Deep learning for large time-step simulations of stochastic differential equations

Shuaiqiang Liu[*], Lech A. Grzelak[*], Cornelis W. Oosterlee[*&]

[*]Delft University of Technology, the Netherlands
[&]Centrum Wiskunde & Informatica, the Netherlands

July 2, 2020

Outline

1. Introduction

- Stochastic differential equations (SDEs) describe uncertainty in finance, physics, epidemics, etc.
- Accurate numerical schemes are needed to carry out large time step Monte Carlo simulations, but difficult to develop.
- Deep learning techniques have been used to solve PDEs, especially with a physics-informed neural network(Raissi 2017, Yohai 2019, etc) .
- Can deep neural networks learn high-order numerical schemes to solve SDEs?

1.1 Stochastic differential equations

- Considering a generic scalar Itô SDE,

$$dY_t = a(Y_t,t)dt + b(Y_t,t)dW_t, 0 \leq t \leq T, \tag{1}$$

  where Y_t is the random variable with drift $a(Y_t,t)$, variance $b(Y_t,t)$, Wiener process W_t, given initial value $Y_0 := Y_{t=0}$.

- The real-valued random variable Y is defined on the probability space $(\Omega, \Sigma, \mathbb{P})$, with sample space Ω, σ-algebra Σ and probability measure $\mathbb{P}$.

1.2 Strong convergence of numerical solution

Solution in the integral form

$$Y_{t+\Delta t} = Y_t + \int_t^{t+\Delta t} a(Y_s, s)ds + \int_t^{t+\Delta t} b(Y_s, s)dW_s$$

Definition

Let the exact solution of an SDE at time t_i be given by Y_{t_i}, its discrete approximation $\hat{Y}_i$ with time step Δt converges in the strong sense, with order $\beta_s \in \mathbb{R}^+$, if there exists a constant C such that

$$E|Y_{t_i} - \hat{Y}_i| \le C(\Delta t)^{\beta_s}. \tag{2}$$

1.3 Numerical discretization

Classical Numerical schemes

Euler-Maruyama (strong order β_s = 0.5):

$$Y_{t+\Delta t} = Y_t + a(Y_t)\Delta t + b(Y_t)\sqrt{\Delta t}Z$$

Milstein (strong order β_s = 1.0):

$$Y_{t+\Delta t} = Y_t + a(Y_t)\Delta t + b(Y_t)\sqrt{\Delta t}Z + \frac{1}{2}b'(Y_t)b(Y_t)\Delta t(Z^2 - 1),$$

where $b'(Y)$ is the first derivative of $b(Y)$, $Z \sim N(0, 1)$.

1.4 Higher-order numerical approximation

How to improve numerical accuracy?

- Include higher-order terms[1]: ODE Runge-Kutta schemes plus high-order random terms of Itô calculus. There are 8 terms when β_s = 1.5, and 12 terms when β_s = 2.0.

$$Y_{t+\Delta t} = \underbrace{Y_t + a(Y_t)\Delta t + b(Y_t)\sqrt{\Delta t} Z + \frac{1}{2} b'(Y_t) b(Y_t) \Delta t (Z^2 - 1) + \ldots}_{\text{12 terms}}$$

High-order schemes are expensive to develop and implement.

- Reduce time step Δt, e.g., finer time grid.
The computational cost grows rapidly with more time points.

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

1.5 Coefficients of numerical schemes

- Euler-Maruyama, $Y_{t+\Delta t} = \underbrace{Y_t + a(Y_t)\Delta t}_{\alpha_0} + \underbrace{b(Y_t)\sqrt{\Delta t}}_{\alpha_1} Z$, where,

$$\begin{cases} \alpha_0 := Y_t + a(Y_t, t)\Delta t, \\ \alpha_1 := b(Y_t, t)\sqrt{\Delta t}, \end{cases} \tag{3}$$

The coefficients are a function of model parameters and time.

$$\alpha_j = \bar{H}_j \left(Y_t, a(Y_t, t), b(Y_t, t), t, \Delta t\right). \tag{4}$$

$\Rightarrow$Is it possible to learn these functions using machine learning?

2. Stochastic collocation method

Stochastic collocation Monte Carlo sampler (SCMC)[2]:

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

$$F_Y(Y) \overset{d}{=} U \overset{d}{=} F_X(X), \tag{5}$$

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

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

$$\bar{y} = F_Y^{-1}(F_X(\bar{x})) := g(\bar{x}). \tag{6}$$

where the function $g(\cdot)$ maps sample $\bar{x}$ from X to sample $\bar{y}$ from Y, in the sense of both distribution and element-wise.

