|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | «Фундаментальные науки» |
| КАФЕДРА | «Вычислительная математика и математическая физика» |

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

|  |
| --- |
| ***«Применение стохастического метода Галёркина к*** |
| ***анализу регрессионных моделей»*** |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| Студент ФН11-82Б |  | М.Х. Хаписов |
|  | (Подпись, дата) | (И.О.Фамилия) |
| Руководитель ВКР |  | Т.В. Облакова |
|  | (Подпись, дата) | (И.О.Фамилия) |
| Нормоконтролер |  | С.С. Кудрявцева |
|  | (Подпись, дата) | (И.О.Фамилия) |

*2024 г.*

# РЕФЕРАТ

Расчетно-пояснительная записка 117 с., 21 рис., 3 табл., 18 источников.

ОРТОНОРМИРОВАННЫЕ ПОЛИНОМЫ, ПОЛИНОМИАЛЬНЫЙ ХАОС, ПОЛИНОМЫ ЭРМИТА, СТОХАСТИЧЕСКИЙ МЕТОД ГАЛЁРКИНА, МЕТОД НАИМЕНЬШИХ КВАДРАТОВ, ДЕТЕРМИНИРОВАННЫЕ МОДЕЛИ, МОДЕЛЬ ШЛЁГЛЯ, УРАВНЕНИЕ ФОККЕРА-ПЛАНКА, ПЛОТНОСТЬ ВЕРОЯТНОСТИ ЭВОЛЮЦИИ СИСТЕМЫ, ФУНКЦИЯ РАСПРЕДЕЛЕНИЯ ЭВОЛЮЦИИ СИСТЕМЫ

Объектом исследования являются математические модели, описывающие изменения состояния системы под действием различных факторов стохастической природы.

Цель работы — проверка эффективности стохастического метода Галёркина, а также определение границ его применимости.

Поставленная цель достигается путём изучения и проработки теоретических основ, разработки программного кода, реализующего метод стохастической проекции для подбора моделей РПХ, описывающих реальные физические процессы, в том числе при отсутствии аналитического решения.

# СОДЕРЖАНИЕ

[ОБОЗНАЧЕНИЯ 4](#_Toc169348367)

[ВВЕДЕНИЕ 5](#_Toc169348368)

[1 Теоретическая часть 6](#_Toc169348369)

[1.1 Методы Монте-Карло 6](#_Toc169348370)

[1.2 Полиномиальный хаос 6](#_Toc169348371)

[1.3 Семейства ортогональных полиномов 11](#_Toc169348372)

[1.4 Полиномы Эрмита 16](#_Toc169348373)

[1.5 Общая линейная регрессионная модель 17](#_Toc169348374)

[1.6 Линейная регрессионная модель полиномиального хаоса 21](#_Toc169348375)

[1.7 Стохастический метод Галёркина 22](#_Toc169348376)

[2 Практическая часть 27](#_Toc169348377)

[2.1 Сравнение стохастического метода Галёркина и метода наименьших квадратов 27](#_Toc169348378)

[2.2 Применение стохастического метода Галёркина к нахождению плотности эволюции системы 29](#_Toc169348379)

[2.2.1 Схемы взаимодействий и детерминированные модели 29](#_Toc169348380)

[2.2.2 Модель Шлёгля как простейшая одномерная бистабильная система 31](#_Toc169348381)

[2.2.3 Уравнения Колмогорова 35](#_Toc169348382)

[2.2.4 Уравнение Фоккера-Планка и нахождение плотности вероятности эволюции системы 37](#_Toc169348383)

[2.2.5 Система с бистабильным поведением 41](#_Toc169348384)

[2.2.6 Система с моностабильным поведением 48](#_Toc169348385)

[2.3 Задача о линейном затухающем осцилляторе 56](#_Toc169348386)

[ЗАКЛЮЧЕНИЕ 66](#_Toc169348387)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 68](#_Toc169348388)

[ПРИЛОЖЕНИЕ А Программный код на языке Python 70](#_Toc169348389)

[ПРИЛОЖЕНИЕ Б Графическая часть ВКР 109](#_Toc169348390)

# ОБОЗНАЧЕНИЯ

 – вектор входных факторов

 – модель

 – выходные данные

 – отклик модели

 – ортонормированный полиномиальный базис

 – коэффициенты разложения отклика модели  по ортонормированному базису

 – компоненты случайного вектора 

 – маргинальная плотность распределения компоненты 

 – семейство ортонормированных полиномов степени , определённое для каждой компоненты 

 – семейство ортонормированных многомерных полиномов, определённое для всего входного вектора 

 – усечение ряда

 – разложение полиномиального хаоса

 – базисные функции регрессионной модели

 – базисные коэффициенты регрессионной модели

 – вектор откликов

 – вектор базисных коэффициентов регрессионной модели

 – матрица базисных функций в регрессионной модели

 – вектор ошибок

****** – вектор базисных функций разложения полиномиального хаоса

 – оценка коэффициентов разложения полиномиального хаоса

 – многочлен Эрмита степени 

# ВВЕДЕНИЕ

При решении широкого класса задач одним из основных методов в настоящее время является компьютерное моделирование, которое, несмотря на своё бурное развитие в настоящее время, 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 неопределённостью входных факторов.

В связи с этим всё чаще сегодня прибегают к использованию стохастических методов. Так, например, широко используется подход на основе, так называемого, черного ящика, преобразующего по некоторому неизвестному и содержащему элементы случайности правилу вектор входных переменных  в отклик системы Сюда можно отнести как классический метод линейной регрессии, так и обретающий сегодня всё большую популярность метод разложения полиномиального хаоса (РПХ). Этот метод рассматривает входные данные  как реализацию некоторой случайной величины, что позволяет находить отклик в виде ряда по ортогональной системе полиномов, определяемой законом распределения вектора . При вычислении коэффициентов этого разложения применяют разнообразные интрузивные и неинтрузивные методы.

Целью данной работы является изучение эффективности применения стохастического метода Галёркина.

# Теоретическая часть

## **Методы Монте-Карло**

У методов Монте-Карло нет конкретного, общепринятого определения. В данной работе будем считать, что методы Монте-Карло – это группа численных методов решения математических задач при помощи моделирования случайных величин. Суть метода заключается в описании исследуемого процесса математической моделью с использованием генератора случайных чисел. Таким образом, для реализации метода необходимо сгенерировать случайную выборку, соответствующую закону распределения случайной величины, служащей нам входными данными, и оценить вероятность успеха:

|  |  |
| --- | --- |
| , | (1.1) |
|  |  |

где – число успешных испытаний, а  – общее число испытаний [1].

Для использования метода Монте-Карло необходимо построить цифровую модель системы и смоделировать большую выборку реализаций, по которой и будет определяться эффективность работы системы. В случае же, когда многократное моделирование реализаций невозможно (например, в силу того, что моделирование системы требует большого количества ресурсов вычислительной машины, а также большого количества времени), прибегают ко всевозможным альтернативам, например, к аппроксимации стохастических динамических систем функциональными рядами. [2]

## **Полиномиальный хаос**

Часто решение задачи прогнозирования какой-либо системы сводится к работе с моделью, принимающей большое количество входных данных, таких как граничные и начальные условия, свойства исследуемого объекта и т.д. При этом точные значения этих данных, как правило, неизвестны, так как вычисляются эти данные с помощью зашумлённых измерений. Существуют, тем не менее, вероятностные подходы, в которых данные параметры модели рассматриваются как случайные величины, функцию распределения и моменты которых можно было бы вычислить. [3]

Одним из таких методов является метод полиномиального хаоса. Он заключается в представлении случайной величины как полиномиальной функции от других случайных величин. Этот метод очень удобен тем, что позволяет использовать случайные величины, распределённые по разным законам, и вычислять их моменты.

