УДК 519.2 URL: https://ptsj.bmstu.ru/catalog/math/compmath/1050.html

РЕАЛИЗАЦИЯ СТОХАСТИЧЕСКОГО МЕТОДА ГАЛЕРКИНА К АППРОКСИМИРОВАНИЮ ФУНКЦИЙ, ЗАВИСЯЩИХ ОТ СЛУЧАЙНЫХ ПАРАМЕТРОВ

М.Х. Хаписов khapisovmkh@student.bmstu.ru

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Рассмотрены возможности применения концепции полиномиального хаоса в исследовании динамических систем со случайными параметрами. С целью максимального учета вида линейного дифференциального оператора, описывающего систему, при вычислении коэффициентов использован интрузивный метод стохастической проекции Галеркина. Метод реализован на языках программирования C++ и Python, численное интегрирование выполнено методом Симпсона на неравномерной сетке. На тестовых функциях проведено сравнение эффективности работы реализованных алгоритмов. Метод был применен к решению задачи о линейном затухающем осцилляторе, на которой была оценена точность аппроксимации.

Ключевые слова: полиномы Эрмита, разложение полиномиального хаоса, стохастический метод Галеркина, метод Симпсона, задача Коши, C++, Python

Введение. При решении широкого класса задач одним из основных методов в настоящее время является компьютерное моделирование, которое, несмотря на свое бурное развитие, далеко не всегда позволяет безошибочно предсказать поведение комплексной системы реального мира. Это по большей части объясняется несоответствием между математическим представлением и описываемым явлением, а также погрешностями вычислений и неопределенностью входных факторов.

В связи с этим все чаще сегодня прибегают к использованию стохастических методов. Например, широко используется подход на основе так называемого черного ящика, преобразующего по некоторому неизвестному и содержащему элементы случайности правилу вектор входных переменных X в отклик системы Y. При вычислении коэффициентов модели используют как классический метод линейной регрессии, так и обретающий сегодня все бóльшую популярность метод разложения полиномиального хаоса (РПХ). В этом методе входные данные X рассматривают как реализация некоторой случайной величины, что позволяет находить отклик в виде ряда по ортогональной системе полиномов, определяемой законом распределения вектора X [1]. При вычислении коэффициентов этого разложения применяют разнообразные интрузивные и неинтрузивные методы.

 ISSN 2541-8009

Цель данной работы — реализация интрузивного стохастического метода Галеркина на C++ и Python, а также сравнение эффективности алгоритмов для решения задачи Коши.

Полиномиальный хаос. Рассмотрим численную модель $Y = (\overline{X})$, в которой входной вектор $\overline{X}$ состоит из m независимых случайных величин $\overline{X} = \{X_j,\ j = \overline{1,\ m}\}$, а Y — случайный отклик модели (выходные данные). В предположении, что отклик Y имеет конечную дисперсию, он может быть записан в виде

$$Y = (\overline{X}) = \sum_{j=0}^{+\infty} y_j \varphi_j(\overline{X}), \tag{1}$$

где $\left\{\varphi_j(\overline{X}),\ j = \overline{0,\ +\infty}\right\}$ образуют базис в вероятностном пространстве, а $y_j \in \mathbb{R}$ — коэффициенты разложения случайного отклика Y по этому базису [2]. При этом функции φ_j являются полиномами от входного случайного вектора $\overline{X}$, а ряд (1) представляет собой разложение полиномиального хаоса [1].

Для расчета коэффициентов y_j в настоящей работе применяется интрузивный метод стохастической проекции Галеркина [3–5], который позволяет учесть вид линейного дифференциального оператора, описывающего модель.

Полиномы Эрмита. В случае нормального закона распределения входного вектора $\overline{X}$ отклик системы представим в виде (1), где $\varphi_j(\overline{X})$ — полиномы Эрмита, поскольку они образуют ортогональный базис в пространстве L^2, причем весовая функция совпадает с плотностью стандартного нормального распределения [6]. Ортогональные многочлены других распространенных распределений приведены в табл. 1.

