Engineering Mechanics Institute Conference 2018
American Society of Civil Engineers
May 29 – June 1, 2018
Cambridge, Massachusetts, USA

An Efficient Solution Approach for High-Dimensional Random PDEs Using Stochastic Gradient Descent and Deep Neural Networks

Mohammad Amin Nabian, Hadi Meidani

Department of Civil and Environmental Engineering
University of Illinois at Urbana-Champaign

Uncertainty Quantification Group
at the University of Illinois at Urbana-Champaign

Contents

Random PDEs

A PDE is a differential equation that contains unknown multivariable functions and their partial derivatives.

Reliable simulation of systems represented by PDEs requires taking into account the uncertainties in system inputs, and quantify their impact on the QoI's.

The sources of uncertainty include:

- Initial or boundary conditions
- Material properties
- External forces

Solving Random PDEs

Developing efficient numerical methods for high-dimensional random PDEs has been a longstanding challenge.

Traditionally, a number of methods are being implemented:

- Monte Carlo Sampling (MCS) [1]
- Perturbation methods [2]
- Operator-based methods [3]
- Moment equations methods [4]
- Generalized Polynomial Chaos (gPC) [5,6]

Motivation

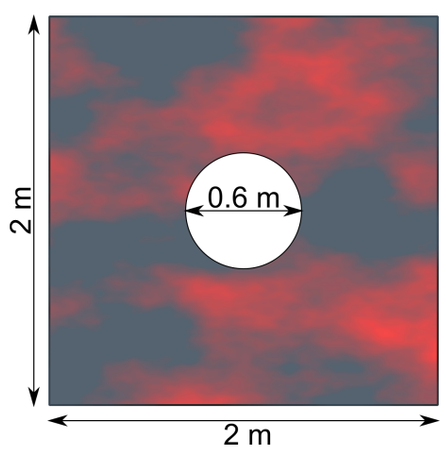
Available methods are still suffering from some shortcomings:

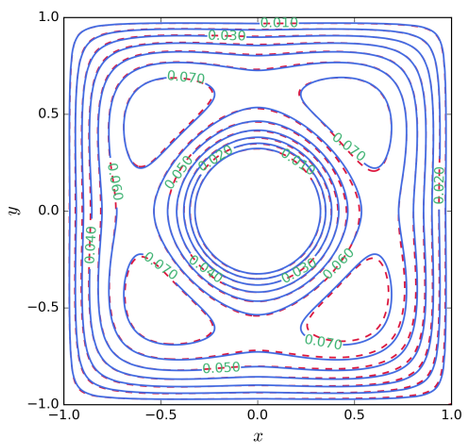
- Perturbation methods are applicable only when the magnitude of uncertainty in random coefficients and PDE output is small.
- Operator-based methods are also limited to small uncertainties, and they also are typically restricted to time-independent problems.
- Moment equation methods suffer from the so-called closure problem.
- gPC methods are divided into two methods of stochastic Galerkin and stochastic collocation
  - stochastic Galerkin method is cumbersome to implement, and when the PDE takes highly complex and non-linear form, the explicit derivation of the Galerkin system is non-trivial.
  - stochastic collocation method introduces errors arising from the integration rule or the interpolation scheme, and in high-dimensions, these errors can be potentially very significant.

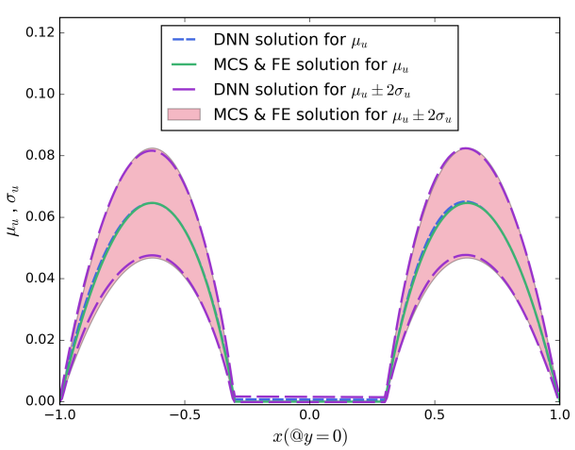
Motivation

We present a new framework for solving high-dimensional random PDEs based on a deep learning approach. The random PDE is approximated by a deep neural network.

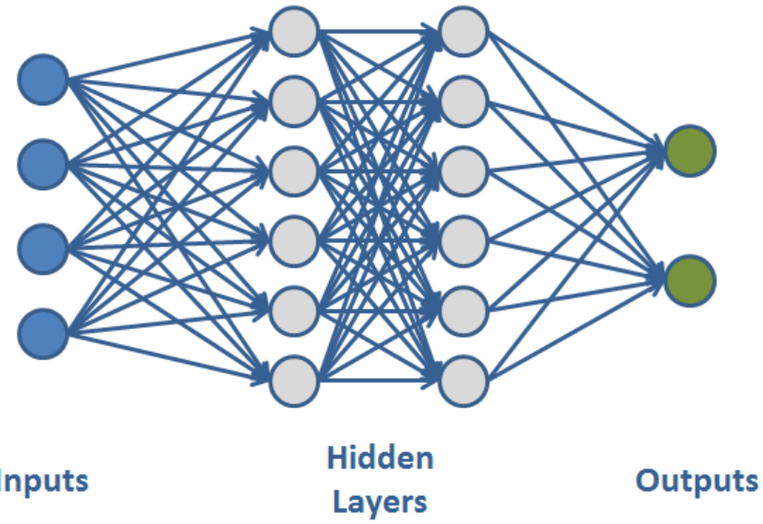
- The method is suitable for variety of random PDEs with any arbitrary magnitude of uncertainty.
- The method eliminates the need for interpolation in high dimensions.
- The method is very straightforward to formulate, and minimal problem-dependent setup is required.
- The solution has a closed analytical form and is infinitely differentiable and therefore can be easily used in a variety of subsequent calculations (e.g. sensitivity analysis).
- The proposed algorithm is embarrassingly parallel on GPUs.
- The proposed method gives us the capability to train an approximate solution offline, and use the surrogate online given different realizations of the variables.
- The framework is mesh-free, and can handle irregular domains.







Deep Neural Networks

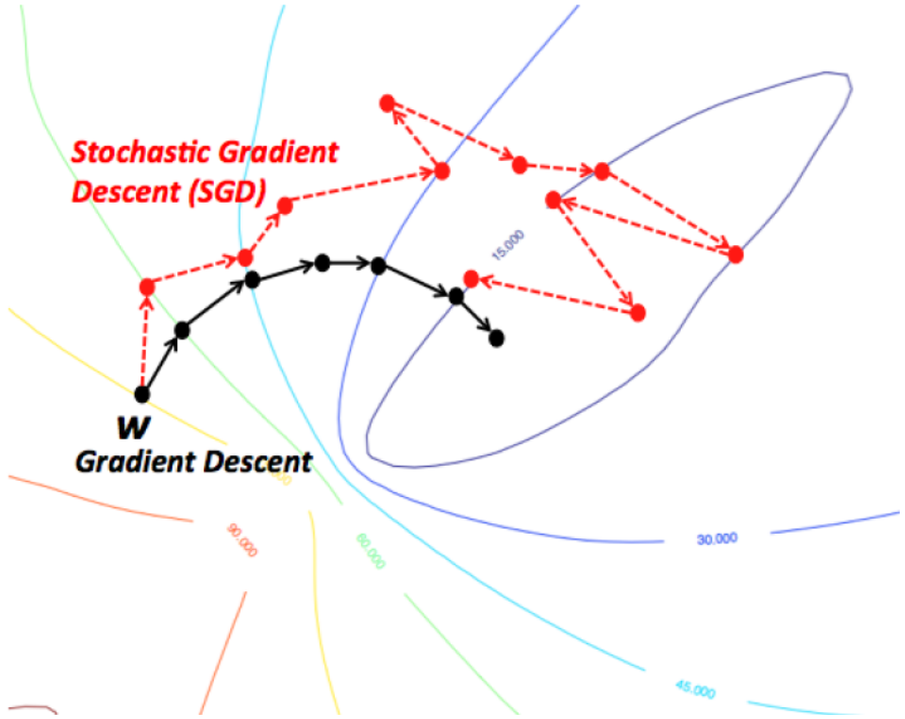


- Single hidden layer neural networks are introduced first.
- Generalization to multiple hidden layers, which makes a neural network deep, is straightforward.
- Given the d-dimensional row vector $\boldsymbol{x}$ as model input, the k-dimensional output of a standard single hidden layer neural network is

$$\boldsymbol{y} = \sigma(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$

- W_1 and W_2 are weight matrices of size $d \times q$ and $q \times k$, respectively.
- b_1 and b_2 are biases of size $1 \times q$ and $1 \times k$, respectively.
- The function $\sigma(\cdot)$ is an element-wise non-linearity.
- For each new set of weight matrix and bias that is added to the above equation, a new hidden layer is added to the neural network.

