The Seven-League Scheme: Deep learning for large time step Monte Carlo simulation of SDEs

Shuaiqiang Liu[3], Lech A. Grzelak[2], **Kees Oosterlee**[1]

[1]Utrecht University, the Netherlands
[2] Rabobank, Utrecht, the Netherlands
[3]Centrum Wiskunde & Informatica, the Netherlands

March, 2021

Outline

Introduction

- Stochastic Differential Equations (SDEs) are widely used to model processes, in finance and other fields.
- Develop an accurate numerical scheme for SDES (in strong convergence sense) to carry out large time step simulation.
- Employ a neural network to obtain this numerical scheme and solve the SDEs, efficiently.

Stochastic differential equations

- Random variable $Y(t)$, on probability space $(\Omega, \Sigma, \mathbb{P})$ with filtration $\mathcal{F}(t)_{t\in[0,T]}$, sample space Ω, σ-algebra Σ and prob. measure $\mathbb{P}$.

- Consider the generic scalar Itô SDE,

$$\mathrm{d}Y(t) = a(t, Y(t), \boldsymbol{\theta})\mathrm{d}t + b(t, Y(t), \boldsymbol{\theta})\mathrm{d}W(t), \qquad 0 \leq t \leq T,$$

with drift $a(t, Y(t), \boldsymbol{\theta})$, diffusion $b(t, Y(t), \boldsymbol{\theta})$, parameters $\boldsymbol{\theta}$, Wiener process $W(t)$, and initial value $Y_0 := Y(t = 0)$.

The strong convergence

Solution in integral form

$$Y(t+\Delta t) = Y(t) + \int_t^{t+\Delta t} a(s, Y(s), \theta) ds + \int_t^{t+\Delta t} b(s, Y(s), \theta) dW(s)$$

Definition

Let the exact solution of an SDE at time t_i be given by $Y(t_i)$, its discrete approximation $\tilde{Y}(t_i)$ with time step Δt converges in the strong sense, with order $\beta_s \in \mathbb{R}^+$, if

$$\mathbb{E}|Y(t_i) - \tilde{Y}(t_i)| \leq K(\Delta t)^{\beta_s}.$$

Numerical discretization

Classical numerical schemes

Euler-Maruyama (strong order $\beta_s = 0.5$):

$$\hat{Y}_{i+1} = \hat{Y}_i + a(t_i, \hat{Y}_i, \theta)\Delta t + b(t_i, \hat{Y}_i, \theta)\sqrt{\Delta t}\hat{X}_{i+1}$$

Milstein (strong order $\beta_s = 1.0$):

$$\begin{aligned}\hat{Y}_{i+1} = \hat{Y}_i \quad &+ \quad a(t_i, \hat{Y}_i, \theta)\Delta t + b(t_i, \hat{Y}_i, \theta)\sqrt{\Delta t}\hat{X}_{i+1} \\ &+ \quad \frac{1}{2}b'(t_i, \hat{Y}_i, \theta)b(t_i, \hat{Y}_i, \theta)\Delta t(\hat{X}_{i+1}^2 - 1).\end{aligned}$$

where $\hat{Y}_{i+1} := \hat{Y}(t_{i+1})$ is a realization from $\tilde{Y}(t_{i+1})$, and $\hat{X}_{i+1}$ is drawn from $X \sim N(0,1)$, $b'(\cdot)$ the first derivative of $b(\cdot)$.

Higher-order numerical approximation

Common ways to improve numerical accuracy.

- Include higher-order terms[1]: ODE Runge-Kutta schemes plus high-order random terms of Itô calculus. For example, there are 8 terms for $\beta_s = 1.5$, and 12 terms for $\beta_s = 2.0$.

$$Y(t+\Delta t) \overset{d}{=} \underbrace{Y(t) + a\Delta t + b\sqrt{\Delta t}X + \frac{1}{2}b'b\Delta t(X^2-1) + \ldots}_{\text{12 terms}}$$

- Reduce time step Δt, e.g., use a finer time grid.

[1] Eckhard Platen (1999). An introduction to numerical methods for stochastic differential equations. Acta Numerica.

Data-driven numerical schemes

- The Euler-Maruyama scheme,

$$\hat{Y}_{i+1} = \underbrace{\hat{Y}_i + a(t_i, \hat{Y}_i, \theta)\Delta t}_{\alpha_0} + \underbrace{b(t_i, \hat{Y}_i, \theta)\sqrt{\Delta t}}_{\alpha_1} \hat{X}_{i+1},$$

$$\begin{cases} \alpha_0 = \hat{Y}_i + a(t_i, \hat{Y}_i, \theta)\Delta t, \\ \alpha_1 = b(t_i, \hat{Y}_i, \theta)\sqrt{\Delta t}, \end{cases}$$