Дадим некоторые пояснения к табл. 1. Функция Г, присутствующая в записи ортонормированного базиса для гамма-распределения, — это гамма-функция, определяемая как

$$\Gamma(z) = \int_0^{+\infty} t^{z-1} e^{-t} dt,$$

а функция B в записи ортонормированного базиса для бета-распределения есть не что иное, как бета-функция, определяемая как

$$B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt,$$

коэффициенты a и b представляют собой некоторые параметры, для которых определяется этот полином.

Таблица 1. **Соответствие между распределениями непрерывных случайных величин и семействами ортогональных полиномов**

Распределение	Плотность распределения	Ортогональные полиномы	Ортонормированный базис $\{\varphi_k,\ k \in \mathbb{N} \cup \{0\}\}$
Равномерное	$\frac{I_{[-1;1]}(x)}{2}$	Лежандра $P_k(x)$	$\frac{P_k(x)}{\sqrt{\frac{1}{2k+1}}}$
Нормальное	$\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$	Эрмита $\mathrm{He}_k(x)$	$\frac{\mathrm{He}_k(x)}{\sqrt{k!}}$
Гамма	$x^a e^{-x} I_{\mathbb{R}_+}(x)$	Лагерра $L_k^a(x)$	$\frac{L_k^a(x)}{\sqrt{\frac{\Gamma(k+a+1)}{k!}}}$
Бета	$\frac{(1-x)^a(1+x)^b}{B(a)B(b)} \times I_{[-1;1]}(x)$	Якоби $J_k^{a,b}(x)$	$\frac{J_k^{a,b}(x)}{\sqrt{{}^{2}_{a,b,k}}}$, ${}^{2}_{a,b,k} = \frac{2^{a+b+1}}{2k+a+b+1} \frac{\Gamma(k+a+1)\Gamma(k+b+1)}{\Gamma(k+a+b+1)\Gamma(k+1)}$

Если же случайная величина распределена по другому закону, то можно представить ее как функцию от некоторой другой случайной величины, базисные функции которой известны.

Рассмотрим подробнее полиномы Эрмита. Полиномы Эрмита — это семейство ортогональных многочленов следующего вида:

$$He_k(x) = (-1)^k e^{\frac{x^2}{2}} \frac{d^k}{dx^k} e^{-\frac{x^2}{2}}.$$

Полиномы Эрмита также удобно вычислять с помощью следующего рекуррентного соотношения:

$$\begin{cases} \mathrm{He}_0(x) \equiv 1, \\ \mathrm{He}_1(x) \equiv x, \\ \mathrm{He}_{n+1}(x) = x \cdot \mathrm{He}_n(x) - n \cdot \mathrm{He}_{n-1}(x), \quad n > 0. \end{cases}$$

 ISSN 2541-8009

или с помощью формулы

$$\mathrm{He}_n(x) = \sum_{j=0}^{\left\lfloor \frac{n}{2} \right\rfloor} \left(-\frac{1}{2} \right)^j \frac{n!}{j!(n-2j)!} x^{n-2j},$$

где $\left\lfloor \frac{n}{2} \right\rfloor$ — целая часть числа $\frac{n}{2}$.

Поскольку $\langle \mathrm{He}_n, \mathrm{He}_n \rangle = n!$, ортонормированное семейство полиномов Эрмита имеет вид $\left\{ \frac{\mathrm{He}_n(x)}{\sqrt{n!}} \right\}$.

Двумерные полиномы Эрмита получают путем тензоризации одномерных полиномов. В табл. 2 приведены многочлены порядка не выше 3.