[2] L. A. Grzelak, etc (2019). The stochastic collocation Monte Carlo sampler. Quantitative Finance.

2.1 Stochastic collocation method

- 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 $\hat{y}_j = F_Y^{-1}(F_X(\hat{x}_j))$, $\ell_j(\bar{x})$ interpolation basis functions.
- The polynomial chaos expansion reads

$$Y = \sum_{j=1}^{m} \alpha_{j-1} X^{j-1} = \alpha_0 + \alpha_1 X + ... + \alpha_{m-1} X^{m-1}, \qquad (7)$$

  where the coefficients are computed by

$$\mathbf{A}\boldsymbol{\alpha} = \hat{\mathbf{y}}$$

$$\begin{pmatrix} 1 & \hat{x}_1^1 & \hat{x}_1^2 & \dots & \hat{x}_1^{m-1} \\ 1 & \hat{x}_2^1 & \hat{x}_2^2 & \dots & \hat{x}_2^{m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \hat{x}_m^1 & \hat{x}_m^2 & \dots & \hat{x}_m^{m-1} \end{pmatrix} \begin{pmatrix} \hat{\alpha}_0 \\ \hat{\alpha}_1 \\ \vdots \\ \hat{\alpha}_{m-1} \end{pmatrix} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{pmatrix}$$

2.2 SCMC algorithm

- The Cameron-Martin Theorem (1947) states that polynomial chaos approximations based on the normal distribution converge to any distribution.
- An increasing polynomial order in SCMC leads to exponential convergence.

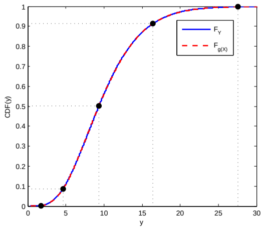
SCMC algorithm

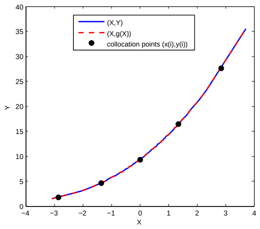
1. Calculate the CDF $F_X(\hat{x}_j)$ on m collocation points $(\hat{x}_1, \hat{x}_2, ..., \hat{x}_m)$ to obtain m pairs $(\hat{x}_j, F_X(\hat{x}_j))$;
2. Invert the CDF by $\hat{y}_j = F_Y^{-1}(F_X(\hat{x}_j))$ to form m triples $(\hat{x}_j, F_X(\hat{x}_j), \hat{y}_j)$;
3. Compute the interpolation function $\bar{y} = \hat{g}(\bar{x})$ on m pairs $(\hat{x}_j, \hat{y}_j)$.

2.3 An example of SCMC

Cheap distribution $X \sim \mathcal{N}(0,1)$, Gauss quadrature points $\hat{x}_i$; expensive distribution $Y \sim \Gamma(5,2)$, optimal collocation points $\hat{y}_i = F_Y^{-1}(F_X(\hat{x}_i))$, m=5 collocation points.

	$\hat{x}_1$	$\hat{x}_2$	$\hat{x}_3$	$\hat{x}_4$	$\hat{x}_5$
$\hat{x}_i$	-2.8570	-1.3556	0.0	1.3556	2.8570
$F_X(\hat{x}_i)$	0.0021	0.0876	0.50	0.9124	0.9979
$F_Y^{-1}(F_X(\hat{x}_i))$	1.7543	4.6651	9.3418	16.4443	27.5888





Stochastic collocation points parameterize the CDFs.

2.4 Connection between classical MC and SCMC

We aim to draw a conditional sample from the distribution,

$$F_{Y_{t+\Delta t}}(\bar{y}|Y_t = Y_i) \sim P(Y_{t+\Delta t} < \bar{y}|Y_t = Y_i).$$

For example, when there are three collocation points $m = 3$,

- SCMC reads

$$Y_{t+\Delta t} = \sum_{j=1}^{3} \alpha_{j-1} X^{j-1} = \alpha_0 + \alpha_1 X + \alpha_2 X^2,$$

- The Milstein scheme reads,

$$\begin{cases} X := Z \\ \alpha_0 := Y_i + a(Y_i, t_i)\Delta t + \frac{1}{2} b'(Y_i, t_i) b(Y_i, t_i), \\ \alpha_1 := b(Y_i, t_i)\sqrt{\Delta t}, \\ \alpha_2 := \frac{1}{2} b'(Y_i, t_i) b(Y_i, t_i). \end{cases}$$