Рассмотрим, в качестве примера, численную модель , в которой входной вектор  состоит из  независимых случайных величин , а  – случайный отклик модели  (выходные данные). В предположении, что отклик  имеет конечную дисперсию, он может быть записан, как:

|  |  |
| --- | --- |
|  | (1.2) |

где  образуют базис в вероятностном пространстве, а  – коэффициенты разложения случайного отклика  по этому базису.

Если функции  являются полиномами от случайных величин, то ряд (1.2) называется разложением полиномиального хаоса. [3]

Для расчёта коэффициентов  применяются как интрузивные, так и неинтрузивные методы. Разница состоит в том, что в неинтрузивных методах при расчёте коэффициентов используют только фундаментальную детерминированную модель, не изменяя её (в интрузивных, соответственно, наоборот). Самым распространённым представителем неинтрузивных методов служит метод Монте-Карло. Методы интрузивного типа обычно обладают высокой скоростью сходимости в среднем квадратичном, в то же время неинтрузивные методы обычно более наглядны. Рассматриваемый в этой работе метод Галёркина как раз и является интрузивным методом. [4]

Рассмотрим разложение полиномиального хаоса для случая, когда компоненты  случайного вектора независимы и имеют конечные моменты любого порядка. Обозначим маргинальную плотность распределения этих компонент как . Тогда, поскольку компоненты случайного вектора независимы, получаем представление совместной плотности распределения в виде:

|  |  |
| --- | --- |
|  | (1.3) |

Теперь для каждой компоненты  определим соответствующее ей семейство ортонормированных полиномов , где  – степень многочлена (примем ). При этом по свойству ортонормированности:

|  |  |
| --- | --- |
|  | (1.4) |

где – символ Кронекера,  – носитель случайной величины .

Теперь многомерные полиномы  можно построить с помощью тензоризации одномерных полиномов:

|  |  |
| --- | --- |
| , | (1.5) |

где  – мульти-индекс.

Докажем ортонормированность полиномов :

|  |  |
| --- | --- |
| . | (1.6) |

Таким образом, ** – ортонормированный базис, а значит любой стохастический процесс  можно разложить по базису **, записав равенство Парсеваля для этого процесса: [5]

|  |  |
| --- | --- |
| . | (1.7) |

Решая практические задачи, вычислять сумму бесконечного ряда, естественно, невозможно, поэтому приходится прибегать к усечению ряда. [6]

Зададим стандартную схему усечения ряда как:

|  |  |
| --- | --- |
| , | (1.8) |

где .

Множество  состоит из многочленов, порядок которых не превосходит , поэтому мощность этого множества .

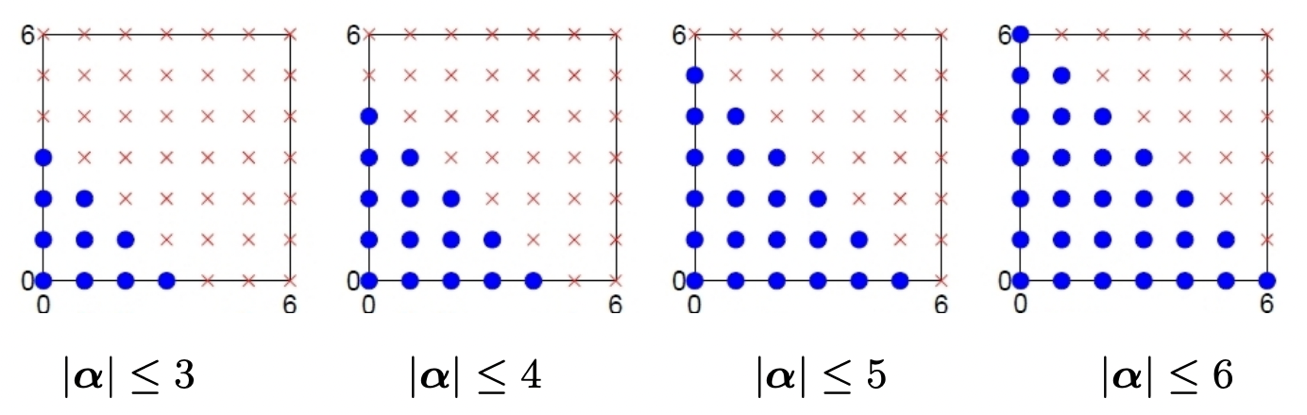


Рисунок 1 – Графическое представление степеней многочленов, которые (степени) принадлежат множеству при 

С помощью такого усечения можно приближать разложение полиномиального хаоса, являющееся, вообще говоря, бесконечным рядом, конечными суммами, причём с любой точностью. Более того, при  усечённое разложение сходится к полному в среднем квадратичном:

|  |  |
| --- | --- |
| , | (1.9) |

где  – полное разложение полиномиального хаоса

Одно из главных преимуществ метода полиномиального хаоса состоит в относительной простоте вычисления моментов:

|  |  |
| --- | --- |
| , | (1.10) |

так как ,  в силу ортонормированности полиномов 

Теперь запишем выражение для второго момента:

|  |  |
| --- | --- |
| . | (1.11) |

Раскроем квадрат и занесём математическое ожидание в сумму:

|  |  |
| --- | --- |
| . | (1.12) |

В силу ортонормированности полиномов  . Подставим это в выражение (1.12):

|  |  |
| --- | --- |
| . | (1.13) |

Таким образом, получаем выражения для моментов:

|  |  |
| --- | --- |
|  | (1.14) |

## **1.3 Семейства ортогональных полиномов**

Для наиболее распространённых распределений уже известны их ортогональные многочлены.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Распре-деление** | **Плотность распределения** | **Ортог. полиномы** | **Ортонормированный базис** |
| Равно-мерное |  | Лежандра |  |
| Нор-мальное |  | Эрмита |  |
| Гамма |  | Лагерра |  |
| Бета |  | Якоби |  |

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

Представим, например, логнормальную случайную величину  как функцию от нормально распределённой случайной величины  с параметрами , .

Поскольку  – логнормальная случайная величина, её можно представить в виде , где , .

Запишем полное разложение полиномиального хаоса для :

|  |  |
| --- | --- |
| *.* | (1.15) |

Заметим теперь, что:

|  |  |
| --- | --- |
| . | (1.16) |

И для  получаем:

|  |  |
| --- | --- |
| . | (1.17) |

Запишем выражение для  в интегральном виде:

|  |  |
| --- | --- |
| . | (1.18) |

где – ортонормированный многочлен Эрмита

Таким образом, мы получили интегральное выражение для :

|  |  |
| --- | --- |
| . | (1.19) |

Докажем последнее равенство в (1.19) индуктивно. Заметим, что для , то есть утверждение (1.19) верно для этого случая.

Пусть теперь утверждение выполнено для некоторого , и проверим его истинность для . Для начала, выпишем определение многочлена Эрмита:

|  |  |
| --- | --- |
| . | (1.20) |

Тогда запишется как:

|  |  |
| --- | --- |
| . | (1.21) |

Явно выпишем многочлен Эрмита в (1.20), используя его определение в (1.19):

|  |  |
| --- | --- |
| . | (1.22) |

Проинтегрируем это выражение по частям. Пусть , тогда  и . Таким образом, получаем:

|  |  |
| --- | --- |
| . | (1.23) |

Пусть , где  – многочлен степени . Тогда (1.23) можно представить в следующем виде:

|  |  |
| --- | --- |
| , | (1.24) |

где , так как .

Подставляя теперь (1.24) в (1.22), получаем рекуррентное уравнение для :

|  |  |
| --- | --- |
|  | (1.25) |

Так как утверждение (1.19) выполнено для , получаем выражение для 

|  |  |
| --- | --- |
| . | (1.26) |

Подставив его в (1.25), получаем выражение для , которое мы и стремились доказать:

|  |  |
| --- | --- |
| . | (1.27) |

Таким образом, утверждение (1.19) доказано.