Stochastic Gradient Descent



http://www.bogotobogo.com/python/scikit-learn/
scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php

- Batch gradient descent computes the gradient using the whole dataset.
  - This is great for convex, or relatively smooth error manifolds.
  - In this case, we move somewhat directly towards an optimum solution, either local or global.

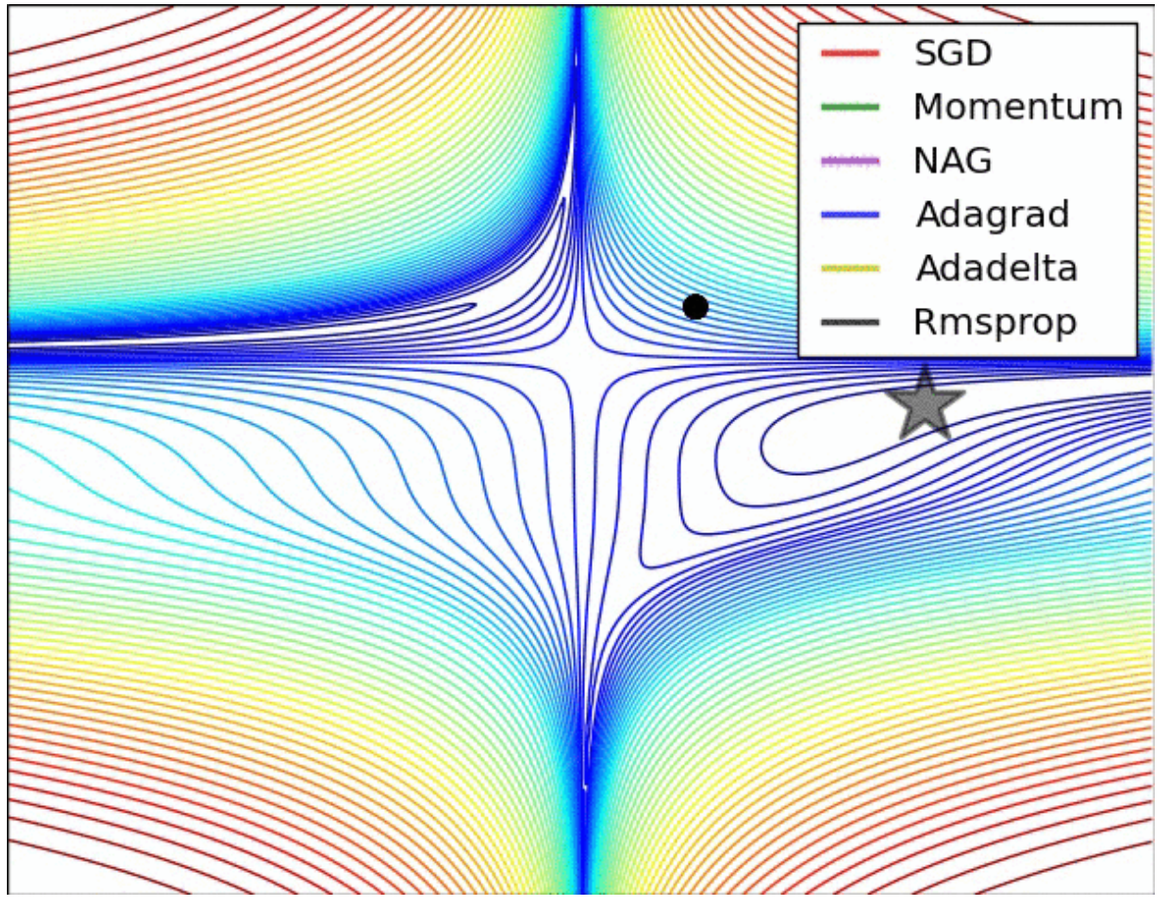
- Stochastic gradient descent approximates the true gradient of the loss function by a gradient at a single example.
  - SGD works well for error manifolds that have lots of local maxima/minima.
  - The noisier gradient tends to take the model out of local minima.

- mini-batch gradient descent tend to average a little of the noise out as single samples are really noisy.
  - A good balance is struck when the mini-batch size is small enough to avoid some of the poor local minima, but large enough that it doesn't avoid the global minima or better-performing local minima.

SGD variants

SGD variants are proposed to accelerate convergence, mainly by adaptively controlling the learning rate. [7]

- Momentum
- NAG
- Adagrad
- Adadelta
- Rmsprop
- Adam



http://ruder.io/optimizing-gradient-descent/

The Proposed Method

In a general random differential equation, the solution *u (t, x, P)* satisfies the following three equalities

$$\mathcal{L}(t, \boldsymbol{x}, \boldsymbol{p}; u(t, \boldsymbol{x}, \boldsymbol{p}; \boldsymbol{\Theta})) = 0, \quad t \in [0, T], \boldsymbol{x} \in \mathcal{D}, \boldsymbol{p} \in \mathbb{R}^d,$$

$$\mathcal{I}(\boldsymbol{x}, \boldsymbol{p}; u(0, \boldsymbol{x}, \boldsymbol{p}; \boldsymbol{\Theta})) = 0, \quad \boldsymbol{x} \in \mathcal{D}, \boldsymbol{p} \in \mathbb{R}^d,$$

$$\mathcal{B}(t, \boldsymbol{x}, \boldsymbol{p}; u(t, \boldsymbol{x}, \boldsymbol{p}; \boldsymbol{\Theta})) = 0, \quad t \in [0, T], \boldsymbol{x} \in \partial\mathcal{D}, \boldsymbol{p} \in \mathbb{R}^d$$

- The framework provides a global approximating function $u_{DNN}(t,x,P;\Theta)$ for solution $u(t, x, P)$.
- Approximate solution is in form of a feed-forward fully-connected deep neural network.
- A least square formulation is used to find the parameters Θ.

Let us first define the residuals as

$$
\begin{aligned}
r_{\mathcal{L}}(\boldsymbol{\Theta}) &= \int_{[0,T]\times\mathcal{D}\times\mathbb{R}^d} (\mathcal{L}(t,\boldsymbol{x},\boldsymbol{p};\boldsymbol{\Theta}))^2 \rho_{\boldsymbol{p}} \, \mathrm{d}t \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{p}, \\
r_{\mathcal{I}}(\boldsymbol{\Theta}) &= \int_{\mathcal{D}\times\mathbb{R}^d} (\mathcal{I}(\boldsymbol{x},\boldsymbol{p};\boldsymbol{\Theta}))^2 \rho_{\boldsymbol{p}} \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{p}, \\
r_{\mathcal{B}}(\boldsymbol{\Theta}) &= \int_{[0,T]\times\partial\mathcal{D}\times\mathbb{R}^d} (\mathcal{B}(t,\boldsymbol{x},\boldsymbol{p};\boldsymbol{\Theta}))^2 \rho_{\boldsymbol{p}} \, \mathrm{d}t \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{p}.
\end{aligned}
$$

We seek to find the optimal parameters Θ^* such that

$$
\begin{aligned}
\boldsymbol{\Theta}^* &= \underset{\boldsymbol{\Theta}}{\operatorname{argmin}} \, r_{\mathcal{L}}(\boldsymbol{\Theta}), \\
\text{s.t.} \quad & r_{\mathcal{I}}(\boldsymbol{\Theta}) = 0, \, r_{\mathcal{B}}(\boldsymbol{\Theta}) = 0.
\end{aligned}
$$

- Boundary or initial constraints in such formulation can in general be treated as soft or hard constraints.
- In soft assignment, the optimal DNN parameters are computed such that

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta}}{\operatorname{argmin}} \underbrace{r_{\mathcal{L}}(\boldsymbol{\Theta}) + \lambda_1 r_{\mathcal{I}}(\boldsymbol{\Theta}) + \lambda_2 r_{\mathcal{B}}(\boldsymbol{\Theta})}_{J_s(\boldsymbol{\Theta};u_s)}$$

- In hard assignment, we let the approximate solution take the following general form

$$u_h(t, \boldsymbol{x}, \boldsymbol{p}; \boldsymbol{\Theta}) = C(t, \boldsymbol{x}) + G(t, \boldsymbol{x}, u_{\text{DNN}}(t, \boldsymbol{x}, \boldsymbol{p}; \boldsymbol{\Theta})),$$

- And the optimal DNN parameters are computed such that

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta}}{\operatorname{argmin}} \underbrace{r_{\mathcal{L}}(\boldsymbol{\Theta})}_{J_h(\boldsymbol{\Theta};u_h)}$$

- The loss function is constructed in such a way to reduce the random PDE problem to an unconstrained optimization problem.
- To solve the resulting unconstrained optimization problem, we make use of stochastic gradient descent optimization (SGD) algorithms.
- In each iteration, SGD approximates the gradient of loss function using only one point in the input space, and update the neural network parameters using the approximate gradient.
- The iterative update in stochastic gradient descent algorithms results in an unbiased estimation of the gradient, with bounded variance.