Таблица 2. **Двумерные многочлены Эрмита порядка не выше 3**

j	Мульти-индекс $\overline{a}$	Элемент базиса φ_j
0	[0, 0]	$\varphi_0 \equiv 1$
1	[1, 0]	$\varphi_1 \equiv x_1$
2	[0, 1]	$\varphi_2 \equiv x_2$
3	[2, 0]	$\varphi_3 \equiv \frac{x_1^2 - 1}{\sqrt{2}}$
4	[1, 1]	$\varphi_4 \equiv x_1 x_2$
5	[0, 2]	$\varphi_5 \equiv \frac{x_2^2 - 1}{\sqrt{2}}$
6	[3, 0]	$\varphi_6 \equiv \frac{x_1^3 - 3x_1}{\sqrt{6}}$
7	[2, 1]	$\varphi_7 \equiv \frac{x_1^2 - 1}{\sqrt{2}} x_2$
8	[1, 2]	$\varphi_8 \equiv x_1 \frac{x_2^2 - 1}{\sqrt{2}}$
9	[0, 3]	$\varphi_9 \equiv \frac{x_2^3 - 3x_2}{\sqrt{6}}$

Стохастический метод Галеркина. Метод Галеркина — это интрузивный метод приближенного решения дифференциального уравнения $Lu = f(\overline{x})$, где

L — непрерывный дифференциальный оператор, который может содержать как полные, так и частные производные любого порядка.

Приближенное решение данной краевой задачи $\overline{u}$ будем искать в следующем виде:

$$\tilde{u}(\overline{x}) = \sum_{j=1}^{n} c_j \varphi_j(\overline{x}), \tag{2}$$

где $\varphi_j(\overline{x})$ — это линейно-независимые функции, а c_j — неопределенные коэффициенты. При этом можно считать, что $\varphi_j(\overline{x})$ представляют собой первые n функций некоторой полной системы функций.

Пусть теперь $\tilde{u}(\overline{x})$ является точным решением дифференциального уравнения $Lu = f(\overline{x})$, т. е. пусть $L\tilde{u} = f(\overline{x})$. Это требование равносильно ортогональности невязки полученного решения $N(\overline{x}) = L\tilde{u} - f(\overline{x})$ ко всем функциям $\varphi_j(\overline{x})$ $\forall j \in \mathbb{N}$, так как система функций $\varphi_j(\overline{x})$ является полной. Однако поскольку можно оперировать только первыми n функциями $\varphi_j(\overline{x})$, можно удовлетворить лишь n условий ортогональности, в связи с чем точного решения в общем случае найти не получится, так как решить систему из бесконечного числа уравнений в общем случае невозможно. Запишем условия ортогональности для n уравнений:

$$\int_D N(\overline{x})\varphi_j(\overline{x})d\overline{x} = \int_D \left(L\left(\sum_{k=1}^{n} c_k \varphi_k(\overline{x}) \right) - f(\overline{x}) \right) \varphi_j(\overline{x})d\overline{x} = 0, \tag{3}$$

где $j = \overline{1,n}$.

Из системы уравнений (3) можно найти все коэффициенты c_j и, подставив их в выражение (2), получить приближенное решение $\tilde{u}(\overline{x})$ данного дифференциального уравнения [7].

Помимо решения дифференциальных уравнений метод Галеркина также можно использовать для нахождения параметров регрессионной модели полиномиального хаоса $Y = (\overline{X}) = \sum_{j=0}^{+\infty} y_j \varphi_j(\overline{X})$ при помощи детерминированных уравнений, определяющих поведение системы. При этом коэффициенты регрессии $\overline{a}$ считаются неизвестными, они находятся путем вычисления стохастических проекций Галеркина. Будем называть данную спецификацию метода Галеркина *стохастическим методом Галеркина* [8].

Приведем алгоритм стохастического метода Галеркина.

1. Записать разложение полиномиального хаоса для входных факторов, учитывая их совместное распределение.

 ISSN 2541-8009

2. Представить отклик модели в виде линейной комбинации базисных полиномов, которые бы удовлетворяли наложенным на систему граничным условиям.

