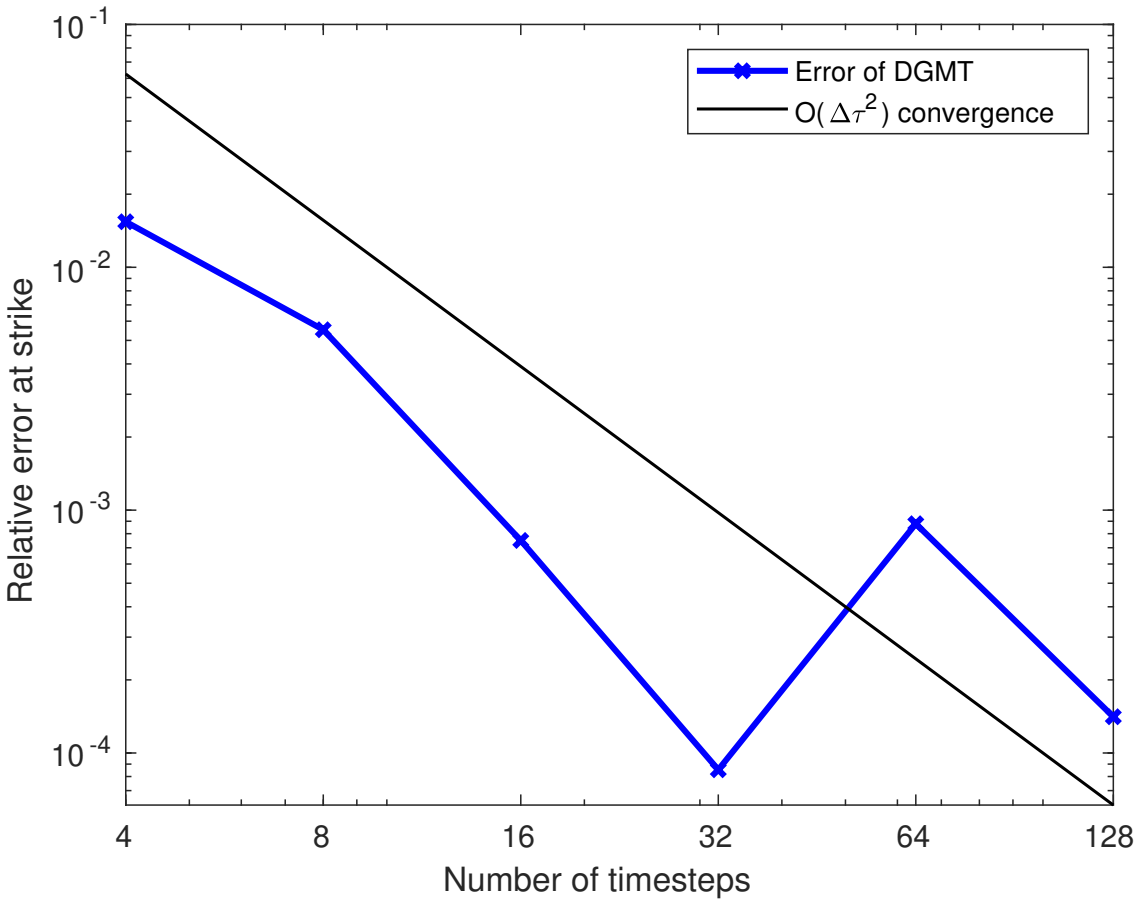
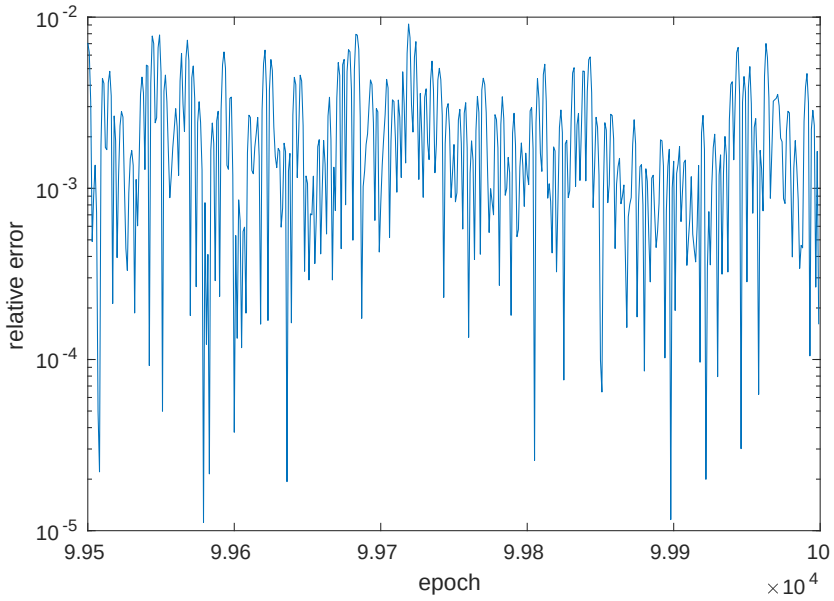