- The Milstein scheme,

$$\begin{cases} \alpha_0 = \hat{Y}_i + a(t_i, \hat{Y}_i, \theta)\Delta t + \frac{1}{2} b'(t_i, \hat{Y}_i, \theta) b(t_i, \hat{Y}_i, \theta), \\ \alpha_1 = b(t_i, \hat{Y}_i, \theta)\sqrt{\Delta t}, \\ \alpha_2 = \frac{1}{2} b'(t_i, \hat{Y}_i, \theta) b(t_i, \hat{Y}_i, \theta). \end{cases}$$

A numerical scheme is a mapping function.

$$\tilde{Y}(t_{i+1}) | \tilde{Y}(t_i) \overset{d}{=} \sum_{j=0}^{m-1} \alpha_j X^j, \text{with } \alpha_j = H_j \left(t_i, \tilde{Y}(t_i), \Delta t, \theta \right).$$

The Seven-League Scheme

- "We march through time with the Seven-League scheme (*the 7L scheme*)."

- The 7L scheme consists of two key components,
  - Stochastic Collocation Monte Carlo sampler (SCMC)[1], efficient sampling from an 'expensive' distribution;
  - Neural network, as a powerful function approximator.

[1] L. A. Grzelak et al. (2019). The stochastic collocation Monte Carlo sampler. Quant. Finance.

Stochastic Collocation Monte Carlo sampler

- Two scalar random variables, Y and X, are connected by,

$$F_Y(Y) \stackrel{d}{=} U \stackrel{d}{=} F_X(X),$$

where $U \sim \mathcal{U}([0,1])$, $F_Y(\bar{y}) := P(Y \leq \bar{y})$, $F_X(\bar{x}) := P(X \leq \bar{x})$.

- When $F_Y(\bar{y})$ and $F_X(\bar{x})$ are strictly monotonic, we have

$$\bar{y} = F_Y^{-1}(F_X(\bar{x})) := g(\bar{x}),$$

both distributional and also element-wise.

- Choosing optimal collocation points, $(\hat{x}_j, \hat{y}_j)$, to approximate,

$$\bar{y} = g(\bar{x}) \approx \hat{g}_m(\bar{x}) = \sum_{j=1}^{m} \hat{y}_j \ell_j(\bar{x}),$$

where $\hat{y}_j = F_Y^{-1}(F_X(\hat{x}_j))$ (e.g. $\hat{x}_j$ Gauss-Hermite quadrature points), $\ell_j(\cdot)$ interpolation basis functions.

SCMC algorithm

- Exponential convergence to the target distribution.

SCMC algorithm

1. Calculate CDF $F_X(x_j)$ on $(x_1, \ldots, x_m)$, e.g., from Gauss-Hermite quadrature, giving m pairs $(x_j, F_X(x_j))$;
2. Invert the target CDF $y_j = F_Y^{-1}(F_X(x_j))$, $j = 1, \ldots, m$, and form m pairs of collocation points (x_j, y_j);
3. Define the interpolation function, $y = g_m(x)$, based on these m pairs (x_j, y_j);
4. Obtain sample $\hat{Y}$ by applying the mapping $\hat{Y} = g_m(\hat{X})$, where sample $\hat{X}$ is drawn from X.

Conditional distribution by SCMC

Given the current state $Y(t)$, the conditional variable $Y(t+\Delta t)$ can be written as,

$$Y(t+\Delta t)|Y(t) \stackrel{d}{=} g(X) \approx g_m(X).$$

Generating a sample path is drawing samples from the probability distribution conditional on a previous realization,

$$\hat{Y}_{i+1}|\hat{Y}_i = g_m(\hat{X}^j_{i+1}).$$

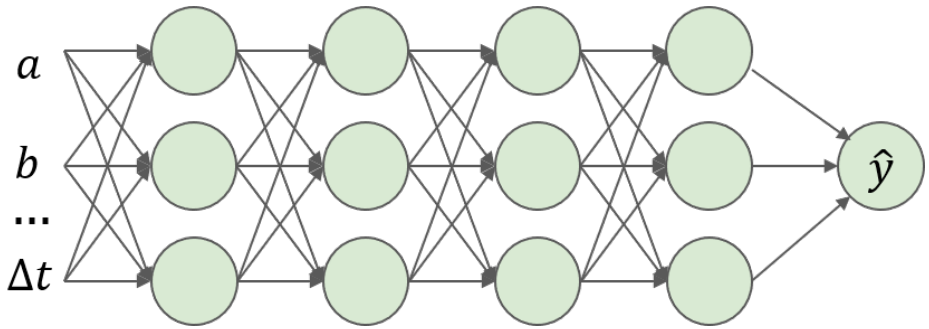
In the case of Lagrange interpolation, we arrive at the coefficient form,

$$\hat{Y}_{i+1}|\hat{Y}_i = g_m(\hat{X}^j_{i+1}) = \sum_{j=0}^{m-1} \hat{\alpha}_{i+1,j}\hat{X}^j_{i+1},$$

Learn stochastic collocation points

- We use an artificial neural network (ANN) to determine the conditional collocation points, $y_j(t_{i+1})$, based on the previous realization,

$$y_j(t_{i+1})|\hat{Y}_i = H_j(\hat{Y}_i, t_i, \Delta t, \theta).$$



- When the conditional stochastic collocation points are known, the mapping $g_m(\cdot)$ is constructed via interpolation.