Instead of minimizing $J_h(\Theta)$, we will minimize a sampled-based loss function iteratively.

On the i^{th} iteration, a random input point $\{t^{(i)},x^{(i)},P^{(i)}\}$ is generated uniformly from $[0, T] \times D$, and from I_P according to ρ_P, and the sample-based loss function is formed as follows

$$\tilde{J}_h^{(i)}(\boldsymbol{\Theta}) = \left[\mathcal{L}(t^{(i)}, \boldsymbol{x}^{(i)}, \boldsymbol{p}^{(i)}; u_h(t^{(i)}, \boldsymbol{x}^{(i)}, \boldsymbol{p}^{(i)}; \boldsymbol{\Theta}))\right]^2$$

and a stochastic descent step is taken

$$\boldsymbol{\Theta}^{(i+1)} = \boldsymbol{\Theta}^{(i)} - \eta^{(i)} \nabla_{\boldsymbol{\Theta}} \tilde{J}_h^{(i)}(\boldsymbol{\Theta})$$

Example1: Diffusion Process

In this example, we consider a diffusion process in the following form

$$\nabla \cdot (a(t, x, \mathbf{P}) \nabla u(t, x, \mathbf{P})) = c, \quad t \in [0, 1], x \in [0, 1], \mathbf{P} \in [-1, 1]^d$$

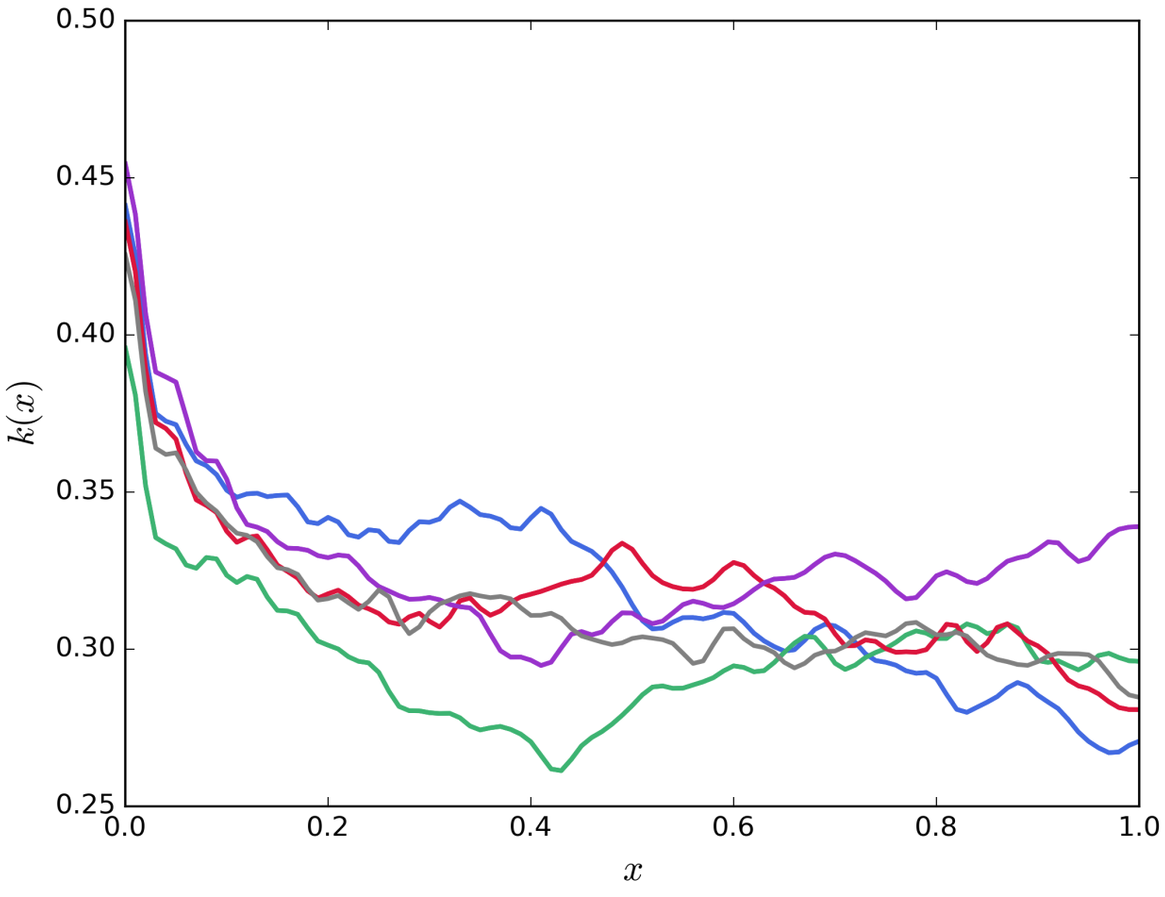
$$u(t, 0, \mathbf{P}) = 0,\ u(t, 1, \mathbf{P}) = 0,\ u(0, x, \mathbf{P}) = 10(x - x^2)$$

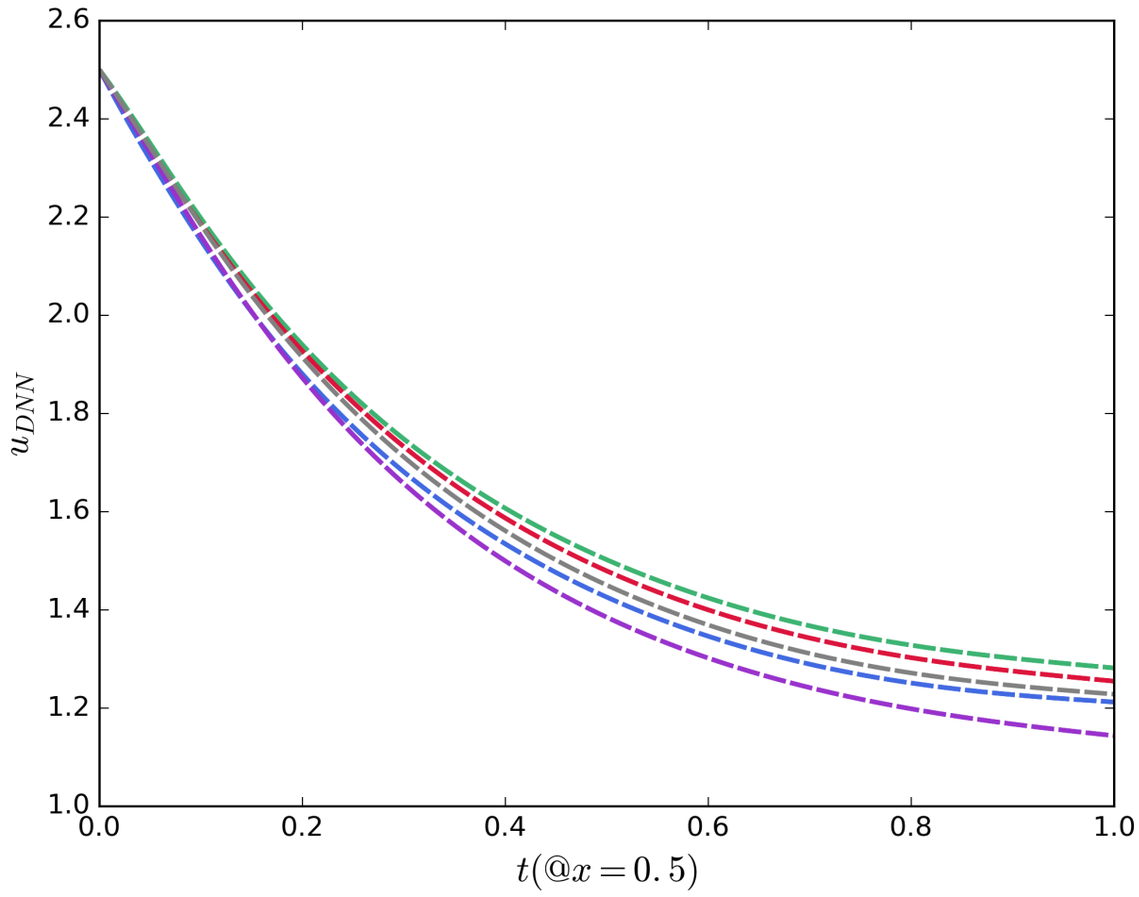
in which d is the dimensionality of stochastic domain and is set to 50, c is a constant and is set to 3, and $a\ (t, x, P)$ is the diffusion coefficient which is assumed to have the following analytical form