2.5 Connection between classical MC and SCMC

$\Rightarrow$ These coefficients vary at different time points,

$$\boldsymbol{\alpha}_{i+1} = \bar{\mathbf{H}}_j \left(Y_i, a(Y_i, t_i), b(Y_i, t_i), t_i, \Delta t\right). \tag{8}$$

$$\mathbf{A}\boldsymbol{\alpha}_{\mathbf{i+1}} = \hat{\mathbf{y}}_{\mathbf{i+1}}$$

$$\begin{pmatrix} 1 & \hat{x}_1^1 & \hat{x}_1^2 & \dots & \hat{x}_1^{m-1} \\ 1 & \hat{x}_2^1 & \hat{x}_2^2 & \dots & \hat{x}_2^{m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \hat{x}_m^1 & \hat{x}_m^2 & \dots & \hat{x}_m^{m-1} \end{pmatrix} \begin{pmatrix} \hat{\alpha}_{i+1,0} \\ \hat{\alpha}_{i+1,1} \\ \vdots \\ \hat{\alpha}_{i+1,m} \end{pmatrix} = \begin{pmatrix} \hat{y}_{i+1,1} \\ \hat{y}_{i+1,2} \\ \vdots \\ \hat{y}_{i+1,m} \end{pmatrix},$$

$$\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{H}} \left(Y_i, a(Y_i, t_i), b(Y_i, t_i), t_i, \Delta t\right). \tag{9}$$

$\Rightarrow$ Stochastic collocation points vary at different time points.

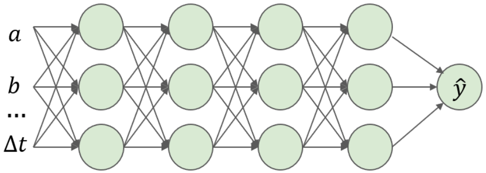
PS: for Markov processes, $\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{H}} \left(Y_i, a(Y_i, t_i), b(Y_i, t_i), \Delta t\right)$ does not depend time t_i.

2.6 Learn stochastic collocation points

There are two options to determine these coefficients at time t_i,

- Output the coefficients inferred from SC points;
- Output SC points, inferring coefficients $\boldsymbol{\alpha}_{i+1} = \mathbf{A}^{-1}\hat{\mathbf{y}}_{\mathbf{i+1}}$.

  $\Rightarrow$ SC points have physical meaning and are interpretable.



$$\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{H}}\left(Y_i, a(Y_i, t_i), b(Y_i, t_i), t_i, \Delta t\right).$$

3. Deep Neural Networks

Artificial neural networks are used as function approximators.

- Fully connected neural networks are a composite function,

$$H(x|\theta) = h^{(L)}(...h^{(2)}(h^{(1)}(x;\theta^{(1)});\theta^{(2)});...\theta^{(L)})$$

where $\theta = (\mathbf{W}_i, \mathbf{b}_i)$, $\mathbf{W}_i$ weights and $\mathbf{b}_i$ bias each hidden layer.

- Supervised learning to approximate the target function,

$$\operatorname*{argmin}_{\theta} \text{Loss}(\theta) := \operatorname*{argmin}_{\theta} \sum D(F(x_i|\theta), y_i),$$

where $D(\cdot,\cdot)$ measures the difference between $H(x_i|\theta)$ and y_i.

3.1 Approximation capacity

Why deep neural networks?

- Universal approximation theorem.
- Expressive power grows exponentially with the depth of ANN.

Deep neural networks[3]

Give any $\hat{\epsilon} \in (0,1)$, there exists a neural network which is capable of approximating any function from $F_{d,n}$ (d inputs, up to n-th derivatives) with error $\hat{\epsilon}$, using the following configurations:

- at least piece-wise activation functions,
- at least $c(\ln(1/\hat{\epsilon})+1)$ hidden layers and $c\hat{\epsilon}^{-d/n}(\ln(1/\hat{\epsilon})+1)$ weights and computation units, where $c := c(d,n)$ is constant, depending two parameters d and n.

$\Rightarrow$ An arbitrary approximation error can always be achieved with a sufficient bounded number of hidden layers and nodes.

[3] Dmitry Yarotsky (2017). Error bounds for approximations with deep ReLU networks.