Теперь, наконец, выпишем полное разложение полиномиального хаоса для :

|  |  |
| --- | --- |
|  | (1.28) |

## **1.4 Полиномы Эрмита**

Полиномы Эрмита – это семейство ортогональных многочленов вида (1.20), соответствующих нормальному распределению. Их удобно вычислять с помощью следующего рекуррентного соотношения:

|  |  |
| --- | --- |
|  | (1.29) |

Так как , ортонормированное семейство полиномов Эрмита имеет вид . Зададим стандартную схему усечения и запишем для неё соответствующие ортонормированные многочлены Эрмита

Таблица 2 – Двумерные многочлены Эрмита порядка не выше 3

|  |  |  |
| --- | --- | --- |
|  | Мульти-индекс | Элемент базиса |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## **1.5 Общая линейная регрессионная модель**

Рассмотрим двумерную выборку случайных переменных : , где  – векторы неслучайных факторов (входных данных),  – отклики модели, то есть выходные данные, измеренные с ошибкой .

Определим функцию регрессии  на  как условное математическое ожидание .

Приведём некоторые свойства функции регрессии:

1.  – свойство несмещённости ошибок;
2. ** , то есть регрессия – лучшее приближение  в среднем квадратичном.

Получаем модель , где выполнены следующие условия на ошибки :

|  |  |
| --- | --- |
|  | (1.30) |

В общем случае найти функцию  невозможно, но, если известен вид этой функции, её коэффициенты можно найти методом наименьших квадратов.

Рассмотрим линейную регрессионную модель с откликом:

|  |  |
| --- | --- |
| , | (1.31) |

где  – размерность пространства,  – базисные функции,  – параметры регрессии,  – ошибка.

Запишем (1.31) в матричном виде:

|  |  |
| --- | --- |
| , | (1.32) |

где – вектор откликов,  – вектор параметров регрессии,  – вектор ошибок,  – матрица базисных функций.

При этом, в силу неслучайности факторов, . Теперь докажем следующую теорему

Теорема 1. Если выполнены условия (1.30) и при этом , то  – оценка наименьших квадратов параметра , причём  и оптимальна в классе линейных несмещённых оценок параметра 

Доказательство.

Для нахождения оценки наименьших квадратов параметра необходимо минимизировать сумму квадратов ошибок:

|  |  |
| --- | --- |
| . | (1.33) |

Продифференцируем это выражение по каждому :

|  |  |
| --- | --- |
| . | (1.34) |

Запишем это равенство в матричном виде:

|  |  |
| --- | --- |
| . | (1.35) |

Выразив отсюда , получаем оценку :

|  |  |
| --- | --- |
| , | (1.36) |

где  – неслучайная матрица, поэтому

|  |  |
| --- | --- |
| , | (1.37) |

то есть оценка  – несмещённая.

Докажем теперь оптимальность этой оценки. Пусть  – некоторая несмещённая оценка параметра , где  – неслучайная матрица. Тогда  – единичная матрица.

Запишем дисперсионную матрицу :

|  |  |
| --- | --- |
|  | (1.38) |

Пусть , тогда для произвольного:

|  |  |
| --- | --- |
| , | (1.39) |

причём

|  |  |
| --- | --- |
|  | (1.40) |

Из (1.40) следует, что , то есть для минимизации следа матрицы  необходимо, чтобы матрица  являлась оценкой наименьших квадратов параметра , а значит эта оценка оптимальна.

## **1.6 Линейная регрессионная модель полиномиального хаоса**

Запишем усечённое разложение полиномиального хаоса в векторной форме:

|  |  |
| --- | --- |
| . | (1.41) |

При этом вектор коэффициентов  можно найти, используя неинтрузивные подходы, например, метод наименьших квадратов. Для этого необходимо найти оценку наименьших квадратов:

|  |  |
| --- | --- |
| . | (1.42) |

Запишем необходимое условие минимума для этой оценки:

|  |  |
| --- | --- |
| . | (1.43) |

Из (1.43) получаем следующее уравнение:

|  |  |
| --- | --- |
| , | (1.44) |

так как базис  ортонормирован, – единичная матрица.

Запишем аналогичное уравнение для двумерной выборки :

|  |  |
| --- | --- |
| , | (1.45) |

где  – отклик модели. [7]

Причём оценка (1.45) будет являться несмещённой и оптимальной в классе линейных несмещённых оценок согласно теореме 1.

## **1.7 Стохастический метод Галёркина**

Метод Галёркина – это интрузивный метод приближенного решения краевой задачи , где  – непрерывный дифференциальный оператор, который может содержать как полные, так и частные производные любого порядка.

Приближенное решение данной краевой задачи  ищется в следующем виде:

|  |  |
| --- | --- |
| , | (1.46) |

где функции  – это линейно-независимые функции, удовлетворяющие граничным условиям, наложенным на систему, а  –неопределённые коэффициенты. При этом можно считать, что  представляют собой первые  функций некоторой полной системы функций.

Пусть теперь  является точным решением дифференциального уравнения , то есть пусть . Это требование равносильно ортогональности невязки полученного решения  ко всем функциям , так как система функций  является полной. Однако, так как мы можем оперировать только первыми  функциями , мы можем удовлетворить лишь  условий ортогональности, в связи с чем точного решения, в общем случае, мы получить не можем, так как решить систему из бесконечного числа уравнений в общем случае невозможно. Запишем условия ортогональности для  уравнений:

|  |  |
| --- | --- |
| , | (1.47) |

где .

Из системы уравнений (1.47) можно получить все коэффициенты  и, подставив их в выражение (1.46), получить приближенное решение  данной краевой задачи. [8]

Помимо решения дифференциальных уравнений, метод Галёркина также может быть использован для нахождения параметров регрессионной модели полиномиального хаоса при помощи детерминированных уравнений, определяющих поведение системы. При этом коэффициенты регрессии считаются неизвестными, они находятся путём вычисления стохастических проекций Галёркина. Будем называть данную спецификацию метода Галёркина *стохастическим методом Галёркина*. [9]

Запишем алгоритм стохастического метода Галёркина:

1. Записать разложение полиномиального хаоса для входных факторов, учитывая их совместное распределение
2. Записать 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  и получить все необходимые коэффициенты разложения
5. Рассчитать статистические характеристики решения, используя свойства коэффициентов ПХ

Так, для одномерной нормально распределённой выборки коэффициенты  можно найти численно, используя общую формулу:

|  |  |
| --- | --- |
| , | (1.48) |

где  – выборочное среднее выборки,  – выборочная дисперсия выборки.

Для численного интегрирования данного выражения используем метод трапеций. Отсортировав нормально распределённую выборку, получим неравномерную сетку, шаг которой определяется как . Тогда введём обозначение:

|  |  |
| --- | --- |
| , | (1.49) |

и получим, что:

|  |  |
| --- | --- |
|  | (1.50) |

Для двумерной нормально распределённой выборки коэффициенты ищутся аналогичным образом:

|  |  |
| --- | --- |
|  | (1.51) |

Теперь аналогично одномерному случаю введём шаг на двумерной неравномерной сетке и вычислим двойной интеграл в формуле для  численно:

|  |  |
| --- | --- |
|  | (1.52) |

Проведём сравнение стохастического метода Галёркина со спектральными методами анализа регрессионных моделей (pseudo-spectral approach). Метод Галёркина – это интрузивный метод, зависящий от внутреннего устройства модели. Коэффициенты в разложении полиномиального хаоса вычисляются путём вычисления скалярного произведения отклика модели с соответствующим базисным полиномом, вследствие чего не возникает дополнительных вычислительных ошибок. Спектральные же методы являются неинтрузивными: модель воспринимается как чёрный ящик, внутреннее устройство которого не влияет на результат аппроксимации. Коэффициенты разложения считаются численно, с помощью, например, метода наименьших квадратов. Таким образом, спектральные методы оказываются гораздо более простыми в использовании, так как являются неинтрузивными и не зависят от используемой модели. В то же время в методе стохастический проекций Галёркина коэффициенты вычисляются из решения СЛАУ или системы дифференциальных уравнений, что гарантирует отсутствие квадратурных ошибок при подсчёте коэффициентов. Таким образом, стохастический метод Галёркина является более точным методом. [9]

# 2 Практическая часть

## **2.1 Сравнение стохастического метода Галёркина и метода наименьших квадратов**

Пусть вектор входных данных  одномерен и распределён по нормальному закону с параметрами . Длину  вектора входных данных  возьмём равной 800.