3. Подставить разложение полиномиального хаоса и отклик модели в детерминированные уравнения.

4. Вычислить скалярное произведение левой и правой части уравнения, полученного в пункте 3, с каждым базисным полиномом φ_n и получить все необходимые коэффициенты разложения.

5. Рассчитать статистические характеристики решения, используя свойства коэффициентов ПХ.

Для одномерной нормально распределенной выборки коэффициенты a_j можно найти численно, используя общую формулу

$$a_j = \frac{1}{\sigma\sqrt{2\pi}\sqrt{j!}} \int_{-\infty}^{+\infty} Y(x)\,\mathrm{He}_j\left(\frac{x-\mu}{\sigma}\right)\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)dx,$$

где μ — выборочное среднее выборки; σ^2 — выборочная дисперсия выборки.

Для численного интегрирования данного выражения используем метод Симпсона. Отсортировав нормально распределенную выборку, получим неравномерную сетку, шаг которой определяется как $h_k = X_{k+1} - X_{k-1}$. Тогда введем обозначение

$$g_j(x) = Y(x)\frac{\mathrm{He}_j\left(\frac{x-\mu}{\sigma}\right)}{\sqrt{j!}}\frac{\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}{\sigma\sqrt{2\pi}}$$

и получим, что:

$$\begin{cases} a_j \approx \sum\limits_{k=1,2}^{n-2} \dfrac{h_k}{6}\Big(g_j(X_{k-1}) + 4g_j(X_k) + g_j(X_{k+1})\Big), \\ Y(x) \approx \sum\limits_{j=0}^{N-1} a_j \dfrac{\mathrm{He}_j\left(\frac{x-\mu}{\sigma}\right)}{\sqrt{j!}}. \end{cases} \tag{4}$$

Здесь суммирование в первой формуле выражения (4) проводится с шагом 2 [9], во второй формуле выражения (4) рассматривается усечение РПХ до первых N слагаемых.

Сравнение эффективности алгоритмов на C++ и Python на тестовых примерах. Для сравнения алгоритмов возьмем некоторые одномерные тестовые функции, аргумент которых распределен по нормальному закону с из-

вестными параметрами μ и σ. Тогда входными данными будет служить вектор X длины N, каждая компонента x_j которого распределена по нормальному закону с параметрами μ и σ соответственно. Возьмем выборку из 100 таких векторов и усредним полученные результаты по этой выборке.

Вычисления на C++ осуществлялись в среде разработки Visual Studio 2022 с использованием стандарта C++20. Вычисления на Python проводились в среде разработки Visual Studio Code, версия языка Python 3.12.6. Программный код запускался на машине со следующей конфигурацией: CPU Intel(R) Core(TM) i5-9600K 3.70 GHz, RAM DDR4 16 Gb, GPU NVIDIA GeForce RTX 2070, операционная система Windows 11 Pro 23H2. Статистика работы алгоритма представлена в табл. 3, 4.

Таблица 3. **Статистика работы алгоритма на Python**

№ п/п	f	N	μ	σ	Среднеквадратичная ошибка		t, мс
					ε	ε_0	
1	$\sin\left(\frac{\pi x}{6}\right)$	100 01	0	2	$4{,}8 \cdot 10^{-5}$	$4{,}8 \cdot 10^{-5}$	1 835,95
2	$\cos\left(\frac{\pi x}{6}\right)$	100 01	0	2	0,000 19	0,000 19	1 702,19
3	$x\sin\left(\frac{\pi x}{6}\right)$	100 01	6	3	0,218 79	0,014 58	2 001,78
4	$x\sin\left(\frac{\pi x}{6}\right)$	100 001	6	3	0,176 91	0,011 80	17 739,07
5	$(x+1)^2\cos\left(\frac{\pi x}{6}\right)$	10 001	0	2,5	1,869 54	0,033 61	1 799,91
6	$(x+1)^2\cos\left(\frac{\pi x}{6}\right)$	100 001	0	2,5	1,075 57	0,019 34	20 449,96
7	$\mathrm{th}\left(\frac{x}{2}\right)$	1 001	0	1	0,007 47	0,008 23	178,44
8	$\mathrm{th}\left(\frac{x}{2}\right)$	10001	0	1	0,000 18	0,000 20	1 830,32