Artificial Neural Network (ANN)

Artificial neural network is used as a function approximator.

- Fully connected neural network is a composite function,

$$\hat{H}(\acute{x}|\tilde{\Theta}) = h^{(L_A)}(\ldots h^{(2)}(h^{(1)}(\acute{x};\tilde{\theta}_1);\tilde{\theta}_2);\ldots\tilde{\theta}_{L_A}),$$

  with $\acute{x}$ input variables, $\tilde{\Theta}$ weights and biases, L_A hidden layers.

- Minimizing the loss function to approximate the target function,

$$\underset{\tilde{\Theta}}{\text{argmin}}\, L(\tilde{\Theta}) := \underset{\tilde{\Theta}}{\text{argmin}} \sum D(\hat{H}(\acute{x}_i|\tilde{\Theta}), \acute{y}_i),$$

  where $D(\cdot,\cdot)$ measures the distance between predicted $\hat{H}(\acute{x}_i|\theta)$ and true $\acute{y}_i$ values.

The offline stage

The ANN learns conditional stochastic collocation points,

$$y_j(\tau + d\tau)|\hat{Y}(\tau) = H_j\left(\hat{Y}(\tau), \tau, d\tau, \boldsymbol{\theta}\right), \quad j = 1, 2, \ldots, m,$$

1. Generate many data samples for different time steps $d\tau$ and model parameters $\boldsymbol{\theta}$.
2. Use a numerical scheme (e.g. Euler) with tiny $\Delta\tau$ to find the conditional distribution $Y(\tau + d\tau)|Y(\tau)$.
3. Calculate stochastic collocation points $y_j(\tau + d\tau)|\hat{Y}(\tau)$ with SCMC.
4. Train ANN (supervised learning) to obtain $\hat{H}_j \approx H_j$.

The online stage

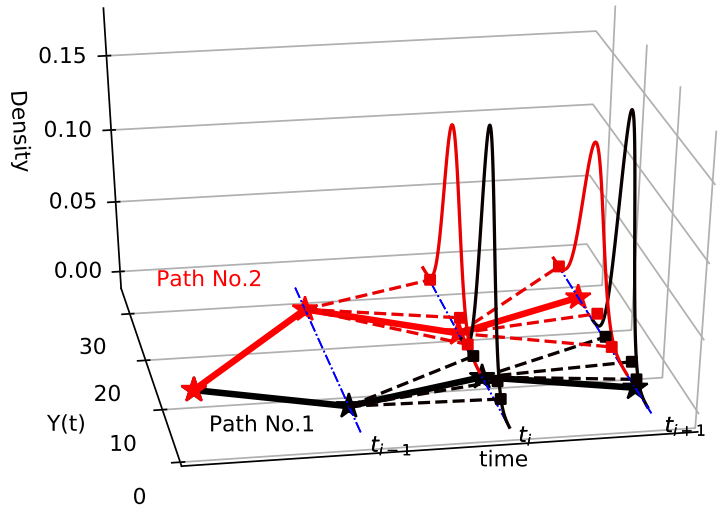
Use the trained ANNs to generate the sample paths.

1. Partition $[0, T]$, $0 < t_1 < t_2 < \ldots < t_N \leq T$, with $\Delta t = t_{i+1} - t_i$.
2. Given a sample $\hat{Y}_i$ at time t_i, compute m collocation points at time t_{i+1} using the trained ANN,

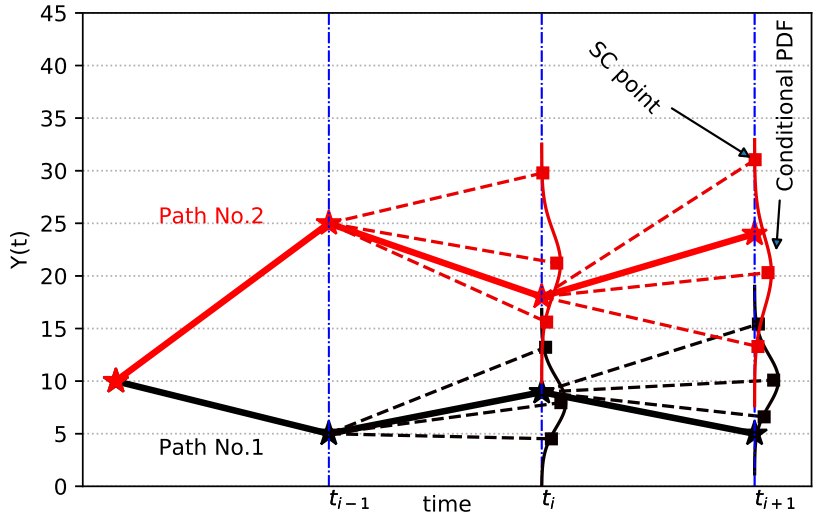
$$\hat{y}_j(t_{i+1})|\hat{Y}_i = \hat{H}_j(\hat{Y}_i, t_i, t_{i+1} - t_i, \theta), j = 1, 2, \ldots, m.$$

3. Compute the mapping function $g_m(\cdot)$ using SCMC.
4. Sample from X to obtain a sample $\hat{Y}_{i+1}|\hat{Y}_i = g_m(\hat{X}_{i+1})$.
5. Return to Step 2 by $t_{i+1} \to t_i$, iterate until terminal time T.