3.2 Error from SCMC

When Gauss-Hermite quadrature is used, let $f_X(x)$ represent weight function, ω_i quadrature weights, $\Psi(x) = (g(x) - \hat{g}(x))^2$ the difference of distribution functions, M collocation points,

$$\int_R \Psi(x) f_X(x) = \sum_{i=1}^{M} \Psi(x_i)\omega_i + \epsilon_M,$$

where the approximation error is

$$\epsilon_M = \frac{M!\sqrt{\pi}}{2^M} \frac{\Psi^{(2M)}(\hat{\xi}_1)}{(2M)!}, \tag{10}$$

$\Rightarrow$ The error of SCMC exponentially converges to zero when the number of stochastic collocation points grows.

3.3 Estimating strong error of ANN-SCMC

ANN-SCMC strong convergence

When the approximation errors from ANN and SCMC are zero, ANN-SCMC has a strong convergence,

$$E|Y_{t_i} - \hat{Y}_i| \leq \epsilon(\Delta\tau) \leq C(\Delta t)^{\beta_s},$$

where time step $\Delta\tau$ is used to create (off-line) training data, and actual time step Δt is for prediction (on-line), $\Delta\tau << \Delta t$.

- During off-line training, small $\Delta\tau$ guarantees small error $\epsilon(\Delta\tau)$.
- When using the trained ANN-SCMC to solve SDEs, strong error $\epsilon(\Delta\tau)$ stays regardless of actual time step Δt.

3.4 ANN-SCMC numerical solver

1. Train ANN-SCMC off-line for $\hat{\mathbf{y}} = \hat{\mathbf{H}}\left(\cdot\right)$ based on generated data (e.g., using any classical MC with time step $\Delta\tau$).
2. Compute collocation points at time $t_{i+1} = t_i + \Delta t$, given $Y_i, a(Y_i, t_i), b(Y_i, t_i)$ at time t_i,

$$\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{H}}\left(Y_i, a(Y_i, t_i), b(Y_i, t_i), t_i, \Delta t\right).$$

3. Calculate the coefficients $\boldsymbol{\alpha}_{i+1}$ through $\boldsymbol{\alpha}_{i+1} = \mathbf{A}^{-1}\hat{\mathbf{y}}_{\mathbf{i+1}}$.
4. Implement the polynomial expansion for all sample paths,

$$Y_{i+1} = \sum_{j=1}^{m} \alpha_{i+1,j-1} Z^{j-1}.$$

5. Return to Step 2 by $t_{i+1} \to t_i$, iterating until terminal time T.

4. Numerical experiments

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

$$dY_t = \mu Y_t dt + \sigma Y_t dW_t, 0 \leq t \leq T,$$

where drift μ and volatility σ are constant, with initial value Y_0.

- The Milstein scheme,

$$Y_{i+1,j} = Y_{ij}\left(1 + \mu\Delta t + \frac{1}{2}\sigma^2\Delta t\right) + \left(\sigma Y_{ij}\sqrt{\Delta t}\right) Z + \left(\frac{1}{2}\sigma^2 Y_{ij}\Delta t\right) Z^2,$$
$$\begin{cases} \alpha_{i+1,0} := Y_{ij} + Y_{ij}\mu\Delta t + \frac{1}{2}\sigma^2 Y_{ij}\Delta t, \\ \alpha_{i+1,1} := \sigma Y_{ij}\sqrt{\Delta t}, \\ \alpha_{i+1,2} := \frac{1}{2}\sigma^2 Y_{ij}\Delta t. \end{cases}$$

- For GBM, the conditional collocation points of ANN-SCMC, $\hat{\mathbf{y}}_{i+1} = \hat{\mathbf{H}}(\mu, \sigma, Y_i, \Delta t)$, are independent of time t.

4.1 ANN settings and off-line training

Hyper-parameters	Options
Hidden layers	4
Neurons (each layer)	50
Activation	Softplus
Initialization	Glorot_uniform
Optimizer	Adam
Batch size	1024
Training epochs	1700
Initial Learning rate	1e-3

ANN	Parameters	Value range	Method
input	drift, μ	[0.05, 0.50]	LHS
	volatility, σ	(0.0, 0.60]	LHS
	initial value, Y_0	(0.0, 15.0]	LHS
	time horizon, τ_{max}	(0, 4.0]	Equally
$\hat{H}_0(\cdot)$ output	point, $\hat{y}_0$	$(0.0, 15.5)$	SCMC
$\hat{H}_1(\cdot)$ output	point, $\hat{y}_1$	$(0.0, 15.5)$	SCMC
$\hat{H}_2(\cdot)$ output	point, $\hat{y}_2$	$(0.0, 17.5)$	SCMC
$\hat{H}_3(\cdot)$ output	point, $\hat{y}_3$	$(0.0, 20.0)$	SCMC
$\hat{H}_4(\cdot)$ output	point, $\hat{y}_4$	$(0.0, 32.5)$	SCMC
$\hat{H}_5(\cdot)$ output	point, $\hat{y}_5$	$(0.0, 55.0)$	SCMC

Training data, $\Delta\tau \to 0.0$ means using analytic solution of GBM.