$$a(x, \mathbf{P}) = 0.2 + \sum_{k=1}^{d} \frac{0.1}{k} \cos^2\left(\frac{\pi}{2} k x\right) \xi_k$$

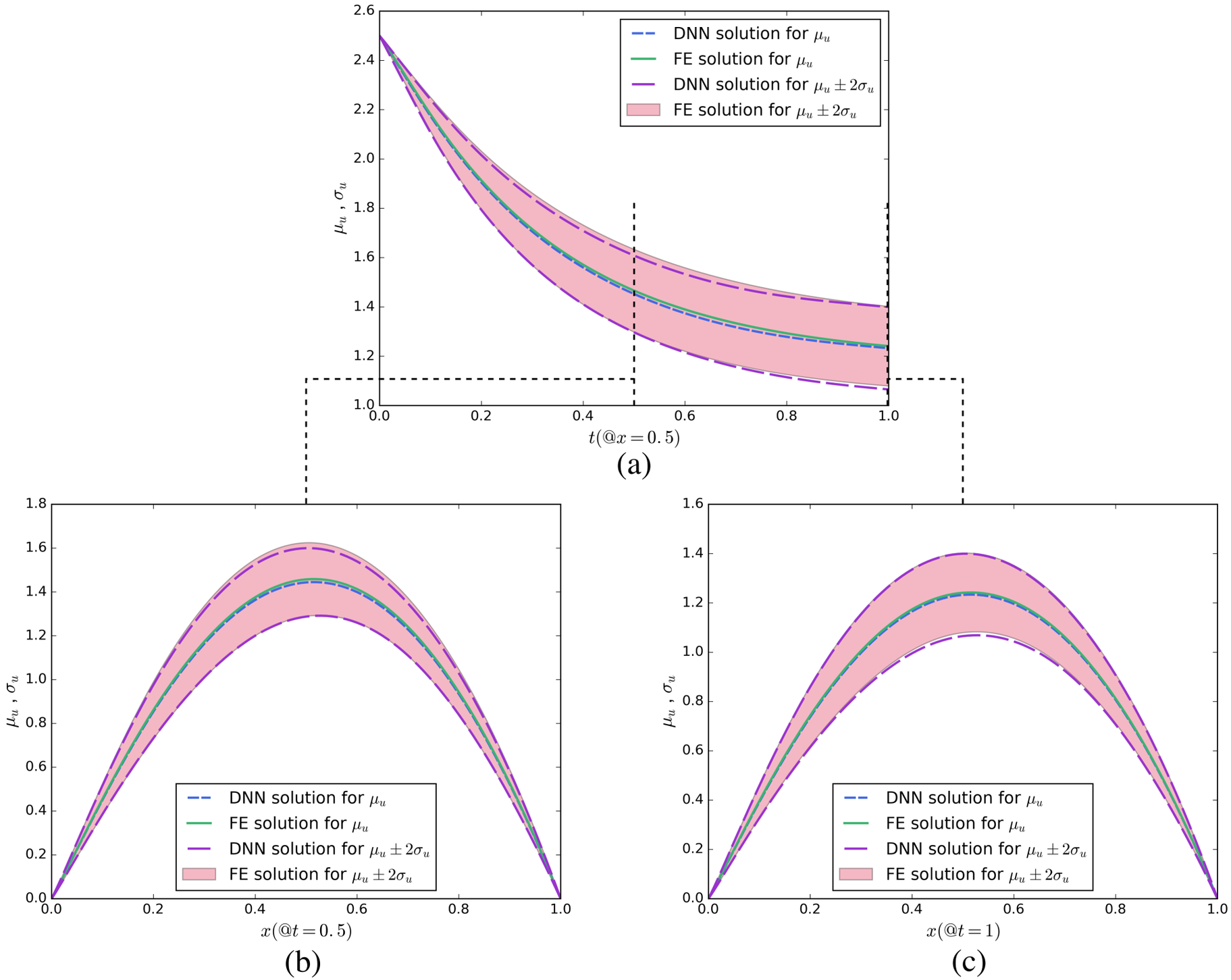
Random variables are independent identically distributed with a uniform distribution on [0, 1].

Sampled trajectories

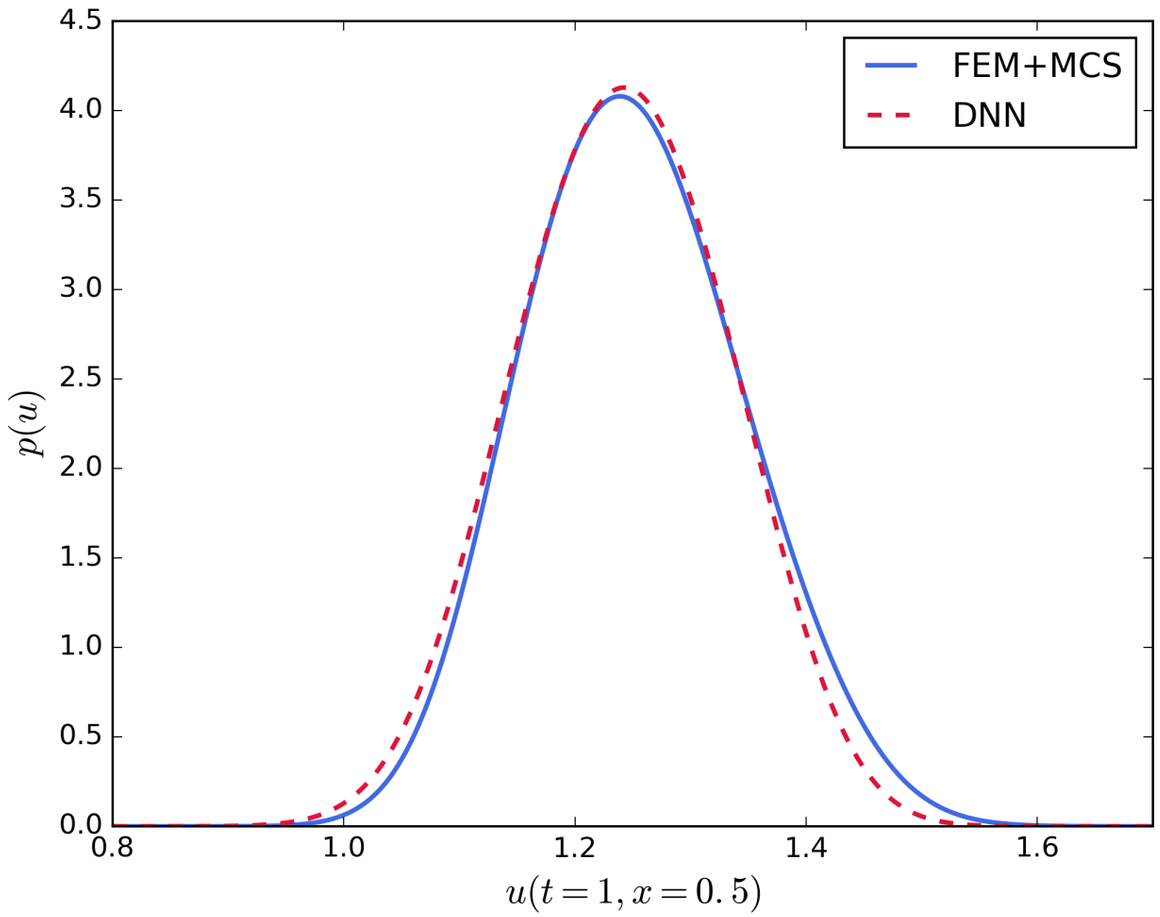




Response statistics



Probability density function



Example2: Steady Heat Conduction

In this example, we consider an steady heat conduction problem in the following form

$$\nabla.\left(k\left(x,y,\mathbf{P}\right)\nabla u\left(x,y,\mathbf{P}\right)\right)=f\left(x,y\right),\quad x,y\in\left[-1,1\right],\mathbf{P}\in\left[0,1\right]^{d}$$