Поскольку входные данные распределены по нормальному закону, в качестве базисных полиномов в методе Галёркина были взяты ортонормированные полиномы Эрмита:

|  |  |
| --- | --- |
| , | (2.1) |

где  – целая часть числа .

Для метода наименьших квадратов в качестве базисных функций были выбраны полиномы Колмогорова-Габора:

|  |  |
| --- | --- |
|  | (2.2) |

Приведём таблицу сравнения среднеквадратичных погрешностей разных методов (обозначим за  среднеквадратичную погрешность стохастического метода Галёркина и за  – среднеквадратичную погрешность метода наименьших квадратов) и количество слагаемых в разложении ряда (аналогично  – степень многочлена, представляющего собой усечение ряда полиномов Эрмита, ** – ряда полиномов Колмогорова-Габора) в зависимости от аппроксимируемой функции , среднеквадратичного отклонения входных данных .

Таблица 3 – Cравнение среднеквадратичных погрешностей стохастического метода Галёркина и метода наименьших квадратов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № |  |  |  |  |  |  |
| 1 |  | 8 | 9 | 9 |  |  |
| 2 |  | 10 | 9 | 9 | 0.0686 | 0.0111 |
| 3 |  | 1.5 | 9 | 9 | 0.309 | 0.28 |
| 4 |  | 2 | 9 | 9 | 0.592 | 1.13 |
| 5 |  | 2 | 9 | 9 | 1.30 | 1.55 |
| 6 |  | 3 | 9 | 9 | 1.59 | 1.66 |
| 7 |  | 1 | 11 | 11 | 1.41 | 1.03 |
| 8 |  | 8 | 17 | 17 | 6.00 | 11.2 |
| 9 |  | 0.5 | 11 | 11 | 0.663 | 0.510 |
| 10 |  | 1 | 11 | 11 | 1.377 | 1.562 |

Здесь  – одномерная функция Ackley,  – одномерная функция Растригина [10].

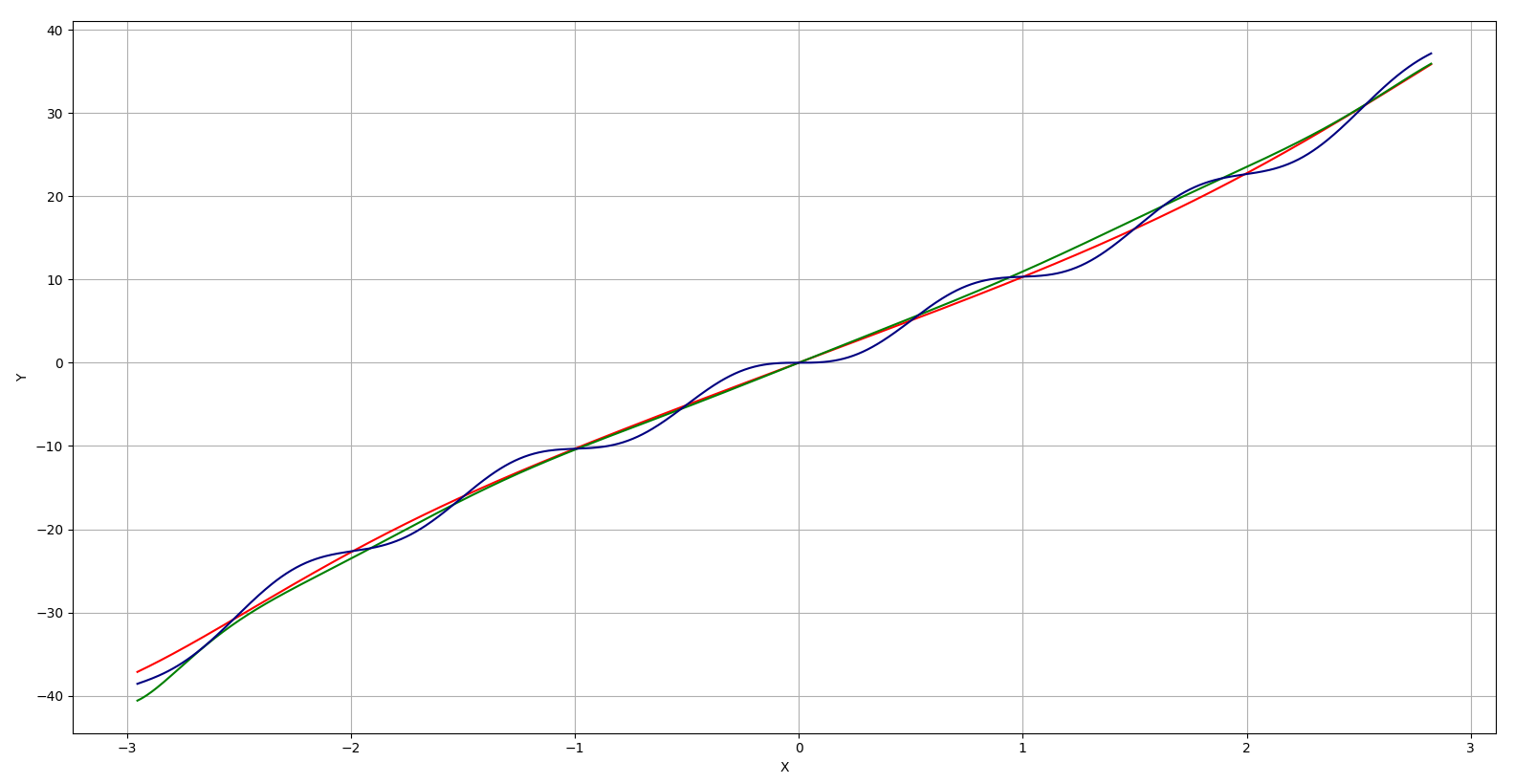


Рисунок 2 – графики аппроксимируемой функции (синий цвет), аппроксимации методом Галёркина (красный цвет) и метода наименьших квадратов (зелёный цвет) для случая, представленного 10-ой строкой таблицы 3

Таким образом, хоть и при малом среднеквадратичном отклонении входных данных  метод наименьших квадратов и является гораздо более точным, при повышении среднеквадратичного отклонения погрешность метода наименьших квадратов очень сильно возрастает, в то время как погрешность метода Галёркина растёт гораздо медленнее.

## **Применение стохастического метода Галёркина к нахождению плотности эволюции системы**

### **2.2.1 Схемы взаимодействий и детерминированные модели**

В различных областях естествознания и техники системы с взаимодействиями и превращениями составляющих их элементов задаются схемами взаимодействий. Например, в химической кинетике для мономолекулярной реакции используется запись . При детерминированном подходе реакцию описывают количеством компонента  и количеством  компонента  в момент времени  [11]. Полагают справедливым феноменологический закон ( – константа скорости реакции): [12]

|  |  |
| --- | --- |
|  | (2.3) |

А для бимолекулярной реакции  вводят обозначения ,, – количества реагентов типа ,, соответственно – и полагают справедливым закон действующих масс: [12]

|  |  |
| --- | --- |
|  | (2.4) |

Схемы взаимодействий также широко применяются в теории надёжности. Например, выход из строя элементов технических систем записывается как  [13].