Generating sample paths with 7L



(a) Sample paths by 7L



(b) The 2D projection

Figure: Here conditional SC points, represented by ■, are conditional on a previous realization, denoted by ★.

Error analysis

The error from SCMC:

$$\mathbb{E}[|g(x) - g_m(x)|] \leq \sqrt{|\epsilon_m|} = \sqrt{\left| \frac{m!\sqrt{\pi}}{2^m} \frac{\Psi^{(2m)}(\hat{\xi}_1)}{(2m)!} \right|}.$$

The error from ANNs:

$$\begin{aligned}
\mathbb{E}\left[|g_m(x) - \tilde{g}_m(x)|\right] &= \int_{\mathbb{R}} \left| \sum_{j=1}^{m} \epsilon_j^A \ell_j(x) \right| f_X(x) \mathrm{d}x \\
&\leq \int_{\mathbb{R}} \max\{|\epsilon_1^A|, \ldots, |\epsilon_m^A|\} f_X(x) \mathrm{d}x \\
&= \max\{|\epsilon_1^A|, \ldots, |\epsilon_m^A|\}.
\end{aligned}$$

$$\begin{aligned}
\mathbb{E}\left[|g(x) - \tilde{g}_m(x)|\right] &\leq \mathbb{E}\left[|g(x) - g_m(x)|\right] + \mathbb{E}\left[|g_m(x) - \tilde{g}_m(x)|\right] \\
&\leq \sqrt{|\epsilon_m|} + \max\{|\epsilon_1^A|, \ldots, |\epsilon_m^A|\}.
\end{aligned}$$

Strong convergence error of 7L

7L strong convergence

Assuming the approximation errors from ANN and SCMC are negligible, the strong convergence of the 7L scheme is defined, as

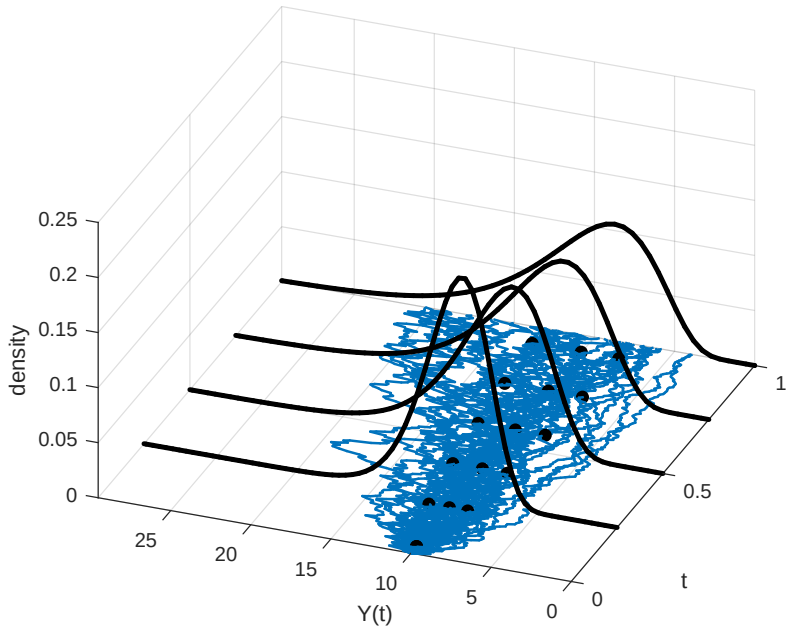
$$\mathbb{E}|\tilde{Y}(t_i) - Y(t_i)| \leq \epsilon(\Delta\tau) \ll K(\Delta t)^{\beta_S},$$

where $\Delta\tau$ is used for offline ANN training, and time step Δt for online ANN prediction, with $\Delta\tau \ll \Delta t$.