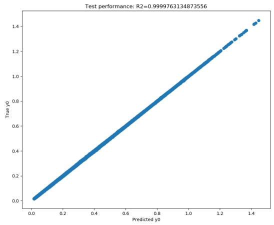
4.2 Performance of ANN-SCMC

- Approximation performance on the test dataset.

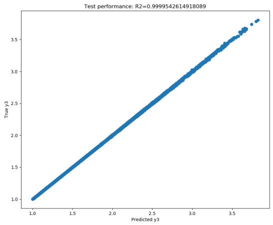
  $\hat{y}_i^a =$ ANN-SCMC, $\hat{y}_i =$ ground truth. $\hat{x}_i$ independent of time. $R^2(\hat{y}_i, \hat{y}_i^a)$ goodness of fit.

	$\hat{x_1}$	$\hat{x_2}$	$\hat{x_3}$	$\hat{x_4}$	$\hat{x_5}$
$\hat{x}_i$	-2.8570	-1.3556	0.0	1.3556	2.8570
$F_X(\hat{x}_i)$	0.0021	0.0876	0.50	0.9124	0.9979
$R^2(\hat{y}_i, \hat{y}_i^a)$	0.99997	0.99999	0.99997	0.99995	0.99951

- Predicted vs true collocation points, for example,



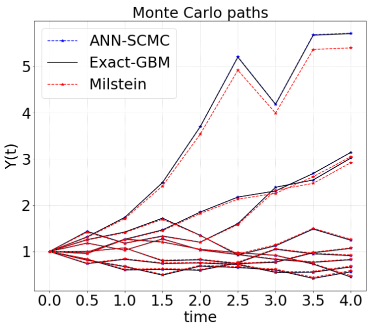
(a) $\hat{y}_2^a$ vs $\hat{y}_2$



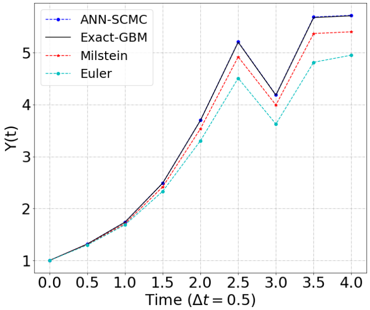
(b) $\hat{y}_3^a$ vs $\hat{y}_3$

4.3 Path-wise error

- Monte Carlo path-wise error:



(c) Monte Carlo paths

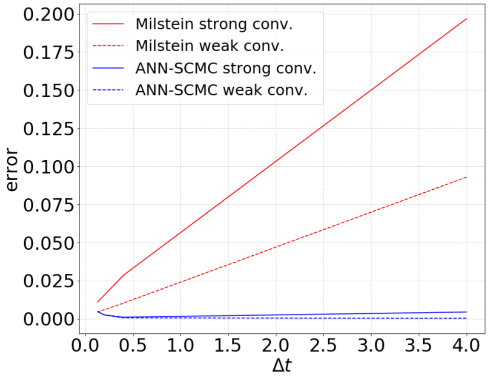


(d) Comparing Euler

$$\sigma = 0.3,\ r = 0.1,\ S_0 = 1.0,\ \Delta t = 0.5.$$

4.4 Convergence error

- Comparing strong convergences based on the closed-form solution of GBM.



Varying time step size, the averaged error over 1000 samples.

4.5 Application: Option pricing

- For Bermudan options, the holder has the right to exercise the contract at pre-specified dates up to maturity.
- Price Bermudan put options via Longstaff-Schwartz approach.
- Setting: under the risk-neutral measure, stock price $Y_0 = 1.0$, risk-less interest rate $r = 0.1$, strike $K = 1.1$, terminal time $T = \Delta t \times M_B$, the number of monitoring dates M_B, time step Δt, the number of sample paths $N = 100000$.