 ISSN 2541-8009

Таблица 4. **Статистика работы алгоритма на C++**

№ п/п	f	N	μ	σ	Среднеквадратичная ошибка		t, мс
					ε	ε_0	
1	$\sin\left(\frac{\pi x}{6}\right)$	10 001	0	2	$4{,}9 \cdot 10^{-5}$	$4{,}9 \cdot 10^{-5}$	51,00
2	$\cos\left(\frac{\pi x}{6}\right)$	10 001	0	2	0,000 19	0,000 19	47,55
3	$x\sin\left(\frac{\pi x}{6}\right)$	10 001	6	3	0,317 05	0,022 63	47,89
4	$x\sin\left(\frac{\pi x}{6}\right)$	100 001	6	3	0,285 43	0,020 44	451,98
5	$(x+1)^2\cos\left(\frac{\pi x}{6}\right)$	10 001	0	2,5	1,862 47	0,033 48	46,77
6	$(x+1)^2\cos\left(\frac{\pi x}{6}\right)$	100 001	0	2,5	1,075 91	0,019 34	443,66
7	$\mathrm{th}\left(\frac{x}{2}\right)$	1 001	0	1	0,007 61	0,008 40	6,31
8	$\mathrm{th}\left(\frac{x}{2}\right)$	10 001	0	1	0,000 18	0,000 20	46,00

Приведем некоторые пояснения к результатам, представленным в табл. 3 и 4. Здесь ε — среднеквадратичная ошибка аппроксимации

$$\varepsilon = \frac{1}{n}\sum_{j=1}^{n}\left(f(x_j) - g(x_j)\right)^2,$$

где $g(x)$ — аппроксимирующая функция, точки x_j при подсчете ошибки выбирались из отрезка $\left[\mu - 3\sigma;\ \mu + 3\sigma\right]$ с шагом $h = 6\sigma/N$.

Относительная среднеквадратичная ошибка аппроксимации вычисляется по формуле

$$\varepsilon_0 = \frac{\varepsilon}{\max\limits_j\left(\left|f(x_j)\right|\right)},$$

t — время выполнения одной итерации в миллисекундах (для вычисления общего времени выполнения программы необходимо умножить этот пара-

метр на число итераций, которое, в данном случае, оставалось постоянным и равным 100).

Сравнив последние столбцы табл. 3 и 4, а также разницу во времени выполнения алгоритмов для решения задачи Коши, убеждаемся в том, что алгоритм, написанный на C++, работает гораздо быстрее. Причины этого кроятся в особенностях этих языков [10].

1. C++ является компилируемым языком, поэтому программа, написанная на C++, перед запуском компилируется в машинный код, который выполняется процессором напрямую. Python же является интерпретируемым языком, а значит, каждый раз при запуске программы интерпретатор должен «переводить» код, написанный на Python, в машинный код построчно, что сильно замедляет процесс выполнения программы.

2. В Python используется сборщик мусора (Garbage Collector), который автоматически управляет памятью: высвобождает ее при необходимости записать данные в некоторую область памяти (например, при создании списка) и освобождает ее при завершении работы с некоторой областью памяти. Хотя это и упрощает разработку, использование сборщика мусора негативно сказывается на производительности.

3. C++ — язык со статической типизацией. Типы переменных определяются на этапе компиляции и не меняются, что позволяет компилятору оптимизировать использование памяти и ускорить выполнение программы. Python же — язык с динамической типизацией. В Python тип переменной может меняться во время выполнения программы, из-за чего необходимо проводить дополнительные проверки типов во время выполнения программы, что, естественно, снижает производительность.