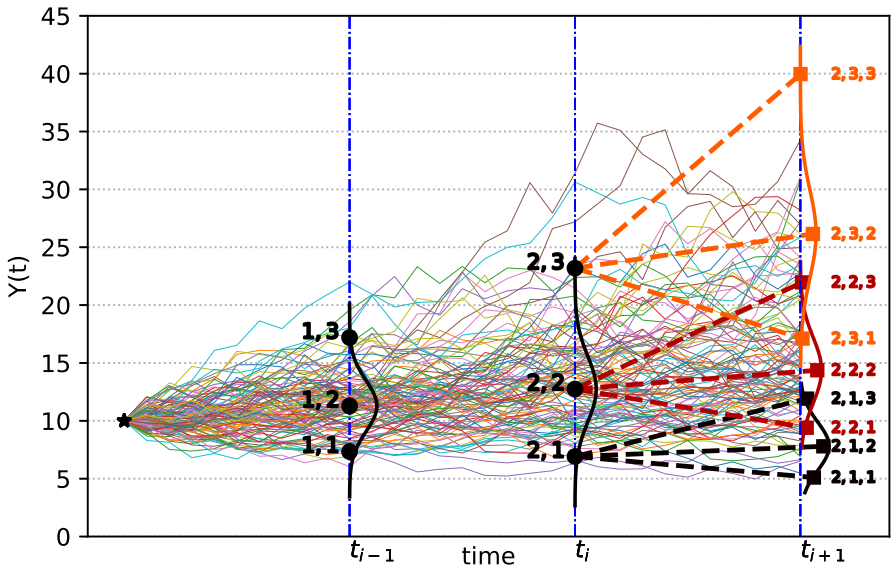
For example, the Euler scheme would need a much finer time step, by a factor $\kappa = \Delta t/\Delta\tau$ to achieve a similar accuracy in the strong sense.

7L-CDC

The 7L Compression-Decompression (CDC) variant can further reduce the computational cost.



(a) Paths for marginal SC points



(b) Sample paths by 7L-CDC

Figure: Schematic diagram of the 7L-CDC scheme at time t_i. Left: Marginal SC points. Right: Sample paths generated by 7L-CDC.

7L-CDC computation time

The time ratio between the 7L-CDC and 7L schemes is

$$\gamma = \frac{t_I M + t_A M_s}{t_A M} = \frac{t_I}{t_A} + \frac{M_s}{M},$$

with t_A the computational time of the ANN, t_I for the interpolation. Given the fact that the number of sample paths is typically much larger than the number of SC points $M \gg M_s$,

$$\gamma \approx \frac{t_I}{t_A}.$$

When the employed interpolation is computationally cheaper than the ANN, $\gamma < 1$, so that the 7L-CDC scheme needs fewer computations than the 7L scheme.

4. Numerical experiments

With $a(t, Y(t)) = \mu Y(t)$ and $b(t, Y(t)) = \sigma Y(t)$, we have Geometric Brownian Motion (GBM),

$$dY(t) = \mu Y(t)\mathrm{d}t + \sigma Y(t)\mathrm{d}W(t), \qquad 0 \leq t \leq T,$$

where drift μ and volatility σ are constant, with initial value Y_0. The model parameters are $\boldsymbol{\theta} := \{\mu, \sigma\}$, and the analytic solution is,

$$Y(t) \overset{d}{=} Y_0 e^{(\mu - \frac{1}{2}\sigma^2)(t-t_0) + \sigma\sqrt{t-t_0}X}. \tag{1}$$

ANN settings and off-line training

We created the data set, using Euler, $\Delta\tau = 0.01$. There are around 80,000 data samples.

ANN	Parameters	Value range	Method
input	drift, μ	(0.0, 0.10]	LHS
	volatility, σ	[0.05, 0.60]	LHS
	value, Y_0	[0.10, 15.0]	LHS
	time, τ_{max}	(0.0, 1.60]	Equidistant
$\hat{H}_1(\cdot)$ output	point, $\hat{y}_1$	$(0.0, 25.65)$	SCMC
$\hat{H}_2(\cdot)$ output	point, $\hat{y}_2$	$(0.0, 25.98)$	SCMC
$\hat{H}_3(\cdot)$ output	point, $\hat{y}_3$	$(0.0, 27.84)$	SCMC
$\hat{H}_4(\cdot)$ output	point, $\hat{y}_4$	$(0.0, 54.67)$	SCMC
$\hat{H}_5(\cdot)$ output	point, $\hat{y}_5$	$(0.0, 154.35)$	SCMC

ANN hyper-parameters	value
Hidden layers	4
Neurons (per layer)	50
Activation	Softplus
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	1024
Initial Learning rate	1e-3

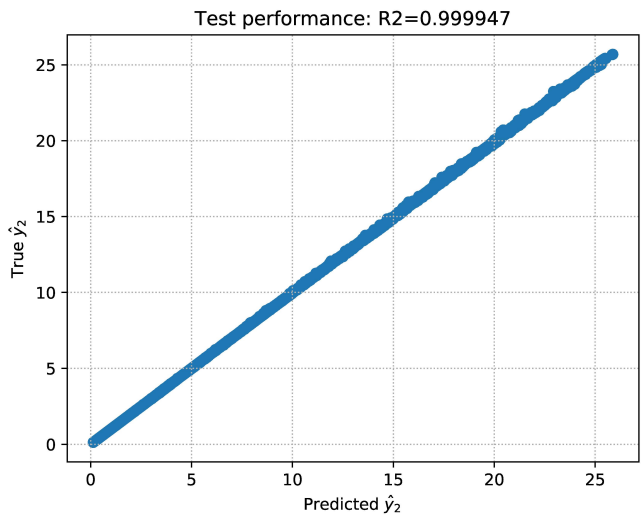
Performance of the ANNs

The data set is divided into three parts, 80% for training, 10% for validation, 10% for testing. After 1500 training epochs, the ANN converged.