4.6 Bermudan option with big time steps

- Reference value V_{ref}=Analytic MC, from analytical paths of GBM; Relative error $|\frac{V_{ref}-V}{V_{ref}}|$; Time step $\Delta t = 0.4$.

	method	M_B=5	M_B=10	M_B=20
	Analytic MC	0.14806 (0.00%)	0.16313 (0.00%)	0.17338 (0.00%)
	Milstein MC	0.14353 (3.06%)	0.15734 (3.55%)	0.16642 (4.01%)
σ=0.30	ANN-SCMC	0.14780 (0.04%)	0.16305 (0.05%)	0.17322 (0.09%)
	Analytic MC	0.19716 (0.00%)	0.22464 (0.00%)	0.24454 (0.00 %)
	Milstein MC	0.19069 (3.29%)	0.21616 (3.78%)	0.23448 (4.11 %)
σ=0.40	ANN-SCMC	0.19711 (0.03%)	0.22454 (0.04%)	0.24438 (0.06%)

5. Summary and Outlook

Conclusions:

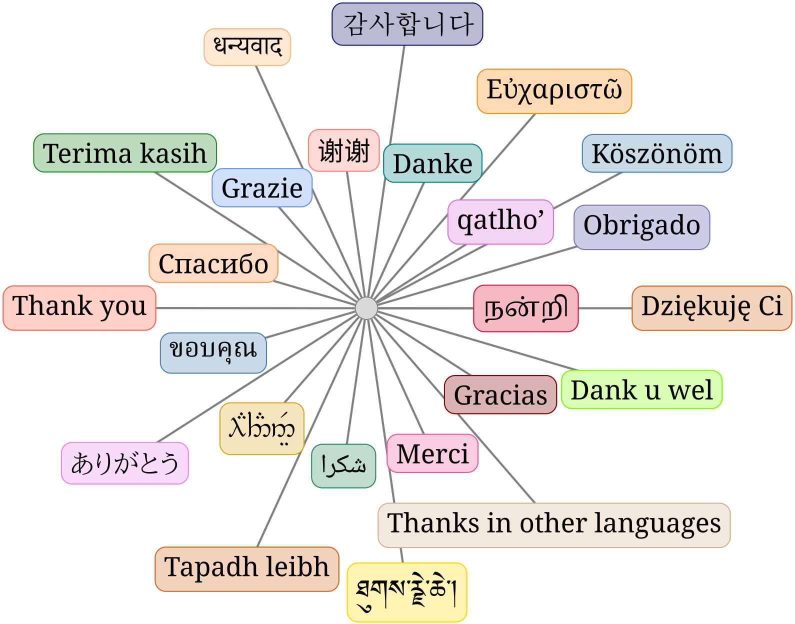
- ANN-SCMC provides a data-driven numerical solver for SDEs;
- Implement large time-step simulations with a small error in strong convergence;
- More accurate than classical schemes.

Ongoing work:

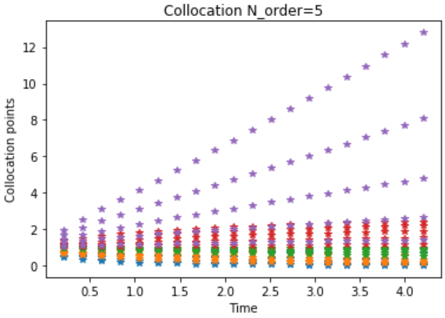
- Test ANN-SCMC on more complicated SDEs;
- Increase the computation speed.

References

1. G. Cybenko, etc(1989). Approximations by superpositions of sigmoidal functions, Mathematics of Control, Signals and Systems.
2. R. H. Cameron and W. T. Martin (1947). The Orthogonal Development of Non-Linear Functionals in Series of Fourier-Hermite Functionals. Annals of Mathematics.
3. Eckhard Platen (1999). An introduction to numerical methods for stochastic differential equations. Acta Numerica.
4. L. A. Grzelak, etc (2019). The stochastic collocation Monte Carlo sampler. Quantitative Finance.
5. Y. Bar-Sinai, etc (2019). Learning data-driven discretizations for partial differential equations. PNAS.
6. Dmitry Yarotsky (2017). Error bounds for approximations with deep ReLU networks. Neural Networks.


감사합니다
धन्यवाद
Εὐχαριστῶ
Terima kasih
谢谢
Danke
Köszönöm
Grazie
qatlho’
Obrigado
Спасибо
Thank you
நன்றி
Dziękuję Ci
ขอบคุณ
Gracias
Dank u wel
ありがとう
شكرا
Merci
Thanks in other languages
Tapadh leibh
ཐུགས་རྗེ་ཆེ།

Backup: Stochastic collocation points



(e) Order-5