Уравнение колебаний со случайным коэффициентом затухания. Приме́ним реализованный алгоритм к решению задачи Коши. Рассмотрим модель линейного затухающего осциллятора:

$$\begin{cases} m\dfrac{d^2u}{dt^2} + c\dfrac{du}{dt} + ku = f\cos pt, \\ u(0) = u^0, \\ \dfrac{du}{dt}(0) = v^0, \end{cases} \tag{5}$$

где m — масса тела; c — коэффициент затухания колебаний; k — жесткость пружины; f — амплитуда вынуждающей силы; p — циклическая частота вынуждающей силы; u^0 — начальное положение тела; v^0 — начальная скорость тела. Примем также следующие ограничения на параметры системы, чтобы обеспечить отсутствие резонанса в системе:

 ISSN 2541-8009

$$\begin{cases} m > 0, \\ k > 0, \\ f > 0, \\ p > 0, \\ 0 < c < 2\sqrt{km}, \\ \omega = \dfrac{\sqrt{4km - c^2}}{2m} \neq p, \end{cases}$$

где ω — собственная частота колебаний системы, $\omega > 0$.

Как известно, общее решение дифференциального уравнения (5) записывается в виде

$$u(t) = \exp\left(-\frac{c}{2m}t\right)\left(A\cos\omega t + B\sin\omega t\right) + C_1\cos pt + C_2\sin pt,$$

где

$$\begin{cases} C_1 = \dfrac{f(k - mp^2)}{c^2p^2 + (k - mp^2)^2}, \\ C_2 = \dfrac{cfp}{c^2p^2 + (k - mp^2)^2}, \\ A = u^0 - C_1, \\ B = \dfrac{1}{\omega}\left(v^0 + \dfrac{c}{2m}A - C_2 p\right). \end{cases}$$

Будем рассматривать случай, когда коэффициент затухания колебаний измерен с нормально распределенной ошибкой, т. е. будем считать, что

$$c = \mu + \sigma\xi, \quad \xi \sim N(0,1).$$

Поскольку модель (5) является динамической, коэффициенты в разложении полиномиального хаоса являются скалярными функциями от времени:

$$Y = \sum_{j=0}^{+\infty} a_j(t)\varphi_j(\xi) = \sum_{j=0}^{+\infty} a_j(t)\frac{\mathrm{He}_j(\xi)}{\sqrt{j!}}.$$

Представим функцию $u(t)$ в виде РПХ:

$$u(t) = \sum_{j=0}^{+\infty} u_j(t)\varphi_j(\xi). \tag{6}$$

Подставим в $u(t)$ начальные условия:

$$u(0) = \sum_{j=0}^{N} u_j(0)\varphi_j(\xi) = u^0\varphi_0,$$

а значит,

$$u_n(0) = u^0\delta_{0n}, \quad n = \overline{0,N}.$$

Аналогично:

$$u'_N(0) = v^0\delta_{0n}, \quad n = \overline{0,N}.$$

Теперь подставим ряд (6) в исходное уравнение:

$$m\sum_{j=0}^{+\infty} u''_j\varphi_j + \left(\mu + \sigma\xi\right)\sum_{j=0}^{+\infty} u'_j\varphi_j + k\sum_{j=0}^{+\infty} u_j\varphi_j = f\cos pt. \tag{7}$$

Приведем уравнение (7) к системе уравнений, используя свойства полиномов Эрмита и умножив уравнение на базисные полиномы φ_n. В итоге, переходя от бесконечной суммы к усечению ряда до $(N+1)$-го слагаемого, получим:

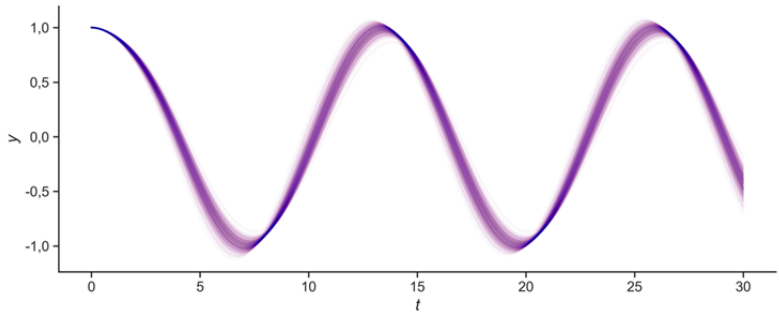
$$\begin{cases} mu''_0 + \mu u'_0 + \sigma u'_1 + ku_0 = f\cos pt, \\ mu''_n + \sigma\sqrt{n}u'_{n-1} + \mu u'_n + \sigma\sqrt{n+1}u'_{n+1} + ku_n = 0,\ n = \overline{1,N-1}, \\ mu''_N + \sigma\sqrt{N}u'_{N-1} + \mu u'_N + ku_N = 0, \\ u_n(0) = u^0\delta_{0n},\ n = \overline{0,N}, \\ u'_n(0) = v^0\delta_{0n},\ n = \overline{0,N}. \end{cases} \tag{8}$$

Для решения уравнения (8) выберем конкретные параметры системы:

$$\begin{cases} m = 1, \\ k = 1{,}16, \\ f = 0{,}9881, \\ p = 0{,}5, \\ u^0 = 1, \\ v^0 = 0. \end{cases}$$

Численно решая эту систему методом Рунге — Кутты третьего порядка, получаем результаты, представленные на рисунке.

 ISSN 2541-8009



Совмещенные траектории модели (*красный цвет*) и метамодели ПХ (*синий цвет*) при вариации коэффициента затухания колебаний

Для случайного параметра c была сгенерирована выборка из нормального распределения $c \sim N(\mu, \sigma)$ с параметрами $\mu = 0,8,\ \sigma = 0,2$ длиной в 1000 элементов. На рисунке отображены решения детерминированных моделей при данных параметрах c (траектории модели) и аппроксимации методом стохастических проекций Галеркина при тех же значениях параметра (траектории метамодели ПХ). Метод показал достаточно высокую точность: относительная среднеквадратичная ошибка метода составила $1{,}1 \cdot 10^{-5}$, что отлично видно на рисунке: красный и синий график неразличимы, из-за чего итоговый график становится фиолетового цвета. При этом алгоритм на языке Python выполнил задачу за 38,16 с, а аналогичный алгоритм на C++ — за 0,8279 с (время выполнения усреднялось по выборке длины 100, общее время выполнения программы на языке Python составило 3816 с, на C++ — 827,9 с).

Метод Галеркина показал очень высокую точность при решении задачи о линейном затухающем осцилляторе при небольшом количестве вычислительных затрат: программа на C++ отработала за относительно небольшое время. Эффективность метода связана прежде всего с тем, что он является интрузивным, т. е. учитывает особенности конкретной задачи и вид распределения входных параметров.

Заключение. В ходе данной работы был реализован стохастический метод Галеркина для аппроксимации функций, аргументы которой являются случайными величинами с известным законом распределения. Было проведено тестирование данного метода на некоторых одномерных функциях, также с его помощью была решена задача Коши, на которой метод показал очень высокую эффективность. Также было проведено сравнение результатов работы алгоритма на языке C++ и на языке Python. Язык C++ показал гораздо бóльшую эффективность, поскольку время выполнения алгоритма на нем было существенно ниже, нежели у аналогичного алгоритма на языке Python.