Общая схема взаимодействий, в которой участвуют элементы типов  имеет вид:

|  |  |
| --- | --- |
| , | (2.5) |

где  – целые неотрицательные числа.

При детерминированном подходе к рассмотрению схемы (2.5) вводят количество  элементов типа , при этом функции  удовлетворяют следующей системе дифференциальных уравнений, которые называются кинетическими:

|  |  |
| --- | --- |
| , | (2.6) |

с начальными условиями  [11].

При этом вид функций  определяют по схеме (2.5) исходя из законов формальной кинетики [12]. В прикладных задачах функции  являются, как правило, полиномами степени не выше третьей [11].

### **2.2.2 Модель Шлёгля как простейшая одномерная бистабильная система**

Рассмотрим в качестве реальной системы упрощённую версию модели Шлёгля, поскольку она считается простейшей возможной одномерной бистабильной системой. Она описывается следующей схемой взаимодействий:

|  |  |
| --- | --- |
|  | (2.7) |

Эта система содержит только один тип частиц; каждое взаимодействие происходит с заданной интенсивностью . Значение  соответствует интенсивности появление новой частицы ,  – гибель частицы, соответствует интенсивности преобразования 2 частиц  в 3 частицы того же типа, аналогично соответствует интенсивности преобразования 3 частиц в 2 частицы того же типа.

При моделировании стохастической модели для схемы взаимодействий (2.7) реализации случайного процесса  на ЭВМ используется метод Монте-Карло [14]. Суть метода заключается в многократном использовании генератора случайных чисел для описания математической модели. Для процесса  задаются интенсивности , начальное количество частиц  и промежуток времени .

Проанализируем теперь детерминированную модель для модели Шлёгля. Уравнение детерминированной модели выглядит следующим образом:

|  |  |
| --- | --- |
| , | (2.8) |

где  – количество частиц типа  [11].

Примем новые обозначения  и найдём корни многочлена :

|  |  |
| --- | --- |
| . | (2.9) |

Пусть . Тогда, после раскрытия скобок и приведения подобных слагаемых, получим:

|  |  |
| --- | --- |
| . | (2.10) |

Примем следующие обозначения:

|  |  |
| --- | --- |
|  | (2.11) |

Тогда, по формуле Кардано, получаем, что корни многочлена (2.9) равны

|  |  |
| --- | --- |
|  | (2.12) |

Стационарное решение дифференциального уравнения (2.8) реализуется при . Заметим, что все эти корни действительны и различны тогда и только тогда, когда найденный в (2.11) . Решая это неравенство, получаем соответствующее неравенство для дискриминанта кубического многочлена:

|  |  |
| --- | --- |
|  | (2.13) |

При  система бистабильна с одной нестабильной фиксированной точкой между двумя стабильными фиксированными точками, при  система моностабильна. Переход от моностабильности к бистабильности происходит через бифуркацию седлового узла.

Детерминированный подход хорошо описывает поведение системы (2.7) при большом количестве молекул, что даёт возможность пренебречь стохастическими флуктуациями [15].

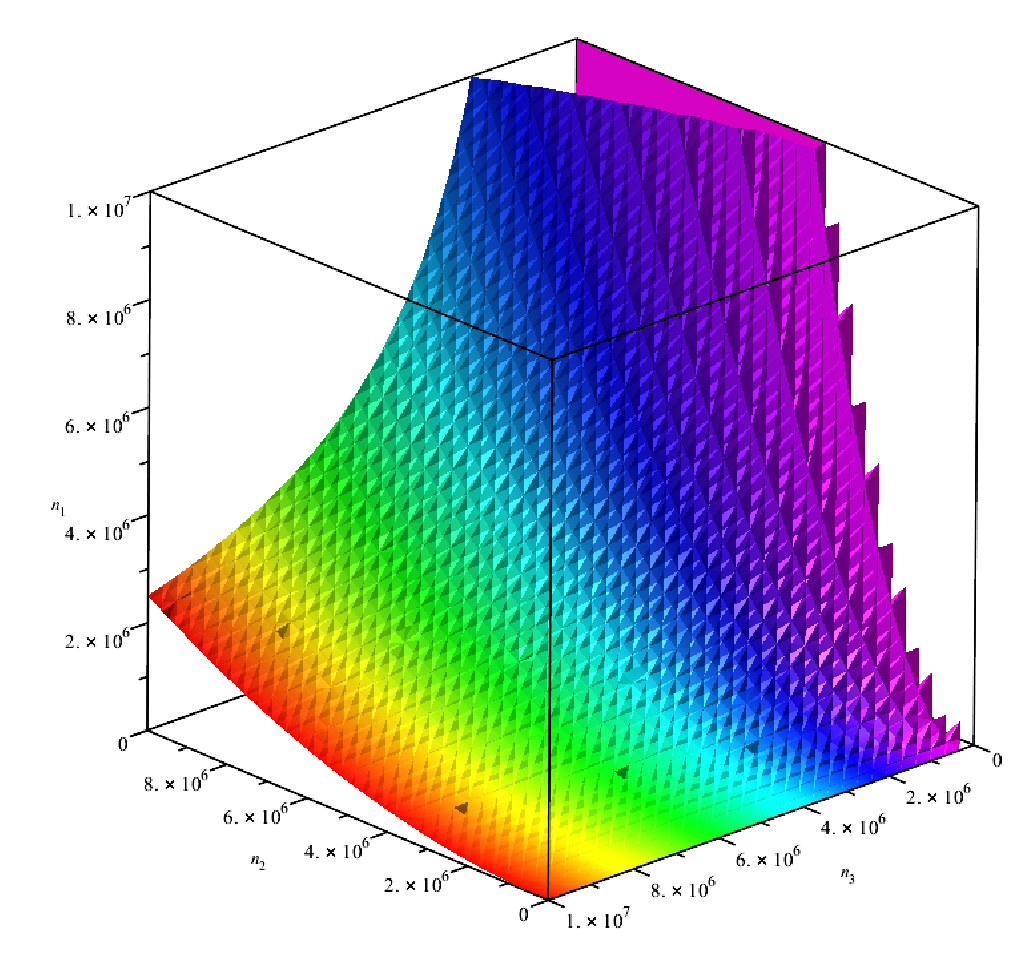
**

Рисунок 3 – графическое решение уравнения , где 

### **2.2.3 Уравнения Колмогорова**

Пусть некоторая система, природа которой нам известна, характеризуется счётным или конечным числом состояний  и переход из состояния в состояние может происходить в любой момент времени. Обозначим вероятность пребывания системы в состоянии  в момент времени  за . Если набор конечен и число состояний равно , принимаем, что . При этом:

|  |  |
| --- | --- |
| . | (2.14) |

Обозначим вероятность перехода из состояния  в состояние  за . Для процессов с непрерывным временем необходимо ввести плотность вероятности перехода  таким образом, чтобы:

|  |  |
| --- | --- |
| . | (2.15) |

Если , то такой процесс называется однородным.

Вероятности состояний  находятся путём решения системы дифференциальных уравнений Колмогорова: [16]

|  |  |
| --- | --- |
| . | (2.16) |

Уравнения составляют по размеченному графу состояний системы, пользуясь следующим мнемоническим правилом:

Производная вероятности каждого состояния равна сумме всех потоков ве­роятности, идущих из других состояний в данное состояние, минус сумма всех потоков вероятности, идущих из данного состояния в другие. [17]

Марковские случайные процессы с дискретным множеством состояний и с непрерывным временем задаются инфинитезимальной матрицей . В однородном процессе, когда , 

Если инфинитезимальная матрица  постоянна, то есть если процесс однороден, переходные вероятности удовлетворяют следующим системам дифференциальных уравнений:

|  |  |
| --- | --- |
| . | (2.17) |
| . | (2.18) |

Первая система уравнений называется прямыми уравнениями Колмогорова, вторая – обратными уравнениями Колмогорова.