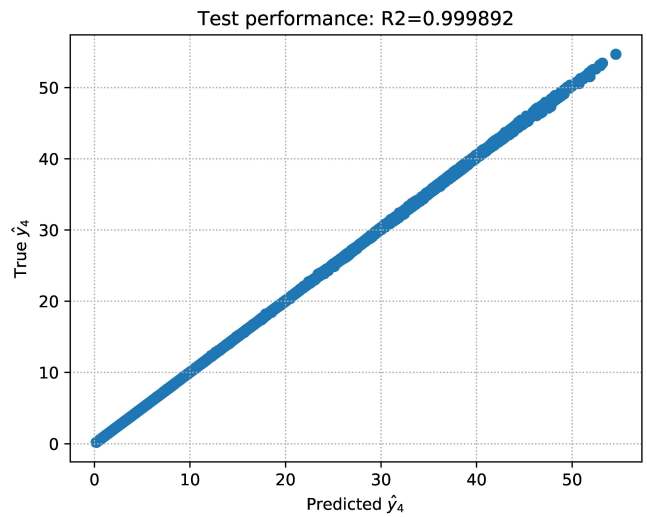
- Performance on the test dataset. $\text{MAE} = \frac{1}{M}\sum_j |y_j - \hat{y}_j|$.

SC points	$\hat{y}_1$	$\hat{y}_2$	$\hat{y}_3$	$\hat{y}_4$	$\hat{y}_5$
R^2	0.999891	0.999947	0.999980	0.999892	0.999963
MAE	0.026	0.027	0.021	0.071	0.066

- Predicted vs true collocation points, for example,



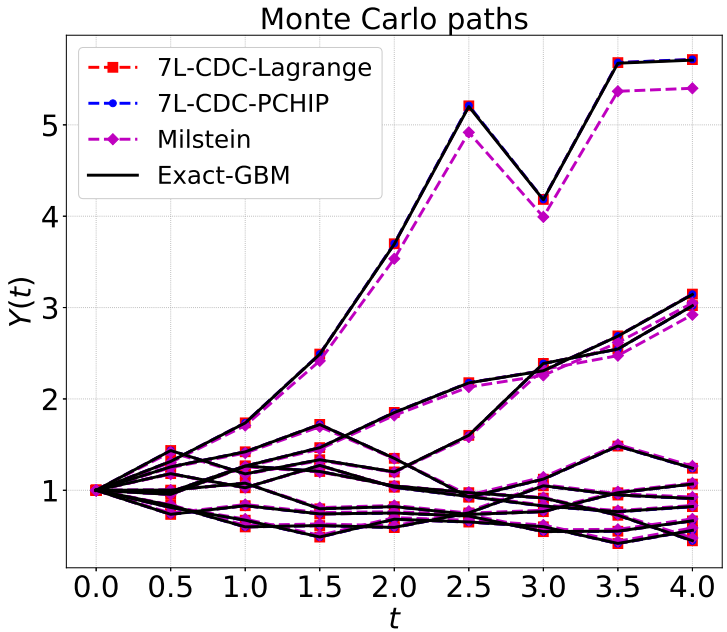
(a) $\hat{y}_2$



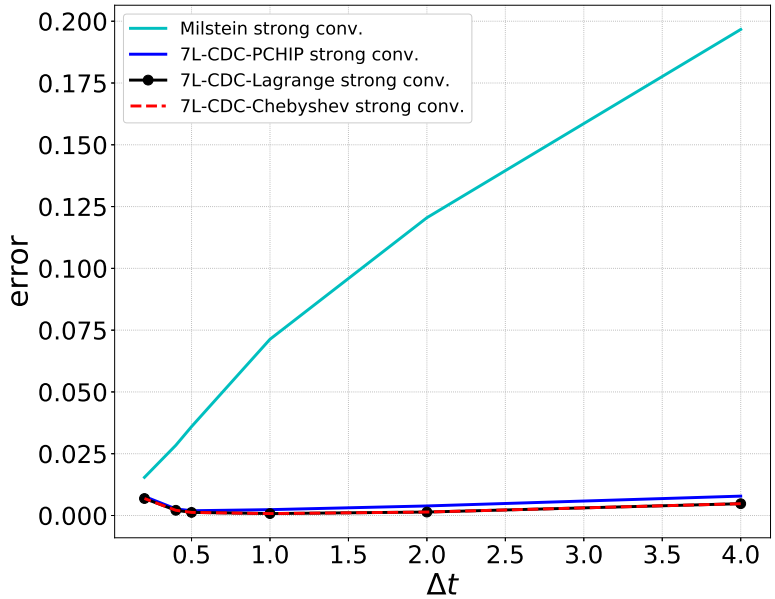
(b) $\hat{y}_4$

Comparing path-wise error

- Monte Carlo path-wise error:



(c) Sample paths, $\Delta t = 0.5$



(d) Strong convergence

Here $\sigma = 0.3$, $r = 0.1$, $S_0 = 1.0$, $T = 4.0$. We use the barycentric version of Lagrange interpolation.