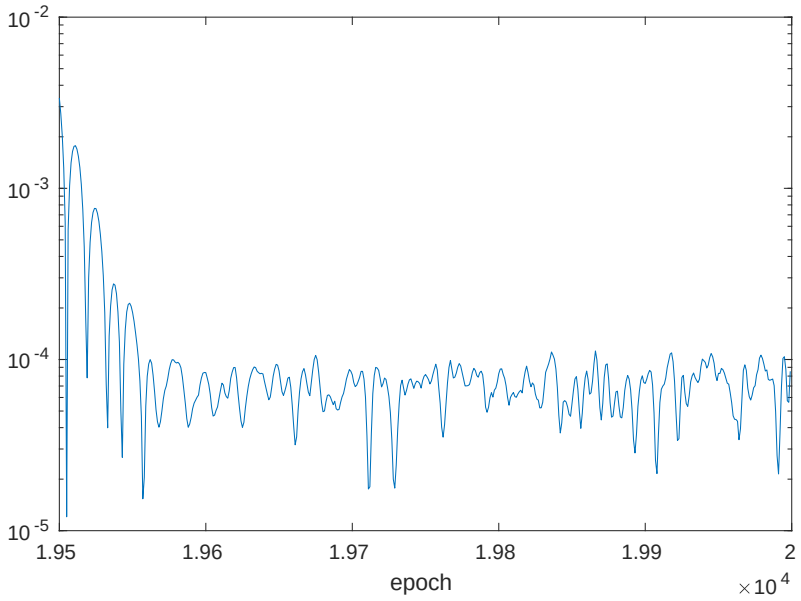
Figure 1: Convergence of DGMT method as $\Delta\tau$ is reduced.

Convergence of Neural Network

The DGMT exhibits a more consistent error per epoch compared to the DGM.



(a) DGM, last 500 epochs



(b) DGMT, last timestep

Figure 2: Comparison of relative errors of DGM and DGMT. As can be seen, DGMT has reduced fluctuations in the error with a much improved "worst case". Total iterations counted for DGMT.

Conclusions

Main points of the talk (main contributions):

- Time-discretization: Solving the time-discretized equations introduces direct dependence of solution values on previous timestep values.
- Order of convergence: Time-discretization leads to a second-order convergent scheme in time.
- Approximation limit of Neural Networks: When the error reaches the approximation limit, we observe stagnation.

The above are features commonly seen in standard FDM/FEM numerical methods. Additionally, we observe improved performance of DGMT:

- A more accurate solution is computed, even when original DGM is given more computational time.
- Stability of solution is greatly increased, and the computed results are generally more predictable (see Figure 2 on previous slide)

Future work

Future work:

- **Numerical results for multi-dimensional problems**: studying if the conclusions we drew from the one-dimensional case hold for the multi-dimensional problems.
- Development of a stopping criterion instead of a fixed number of training epochs.
- Extension of DGMT to nonlinear problems (e.g. American exercise rights, transaction cost models, passport trading options, etc).
- Study of neural network size vs. neural network approximation error. Is it possible to reduce the neural network approximation error? By how much, and at what rate?
- Exploring new designs for the neural network, because the DGM was designed to capture nonsmoothness around the initial conditions of the PDE. With timestepping, we avoid this problem of nonsmoothness.

References I

Birge, J. R. and Linetsky, V. (2007).
Handbooks in operations research and management science: Financial engineering.
Elsevier.

Black, F. and Scholes, M. (1973).
The pricing of options and corporate liabilities.
Journal of political economy, 81(3):637–654.

Crank, J. and Nicolson, P. (1947).
A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type.
In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 43, pages 50–67. Cambridge University Press.

Grohs, P., Hornung, F., Jentzen, A., and Von Wurstemberger, P. (2018).
A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations.
arXiv preprint arXiv:1809.02362.

References II

Han, J., Jentzen, A., and E, W. (2018).
Solving high-dimensional partial differential equations using deep learning.
Proceedings of the National Academy of Sciences, 115(34):8505–8510.

Hutzenthaler, M., Jentzen, A., Kruse, T., and Nguyen, T. A. (2020).
A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations.
SN Partial Differential Equations and Applications, 1(2):1–34.

Kingma, D. P. and Ba, J. (2014).
Adam: A method for stochastic optimization.
arXiv preprint arXiv:1412.6980.

Rannacher, R. (1984).
Finite element solution of diffusion problems with irregular data.
Numerische Mathematik, 43(2):309–327.

References III

Sirignano, J. and Spiliopoulos, K. (2018).
DGM: A deep learning algorithm for solving partial differential equations.
Journal of computational physics, 375:1339–1364.