Доказательство этого напрямую следует из уравнения Колмогорова-Чепмена:

|  |  |
| --- | --- |
| . | (2.19) |

Действительно, из уравнения (2.8) следует, что:

|  |  |
| --- | --- |
| . | (2.20) |

Таким образом, получили прямое уравнение Колмогорова.

Если продифференцировать уравнение Колмогорова-Чепмена по  при , получим обратное уравнение Колмогорова:

|  |  |
| --- | --- |
| . | (2.21) |
|  |  |

### **2.2.4 Уравнение Фоккера-Планка и нахождение плотности вероятности эволюции системы**

Детерминированный подход хорошо подходит для тех случаев, когда размеры молекул достаточно большие и стохастическими флуктуациями можно пренебречь. В том случае, когда всё-таки нужно учитывать стохастические эффекты, удобно бывает записать для системы уравнение Фоккера-Планка. Уравнение Фоккера-Планка – это дифференциальное уравнение в частных производных, описывающее временную эволюцию функции плотности вероятности системы. Частным случаем уравнения Фоккера-Планка является уравнение Эйнштейна-Смолуховского, впервые полученное при описании броуновского движения.

Рассмотрим дифференциальный оператор второго порядка с непрерывными коэффициентами :

|  |  |
| --- | --- |
| . | (2.22) |

Запишем для него прямое уравнение Колмогорова (2.20):

|  |  |
| --- | --- |
| . | (2.23) |

Уравнение (2.23) называется уравнением Фоккера-Планка. С физической точки зрения уравнение Фоккера-Планка описывает обобщённый диффузионный процесс, где вектор  называется вектором сноса,  – тензором диффузии.

Обратное уравнение Колмогорова (2.21) в этом случае даёт следующий результат:

|  |  |
| --- | --- |
| . | (2.24) |

В одномерном случае уравнение Фоккера-Планка записывается в следующем виде:

|  |  |
| --- | --- |
| , | (2.25) |

причём функция  связана с функциями  и  следующим стохастическим дифференциальным уравнением: [18]

|  |  |
| --- | --- |
| . | (2.26) |

Рассмотрим теперь уравнение Фоккера-Планка (2.25) для модели Шлёгля (2.7):

|  |  |
| --- | --- |
| , | (2.27) |

где

|  |  |
| --- | --- |
| , | (2.28) |
|  |  |
| . | (2.29) |
|  |  |

При  уравнение Фоккера-Планка становится стационарным и не зависит от времени, поэтому можно записать:

|  |  |
| --- | --- |
| . | (2.30) |

Решая это уравнение, получим:

|  |  |
| --- | --- |
| , | (2.31) |

где  – константа, нормализующее распределение. Можно показать, что это решение является точным для систем с гауссовым белым шумом [15].

Рассмотрим подынтегральное выражение в (2.31):

|  |  |
| --- | --- |
|  | (2.32) |

Введём следующие обозначения:

|  |  |
| --- | --- |
|  | (2.33) |

Тогда:

|  |  |
| --- | --- |
|  | (2.34) |

Теперь, с учётом (2.31), (2.32), (2.33) и (2.34), получаем выражение для плотности вероятности эволюции системы, которое описывает распределение состояний системы при :

|  |  |
| --- | --- |
| , | (2.35) |

где  можно получить из условия нормировки:

|  |  |
| --- | --- |
| . | (2.36) |

### **2.2.5 Система с бистабильным поведением**

Рассмотрим частный случай модели Шлёгля, когда система имеет бистабильное поведение. Исходя из анализа детерминированной модели (2.13), система имеет бистабильное поведение тогда и только тогда, когда дискриминант многочлена :

|  |  |
| --- | --- |
| . | (2.37) |

Возьмём конкретные значения интенсивностей переходов в схеме (2.7), удовлетворяющие (2.37). Пусть, например, интенсивности заданы следующим образом:

|  |  |
| --- | --- |
|  | (2.38) |

И при подстановке в (2.36) получаем 

Задача Коши, в данном случае, записывается следующим образом:

|  |  |
| --- | --- |
|  | (2.39) |

Выпишем особые точки этого дифференциального уравнения:

|  |  |
| --- | --- |
|  | (2.40) |

Найдём для этой задачи функцию плотности вероятности эволюции системы (2.34):

|  |  |
| --- | --- |
| , | (2.41) |

где 

Заметим, что при  поведение функции является бимодальным, что соответствует двум устойчивым детерминированным состояниям  и . Таким образом, и анализ уравнения Фоккера-Планка, и анализ детерминированной модели указывают на бимодальность системы с одной нестабильной фиксированной точкой между двумя стабильными фиксированными точками.

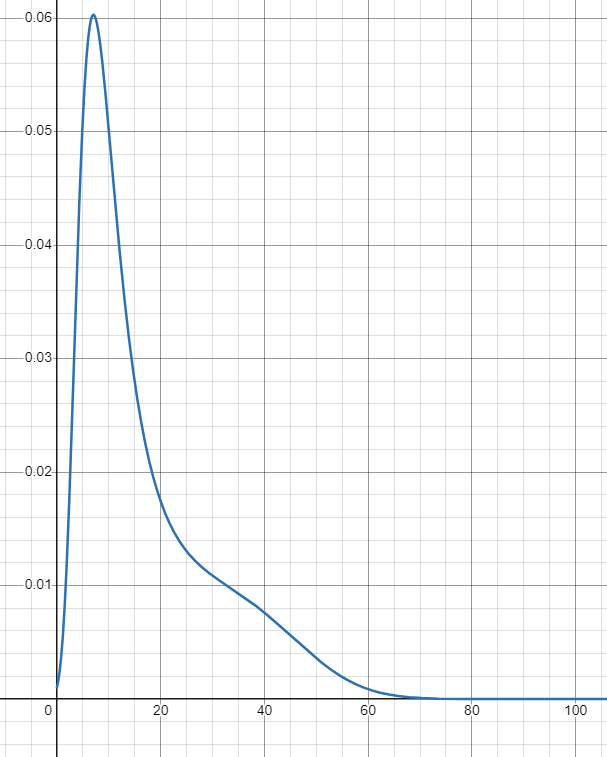


Рисунок 4 – Плотность вероятности эволюции системы с интенсивностями (2.38)

С течением времени наиболее вероятным исходом эволюции системы будет схождение к одному из двух стабильных состояний.

Теперь рассмотрим другой случай. Пусть особые точки дифференциального уравнения (2.8) зависят от случайной величины , причём:

|  |  |
| --- | --- |
|  | (2.42) |

В таком случае получаем, что:

|  |  |
| --- | --- |
|  | (2.43) |

Из (2.43) можно получить выражения для математических ожиданий интенсивностей:

|  |  |
| --- | --- |
|  | (2.44) |

Таким образом, математическое ожидание интенсивностей (2.45) совпадает со значением интенсивностей из прошлого примера (2.38), поэтому при небольшом среднеквадратичном отклонении получаемые результаты должны слабо отличаться от результатов предыдущего примера.

Применим стохастический метод Галёркина к данной системе.