CPU time

The CPU time (seconds) to reach the same accuracy (CPU: E3-1240, 3.40GHz): simulating 10,000 sample paths until terminal time $T = 4.0$, based on 5×5 SC points.

Method /Time (Sec.)	$\Delta t = 1.0$			$\Delta t = 2.0$		
	Create C	Decom. C	Total	Create C	Decom. C	Total
7L-CDC Barycentric	0.054	4.93	4.98	0.027	2.48	2.51
7L-CDC Chebyshev	0.054	9.78	9.83	0.027	4.93	4.96
7L-CDC PCHIP	0.054	11.39	11.44	0.027	5.73	5.76
7L scheme	-	-	12.80	-	-	6.39
Milstein	-	-	27.01	-	-	27.70

$\Rightarrow$ This is much faster on GPUs (recent results)!

We have essentially the same accuracy results with the OU process.

Applications in finance

We can carry out large time step Monte Carlo simulations for

- Pricing path-dependent options
- Computing the sensitivities of path-dependent options.

Bermudan option, big time steps

- The option holder has the right to exercise the contract at pre-specified monitoring dates up to maturity time T.
- Relative error: $\epsilon_{rel} = \left| \frac{V_A^{ref}(t_0) - V_A(t_0)}{V_A^{ref}(t_0)} \right|$;
- Longstaff-Schwartz method; $Y_0 = 1.0$, $r = 0.1$, strike price $\tilde{K} = 1.1$, $T = 4.0$, number of monitoring dates N_b, $M = 100,000$.

	Method \ Value	$\Delta t = 1.0$, N_b=4	$\Delta t = 0.5$, N_b=8
σ=0.30	Analytic MC	0.15213858 (0.00%)	0.16161876 (0.00%)
	Milstein MC	0.13872771 (8.81%)	0.15429369 (4.53%)
	7L-CDC	0.15234901 (0.14%)	0.16196264 (0.21%)

Pricing Asian options

- Pay-off of a fixed strike Asian option:

$$V_A(T) = \max\left(A(T) - \tilde{K}, 0\right),$$

with $A(T) = \frac{1}{N_b}\sum_{k=1}^{N_b} \hat{Y}(t_k)$ discrete average over N_b dates $\in [0, T]$.

- The option value $V_A(t) = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[V(A(T))\middle|\mathcal{F}(0)\right]$.
- $Y_0 = 1.0$, $\tilde{K} = Y_0$, $r = 0.1$, $T = \Delta t \times N_b$, $M = 100,000$.

	Method \ Value	$\Delta t = 1.0$, N_b=4	$\Delta t = 0.5$, N_b=8
σ=0.40	Analytic MC	0.28515109 (0.00%)	0.25723594 (0.00%)
	Milstein MC	0.26394277 (7.44%)	0.24717425 (3.91%)
	7L-CDC	0.28482371 (0.11%)	0.25647592 (0.30%)

Computing Greeks

The Asian option's vega is defined as

$$\frac{\partial V}{\partial \sigma} = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[\sum_{i=1}^{N_b}\frac{\partial V(T, Y(t_i);\sigma)}{\partial Y(t_i)}\frac{\partial Y(t_i)}{\partial \sigma}\Big|\mathcal{F}(0)\right].$$