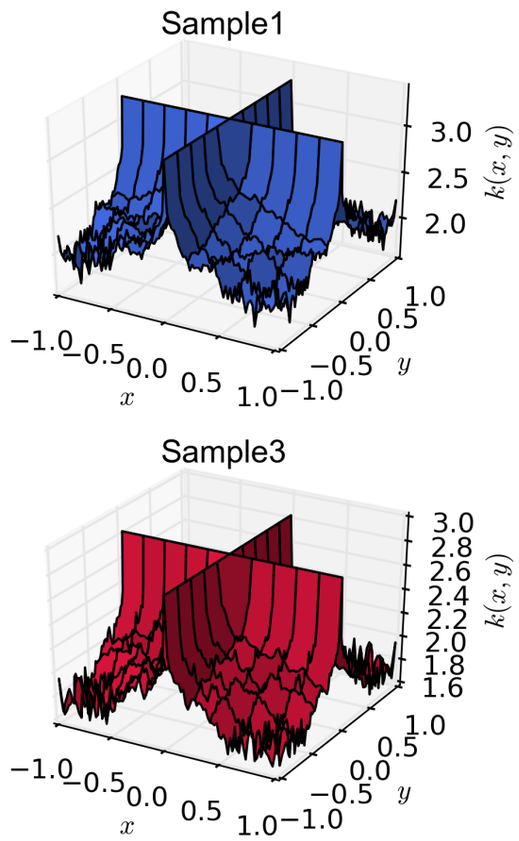
$$u\left(-1,y,\mathbf{P}\right)=u\left(1,y,\mathbf{P}\right)=u\left(x,-1,\mathbf{P}\right)=u\left(x,1,\mathbf{P}\right)=0,$$

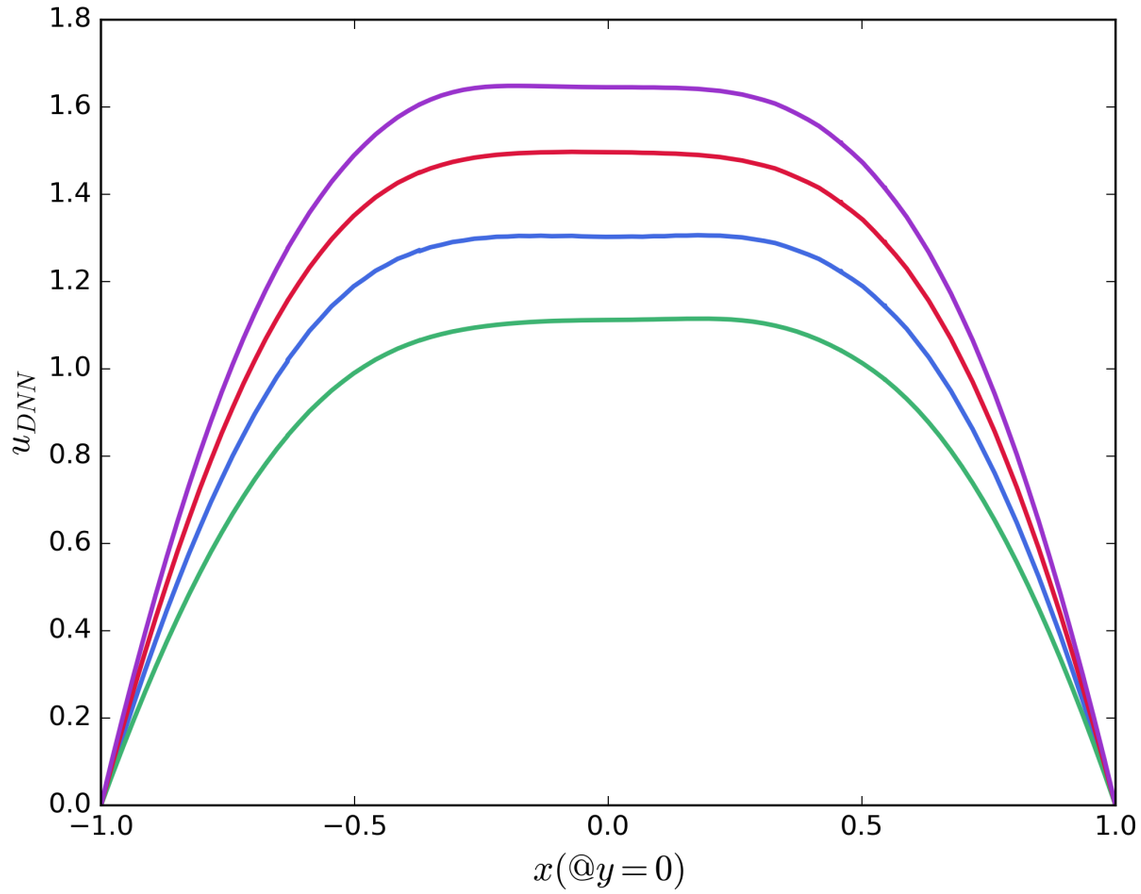
in which *d* is the dimensionality of stochastic domain and is set to 50, *c* is a constant and is set to *100|xy|*, and k *(x, P)* is the conduction coefficient which is assumed to have the following analytical form

$$k\left(x,y,\mathbf{P}\right)=1+\sum_{k=1}^{d}\frac{1}{k}\cos^{2}\left(\frac{\pi}{4}k^{\frac{3}{2}}xy\right)\xi_{k}$$

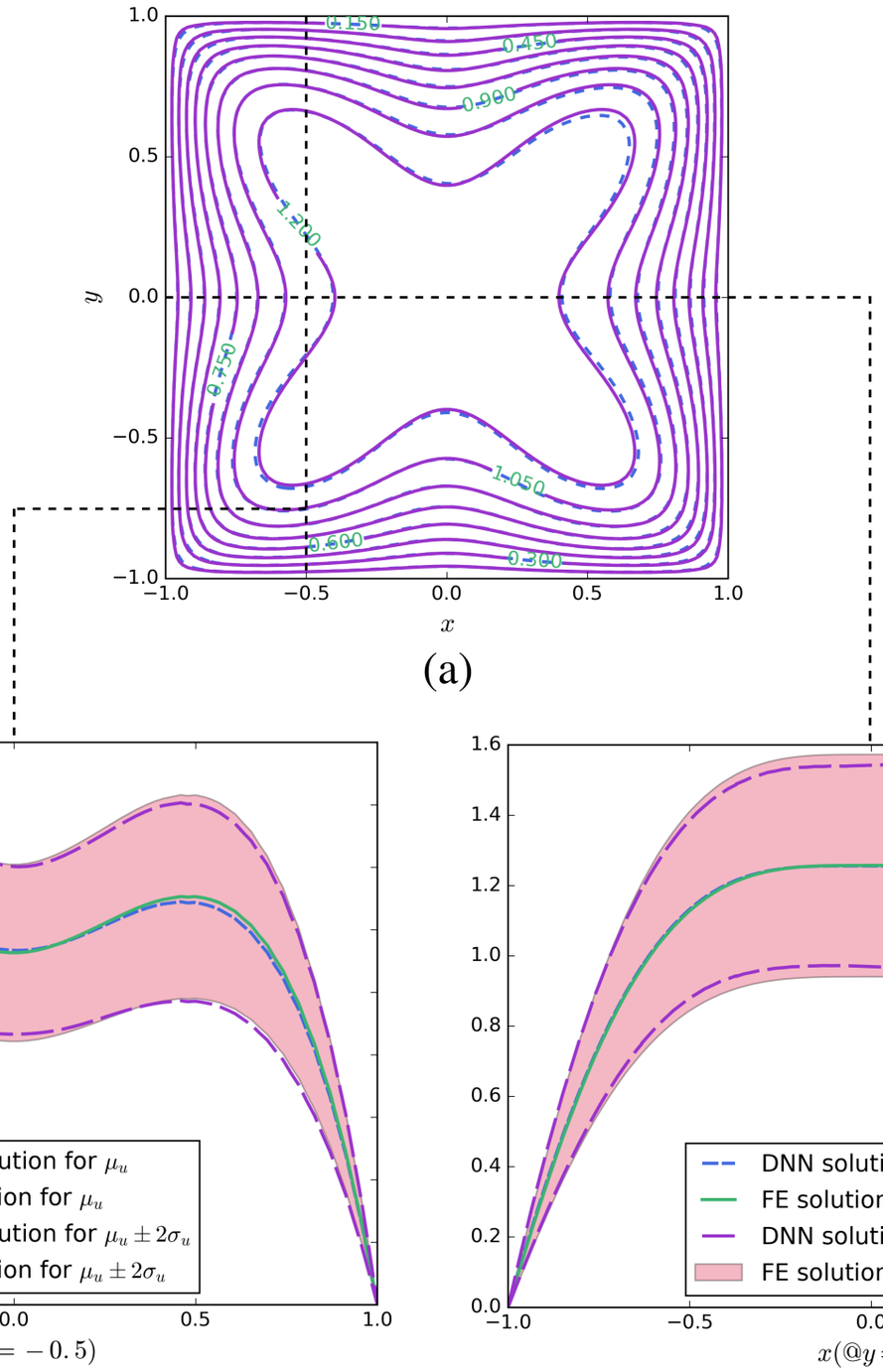
Random variables are independent identically distributed with a uniform distribution on [0, 1].