Рисунок 5 – аппроксимация плотности эволюции системы (2.42) стохастическим методом Галёркина

Разложение проводилось до 8-го порядка; среднеквадратичная ошибка аппроксимации  в данном случае оказалась равна 

Построим сечение данного графика для :

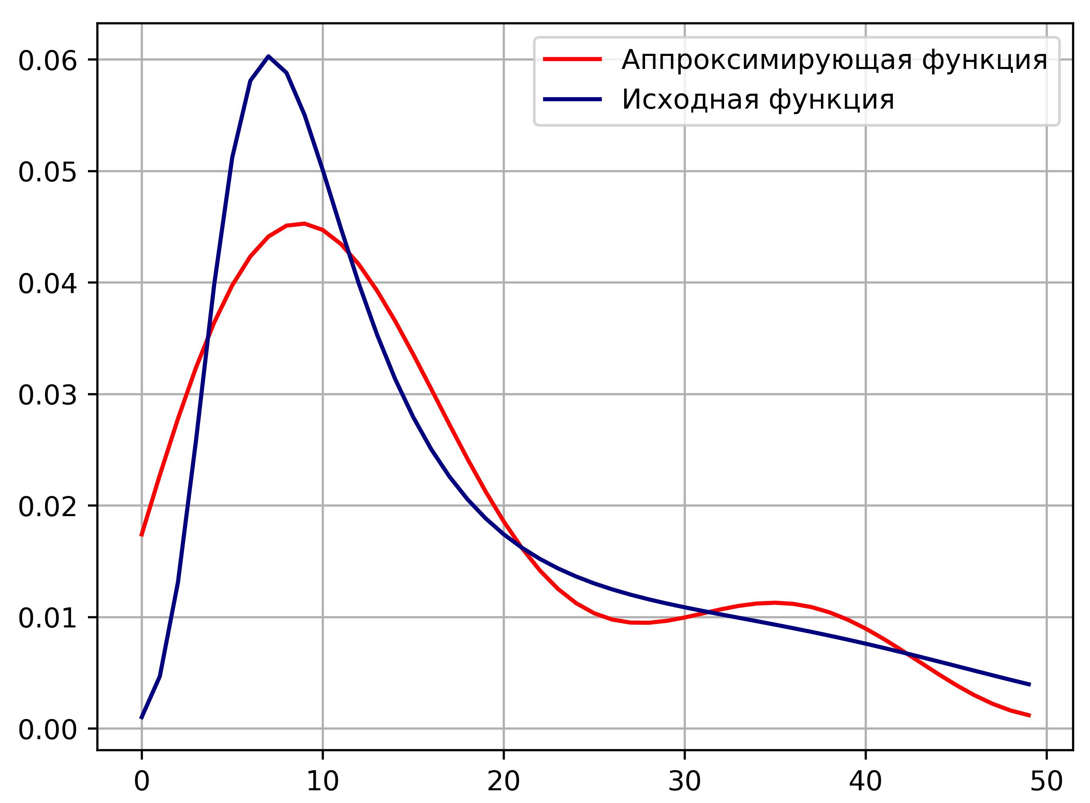


Рисунок 6 – сечение аппроксимирующей функции, построенной с помощью стохастического метода Галёркина (синий цвет – исходная функция, красный цвет – аппроксимация)

Как видно из рисунка 6, аппроксимирующая функция сильно отличается по значениям в точках от аппрокисмируемой. Однако можно заметить, что средние значения на рассматриваемом нами множестве  функций довольно близки: взяв интеграл по рассматриваемому нами множеству от обоих функций, мы получаем довольно близкие значения:

|  |  |
| --- | --- |
|  | (2.45) |

где  – исходная функция (2.41),  – аппроксимирующая функция.

Зададим следующие обозначения:

|  |  |
| --- | --- |
|  | (2.46) |

Нетрудно видеть, что  – не что иное, как функция распределения эволюции системы, а  – аппроксимация этой функции

Проведём сравнение этих двух функций:

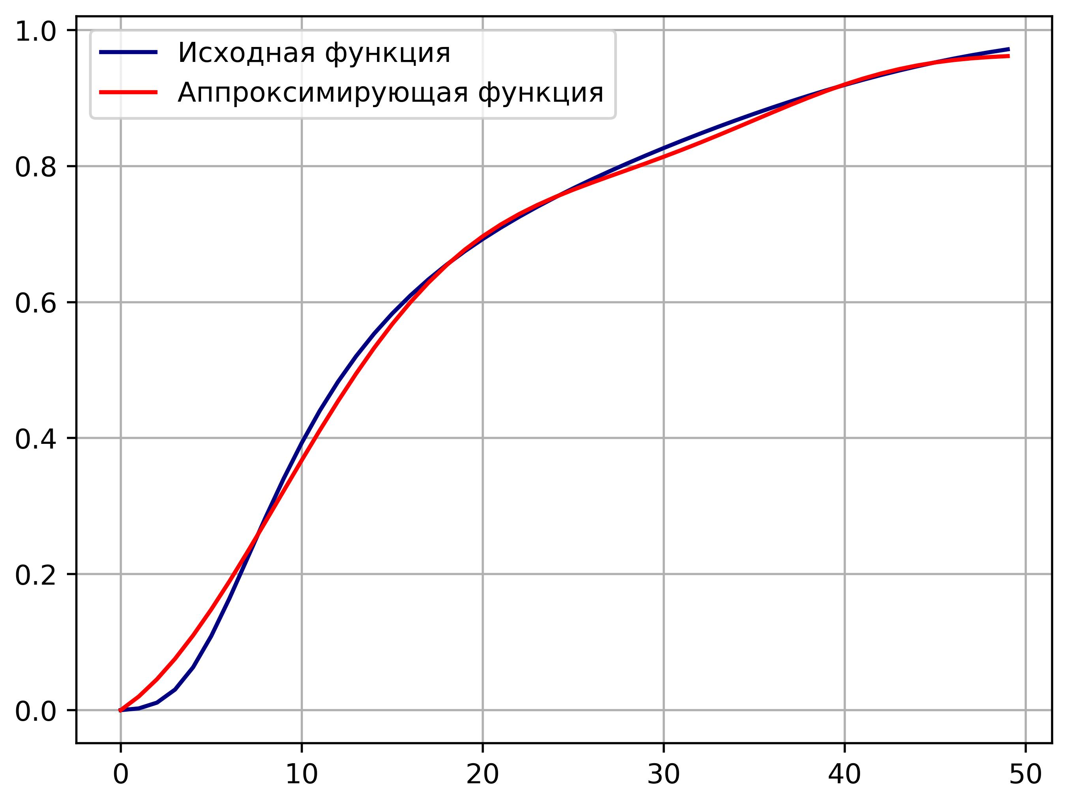


Рисунок 7 – графики функций  (синий цвет) и  (красный цвет)

Среднеквадратичная ошибка аппроксимации в данном случае составила

.

Таким образом, стохастический метод Галёркина хорошо подходит для тех случаев, когда важна не близость значений функций в определённых точках, а схожесть средних значений функций в определённых областях.

### **2.2.6 Система с моностабильным поведением**

Аналогично рассмотрим частный случай модели Шлёгля, когда система имеет моностабильное поведение. В этом случае дискриминант (2.37) многочлена  .