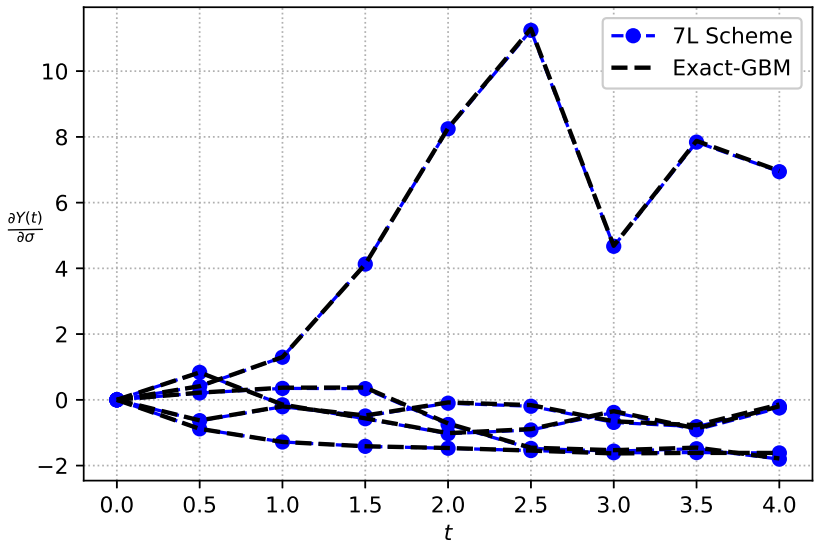
For M sample paths and N time points,

$$\frac{\partial V}{\partial \sigma} \approx e^{-rT}\frac{1}{M}\frac{1}{N}\left[\sum_{k=1}^{M}\sum_{i=1}^{N_b}\left(1_{A(T)>\tilde{K}}\sum_{j=0}^{m-1}\frac{\partial \hat{H}_j}{\partial \sigma}p_j(\hat{X}_{k,i+1})\right)\Big|\mathcal{F}(0)\right].$$

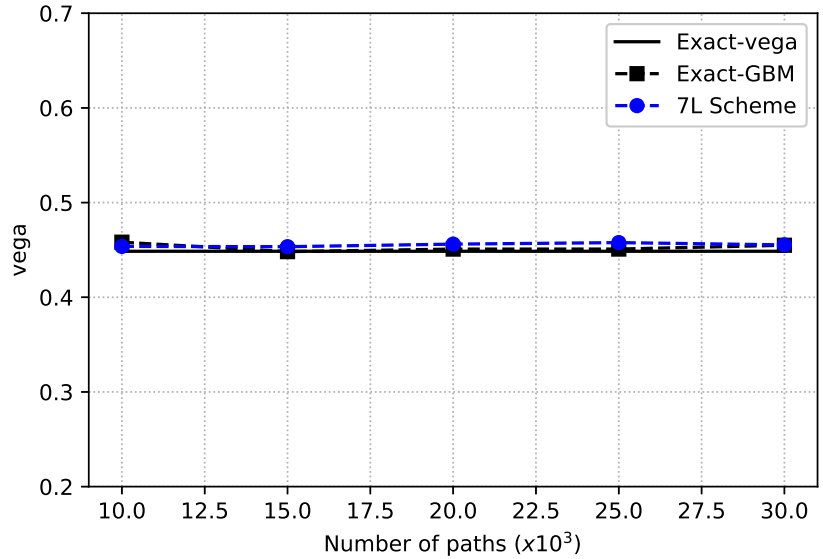
The derivative of the ANNs $\hat{H}_j(\cdot)$ w.r.t σ can be computed,

$$\frac{\partial \hat{H}_j(\hat{Y}_i,\sigma)}{\partial \sigma} = \frac{\partial \hat{H}_j(\sigma;\hat{Y}_i)}{\partial \sigma} + \frac{\partial \hat{H}_j(\hat{Y}_i;\sigma)}{\partial \hat{Y}_i}\frac{\partial \hat{Y}_i}{\partial \sigma}.$$

Asian option's vega



(e) Path-wise sensitivity of $Y(t)$



(f) Asian option's vega, $\Delta t = 1.0$

The parameters are $Y_0 = 1.0$, $r = 0.05$, $\tilde{K} = Y_0$, $\sigma = 0.3$, $T = 4.0$.

5. Summary and Outlook

Conclusion:

- The 7L-scheme provides a data-driven numerical solver for SDEs.
- Numerical error, in the sense of strong convergence, does not grow significantly when the step increases.
- Various applications in computational finance.

Outlook:

- Parallel computing on GPUs (is already almost finalized!)
- Solving multi-dimensional SDEs.

References

1. S. Liu, et al (2020). The Seven-League Scheme: Deep learning for large time step Monte Carlo simulations of stochastic differential equations, arXiv:2009.03202.
2. G. Cybenko, et al(1989). Approximations by superpositions of sigmoidal functions, Mathematics of Control, Signals and Systems.
3. R. Cameron, et al (1947). The Orthogonal Development of Non-Linear Functionals in Series of Fourier-Hermite Functionals. Annals of Mathematics.
4. Eckhard Platen (1999). An introduction to numerical methods for stochastic differential equations. Acta Numerica.
5. L. A. Grzelak, et al (2019). The stochastic collocation Monte Carlo sampler. Quantitative Finance.
6. Y. Bar-Sinai, et al (2019). Learning data-driven discretizations for partial differential equations. PNAS.
7. Dmitry Yarotsky (2017). Error bounds for approximations with deep ReLU networks. Neural Networks.

Training ANNs

The layer function:

$$z_j^{(\ell)} = \varphi^{(\ell)} \left(\sum_i w_{ij}^{(\ell)} z_i^{(\ell-1)} + b_j^{(\ell)} \right),$$

Stochastic Gradient Descent (SGD) optimizes the parameters,

$$\begin{cases} w \leftarrow w - \eta(i) \dfrac{\partial L}{\partial w}, \\ b \leftarrow b - \eta(i) \dfrac{\partial L}{\partial b}, \quad i = 0, 1, 2, ..., \end{cases}$$

where η is a learning rate.