Sampled trajectories

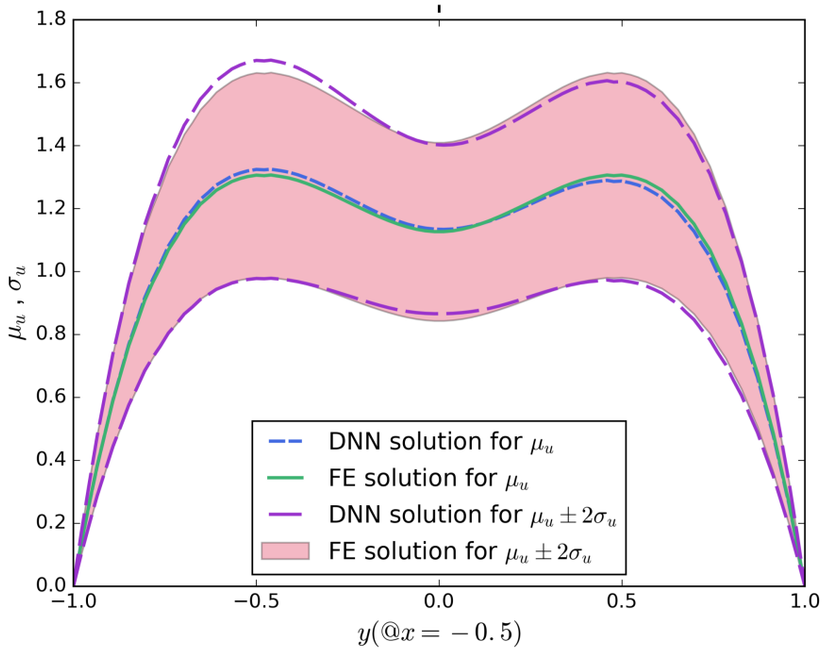




Response statistics



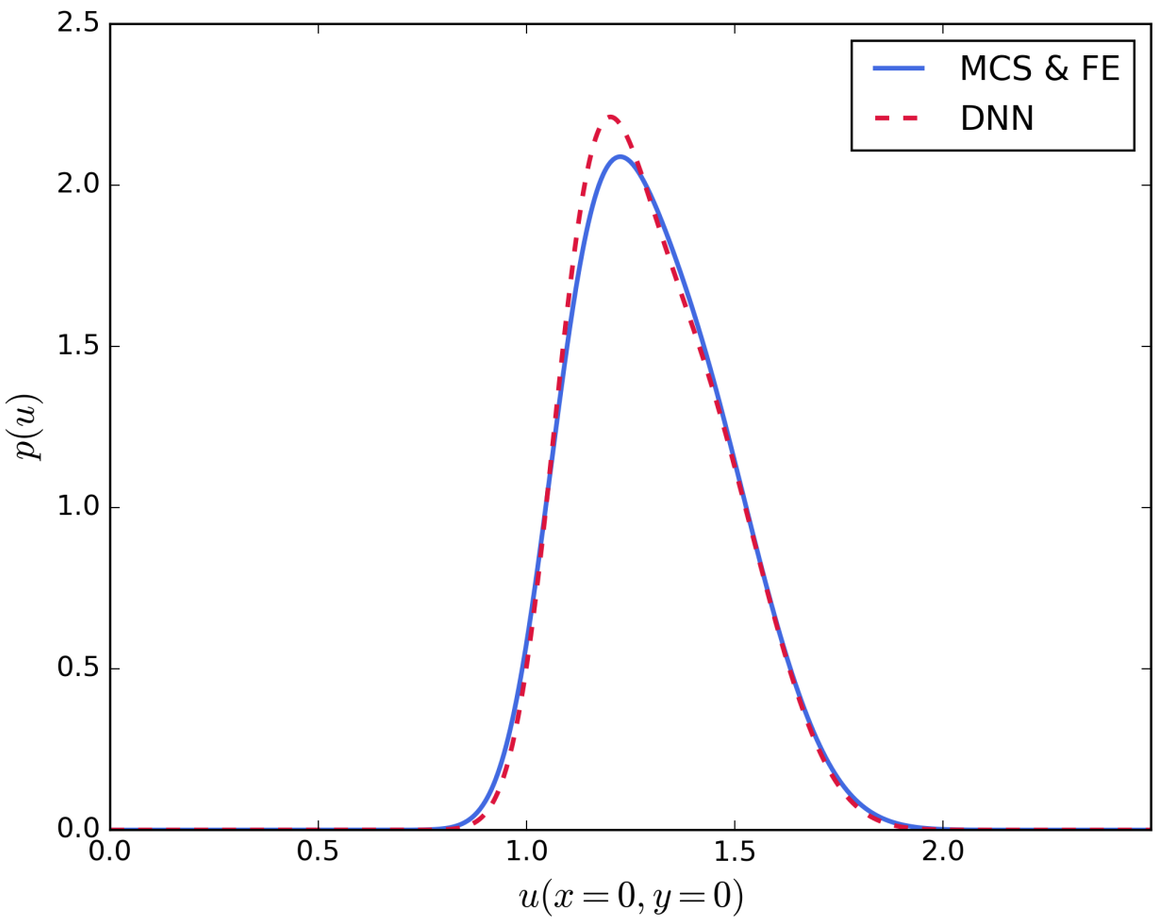
(a)