Литература

[1] Sudret B., Mai C. Computing derivative-based global sensitivity measures using polynomial chaos expansions. *arXiv:1405.5740*, 2015. https://doi.org/10.48550/arXiv.1405.5740

[2] Kaintura A., Dhaene T., Spina D. Review of polynomial chaos-based methods for uncertainty quantification in modern integrated circuits. *Electronics*, 2018, vol. 7 (3), art. no. 30. https://doi.org/10.3390/electronics7030030

[3] Соболь И.М. *Численные методы Монте-Карло*. Москва, Наука, 1967.

[4] Пупков К.А. Вероятностная неопределенность в стохастических технических системах управления. *Инженерный журнал: наука и инновации*, 2013, № 10 (22). https://doi.org/10.18698/2308-6033-2013-10-1096

[5] Parekh J., Verstappen R. Intrusive polynomial chaos for CFD using OpenFOAM. *Proc. of Workshop on Frontiers of Uncertainty Quantification in Fluid Dynamics*, Springer, 2020, vol. 12143.

[6] Смирнов В.И. *Курс высшей математики.* Москва, Наука, 1959.

[7] Канторович Л.В., Крылов В.И. *Приближенные методы высшего анализа*. Ленинград-Москва, Гос. изд-во физ.-мат. лит-ры, 1962.

[8] Neckel T. *Polynomial Chaos Approximation 2: The stochastic Galerkin approach. Algorithms for Uncertainty Quantification*. Lecture 7, Technische Universität München, 2018.

[9] Smith J. *Simpson's Rule Revisited*. URL: https://arxiv.org/abs/2011.13559 (дата обращения 11.11.2024).

[10] Rabah S., Li J., Liu M. *Comparative Studies of 10 Programming Languages within 10 Diverse Criteria*. URL: https://arxiv.org/abs/1009.0305 (дата обращения 11.11.2024).

Поступила в редакцию 11.01.2025

Хаписов Малик Хаписович — студент кафедры «Вычислительная математика и математическая физика», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Научный руководитель — Облакова Татьяна Васильевна, кандидат физико-математических наук, доцент кафедры «Вычислительная математика и математическая физика», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Ссылку на эту статью просим оформлять следующим образом:
Хаписов М.Х. Реализация стохастического метода Галеркина к аппроксимированию функций, зависящих от случайных параметров. *Политехнический молодежный журнал*, 2025, № 3 (98). URL: https://ptsj.bmstu.ru/catalog/math/compmath/1050.html

 ISSN 2541-8009

INTRODUCING THE STOCHASTIC GALERKIN METHOD IN FUNCTIONS APPROXIMATION DEPENDING ON THE RANDOM PARAMETERS

M.Kh. Khapisov khapisovmkh@student.bmstu.ru

Bauman Moscow State Technical University, Moscow, Russian Federation

The paper considers possibilities of applying the polynomial chaos concept in studying dynamic systems with the random parameters. It applies the intrusive stochastic Galerkin projection method in computing the coefficients to take into account in maximum the type of the linear differential operator describing the system. The method is implemented in the C++ and Python programming languages, numerical integration is performed with the Simpson method on the non-uniform mesh. The implemented algorithms efficiency is compared on the test functions. The method was applied to solving the problem of a linear damped oscillator, which was used to assess the approximation accuracy.

Keywords: Hermite polynomials, polynomial chaos decomposition, stochastic Galerkin method, Simpson method, Cauchy problem, C++, Python

Received 11.01.2025

Khapisov M.Kh. — Student, Department of Computational Mathematics and Mathematical Physics, Bauman Moscow State Technical University, Moscow, Russian Federation.

Scientific advisor — Oblakova T.V., Ph. D. (Phys.-Math.), Associate Professor, Department of Computational Mathematics and Mathematical Physics, Bauman Moscow State Technical University, Moscow, Russian Federation.

Please cite this article in English as:
Khapisov M.Kh. Introducing the stochastic Galerkin method in functions approximation depending on the random parameters. *Politekhnicheskiy molodezhnyy zhurnal*, 2025, no. 3 (98). URL: https://ptsj.bmstu.ru/catalog/math/compmath/1050.html