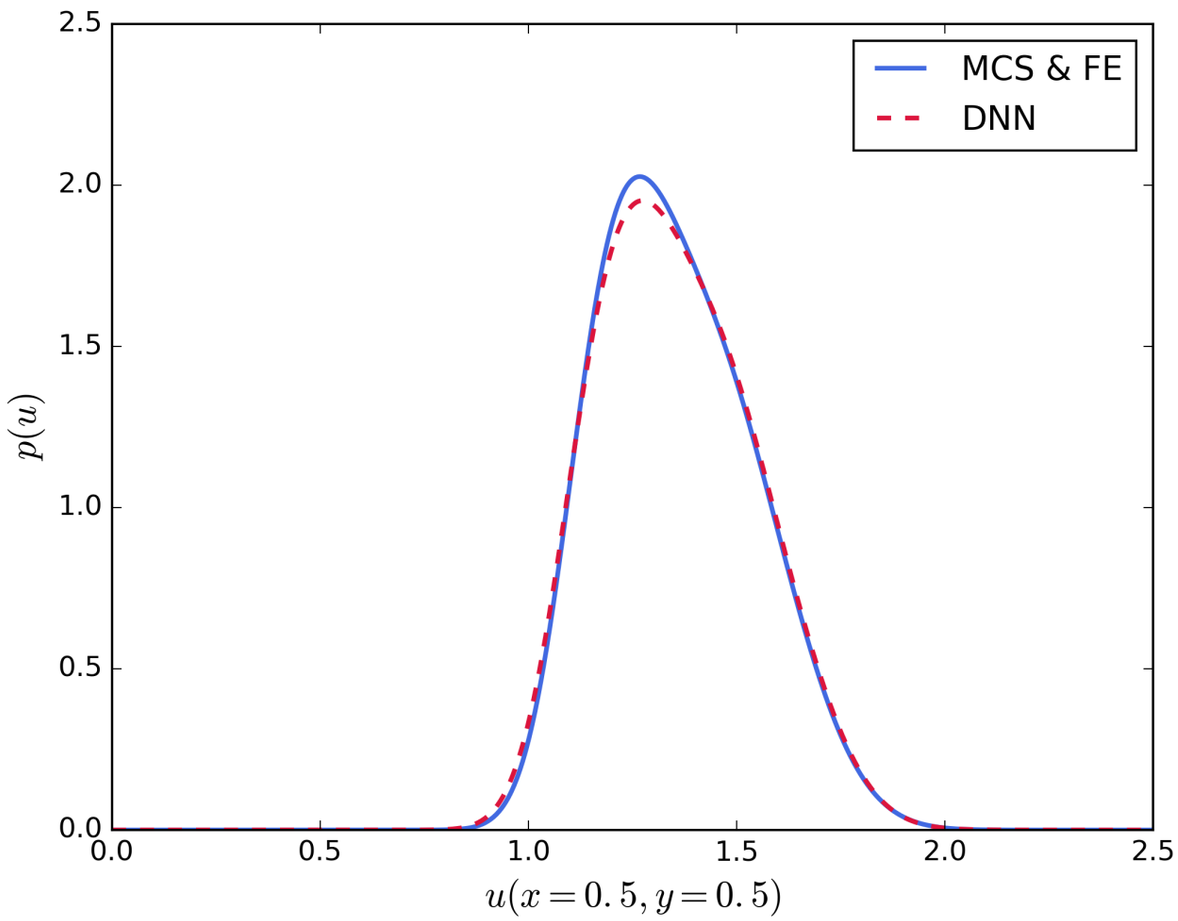


(b)

(c)

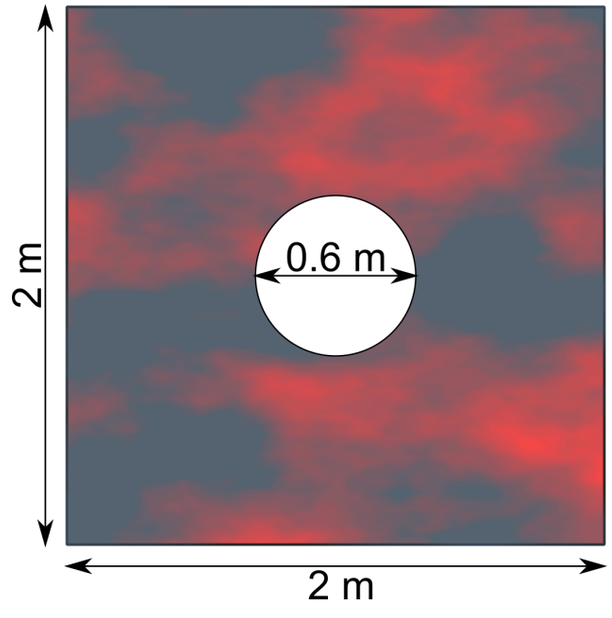
Probability density function



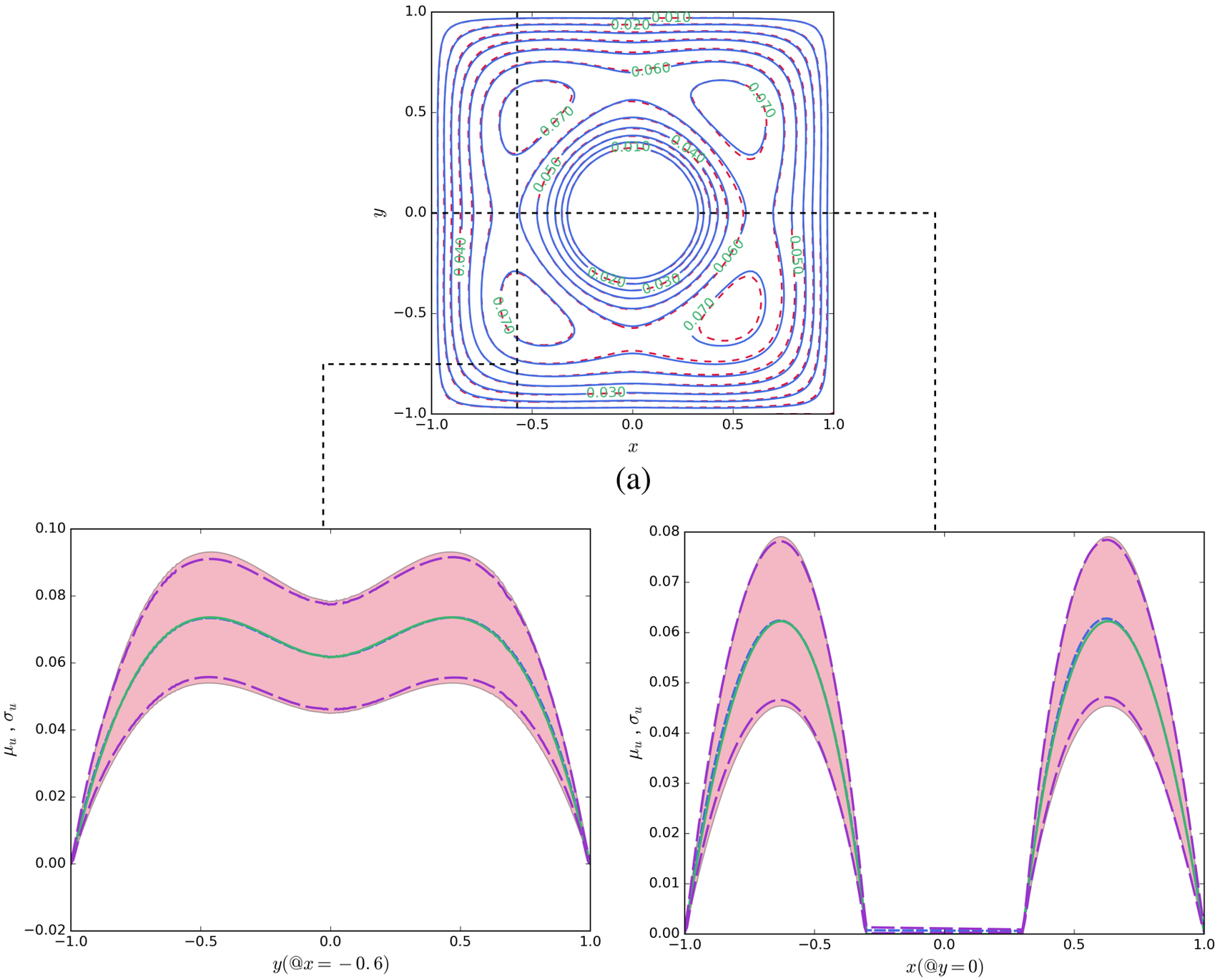


Example3: Steady Heat Conduction on an Irregular Domain

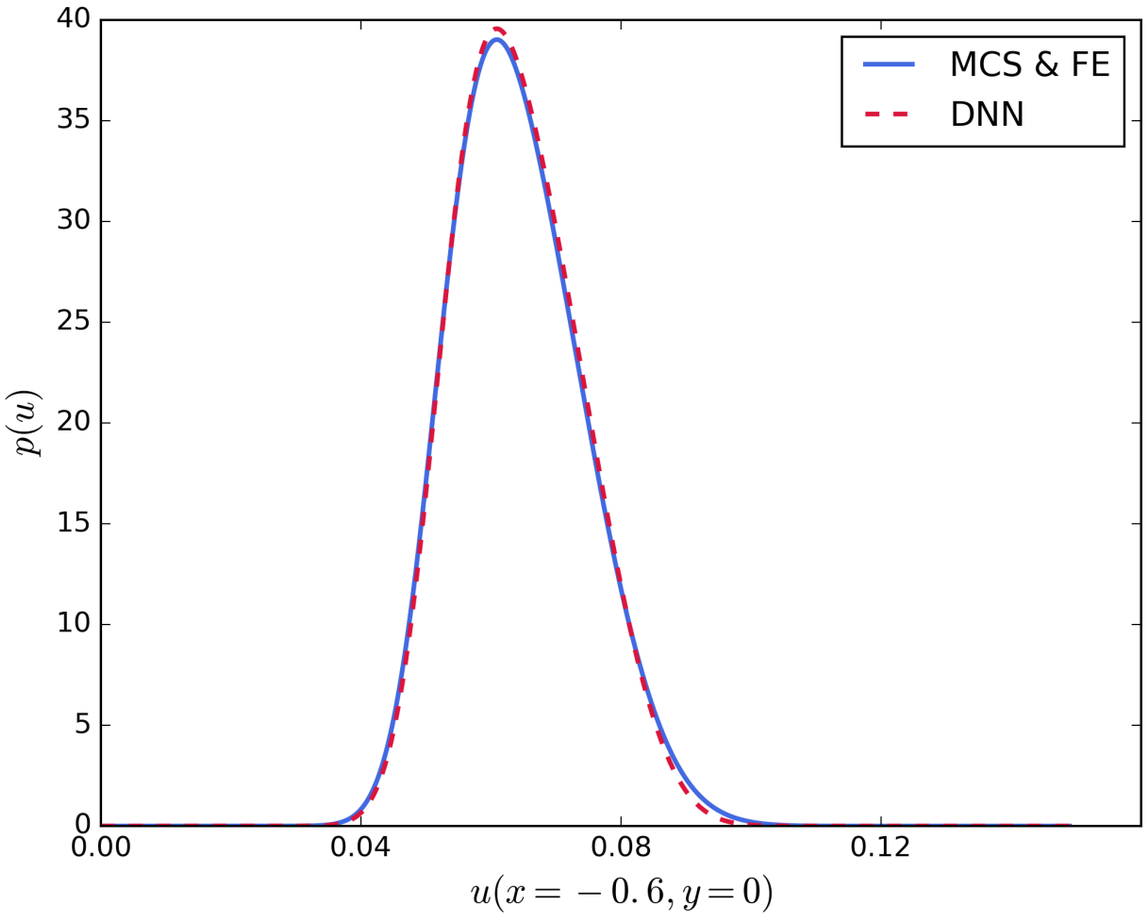
In this example, we consider the same steady heat conduction problem represented in the previous example, on a different domain. *d* is set to 30. Zero Dirichlet boundary condition is assumed around the boundaries.

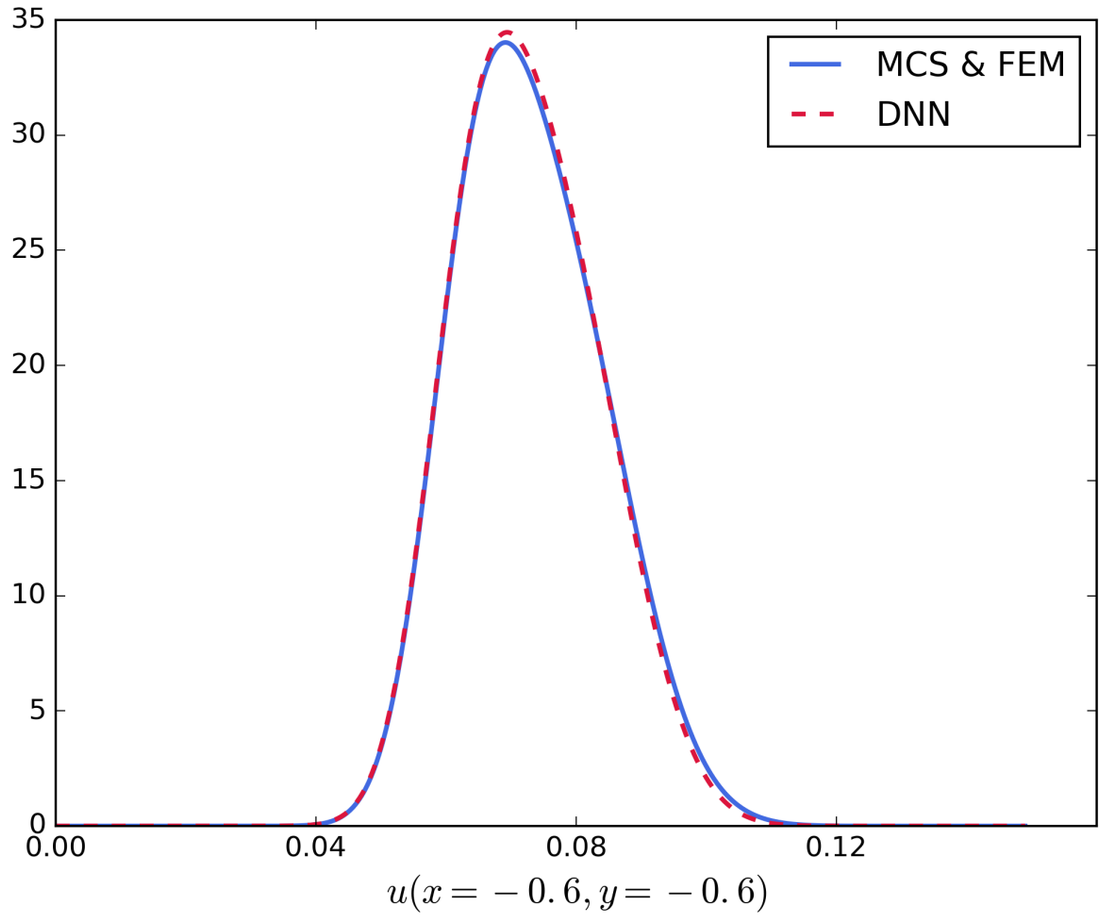


Response statistics



Probability density function





Conclusion

- In the proposed method, the solution to random PDE is approximated by a feed-forward fully-connected deep neural network, with either strong or weak enforcement of initial and boundary constraints.
- The framework is mesh-free.
- Parameters of the approximating deep neural network are determined iteratively using variants of the Stochastic Gradient Descent (SGD) algorithm.
- The performance of the proposed framework in accurate estimation of random PDEs is examined by implementing it for diffusion and heat conduction problems.
- Results are compared with the sampling-based finite element results, and suggest that the proposed framework achieves satisfactory accuracy and can handle high-dimensional random PDEs.

The advantages of using the proposed framework for solving random PDEs include:

(1) The DL solution to random differential equations has a closed analytic form (as opposed to, e.g. Monte Carlo methods) and is infinitely differentiable, and therefore can be easily used in a variety of subsequent calculations,

(2) The proposed algorithm is embarrassingly parallel on Graphical Processing Units (GPUs).

(3) The proposed algorithm is very straightforward to formulate, and minimal problem-dependent setup is required before computations (as opposed to Stochastic Galerkin method for gPC).

(4) The proposed method is general and can be utilized for a variety of random ODEs and random PDEs.

(5) DL solution is valid over the entire computational domain and eliminates the need for interpolation.

References

[1] G. Fishman, Monte Carlo: concepts, algorithms, and applications, Springer Science & Business Media, 2013.

[2] W. K. Liu, T. Belytschko, A. Mani, Probabilistic finite elements for nonlinear structural dynamics, Computer Methods in Applied Mechanics and Engineering 56 (1) (1986) 61–81.

[3] G. Deodatis, Weighted integral method. i: stochastic stiffness matrix, Journal of Engineering Mechanics 117 (8) (1991) 1851–1864.

[4] D. Zhang, Stochastic methods for flow in porous media: coping with uncertainties, Elsevier, 2001.

[5] D. Xiu, Numerical methods for stochastic computations: a spectral method approach, Prince- ton university press, 2010.

[6] D. Xiu, G. E. Karniadakis, The wiener–askey polynomial chaos for stochastic differential equa- tions, SIAM journal on scientific computing 24 (2) (2002) 619–644.

[7] Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. Vol. 1. Cambridge: MIT press, 2016.

Thank you!