Возьмём конкретные значения интенсивностей переходов в схеме (2.7), удовлетворяющие этому неравенству. Пусть, например, интенсивности заданы следующим образом:

|  |  |
| --- | --- |
|  | (2.47) |

И при подстановке в (2.37) получаем 

Задача Коши, в данном случае, записывается следующим образом:

|  |  |
| --- | --- |
|  | (2.48) |

Выпишем соответствующие этой системе особые точки:

|  |  |
| --- | --- |
|  | (2.49) |

Выпишем функцию плотности эволюции системы:

|  |  |
| --- | --- |
| , | (2.50) |

где 

Функция плотности вероятности эволюции системы является унимодальной и достигает своего максимума приблизительно в действительной особой точке дифференциального уравнения

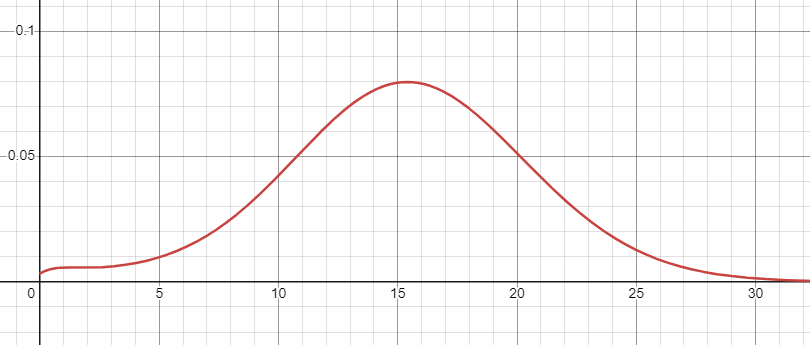


Рисунок 8 Плотность вероятности эволюции системы с интенсивностями (2.47)

Пусть теперь, также по аналогии с бистабильным случаем, интенсивности зависят от случайной величины  следующим образом:

|  |  |
| --- | --- |
|  | (2.51) |

Выпишем математическое ожидание интенсивностей (2.51):

|  |  |
| --- | --- |
|  | (2.52) |

Таким образом, математическое ожидание интенсивностей (2.51) совпадает со значением интенсивностей из прошлого примера (2.47), поэтому при небольшом среднеквадратичном отклонении получаемые результаты должны слабо отличаться от результатов предыдущего примера.

Применим стохастический метод Галёркина к данной системе. Разложение проводилось до 8-го порядка. Построим сечение для :

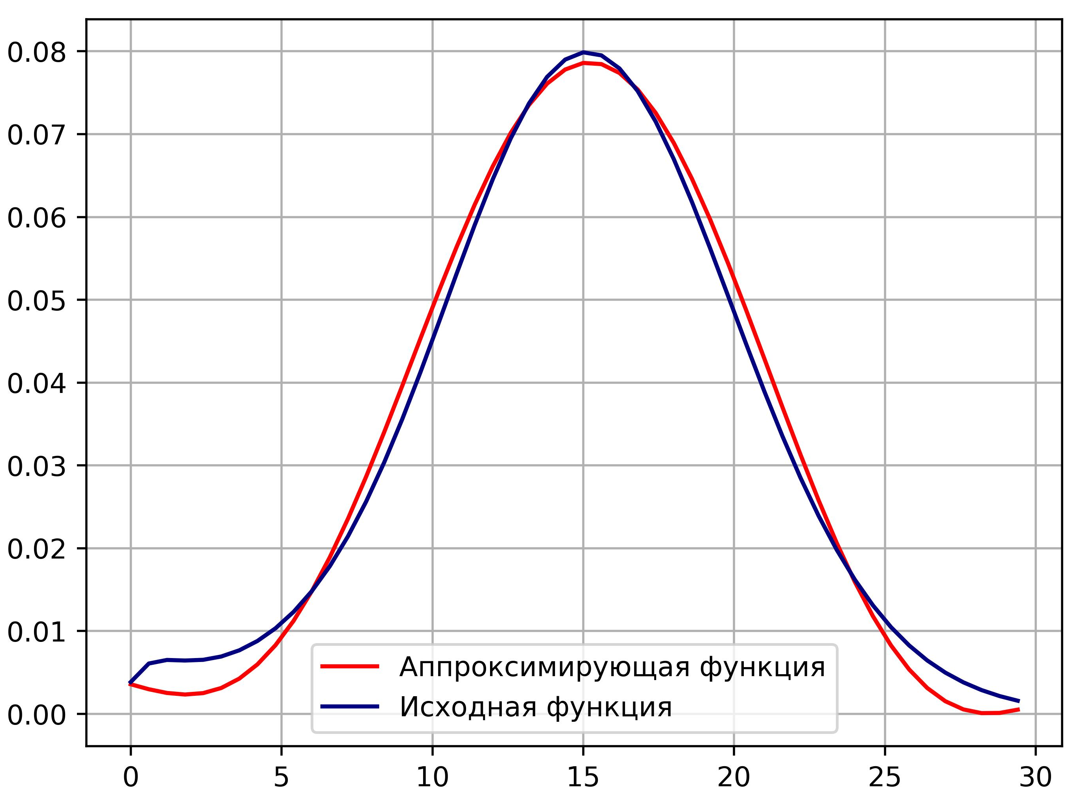


Рисунок 9 – сечение аппроксимирующей функции, построенной с помощью стохастического метода Галёркина (синий цвет – исходная функция, красный цвет – аппроксимация)

Среднеквадратичная ошибка аппроксимации составила .

Аналогично бистабильному случаю, рассмотрим функции распределения эволюции системы:

|  |  |
| --- | --- |
|  | (2.53) |

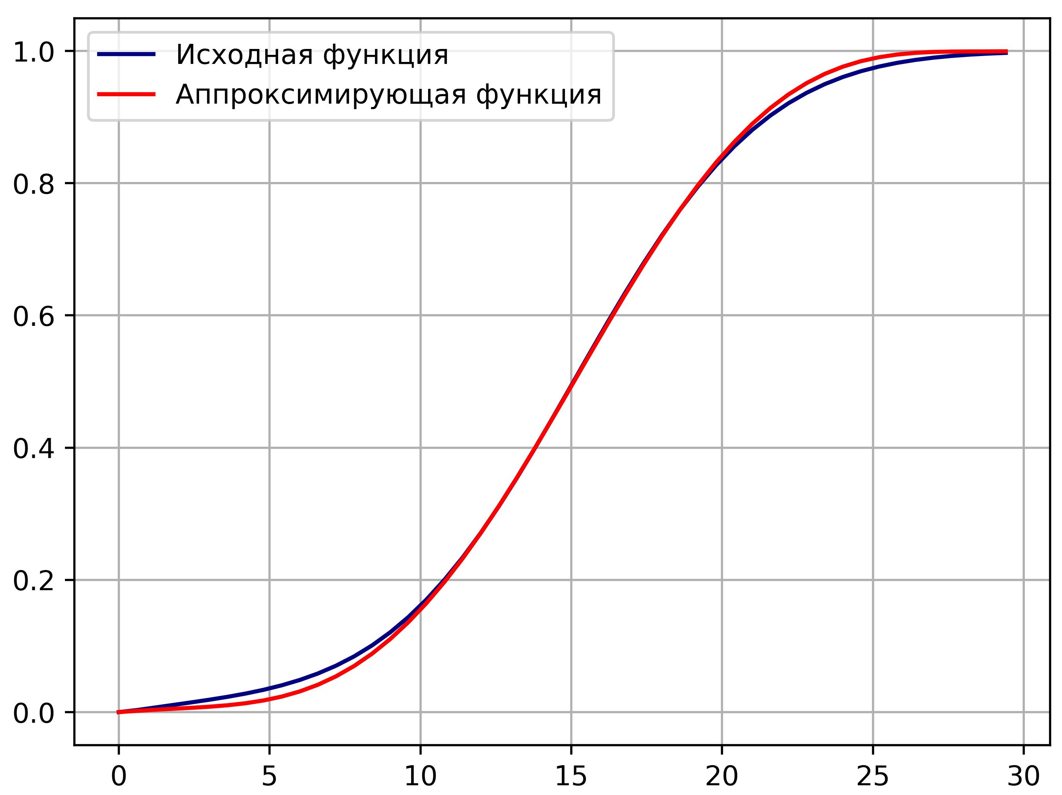


Рисунок 10 – графики функций  (синий цвет) и  (красный цвет)

Среднеквадратичная ошибка аппроксимации составила .

Приведём также для моностабильного случая наборы траекторий при вариации величины .

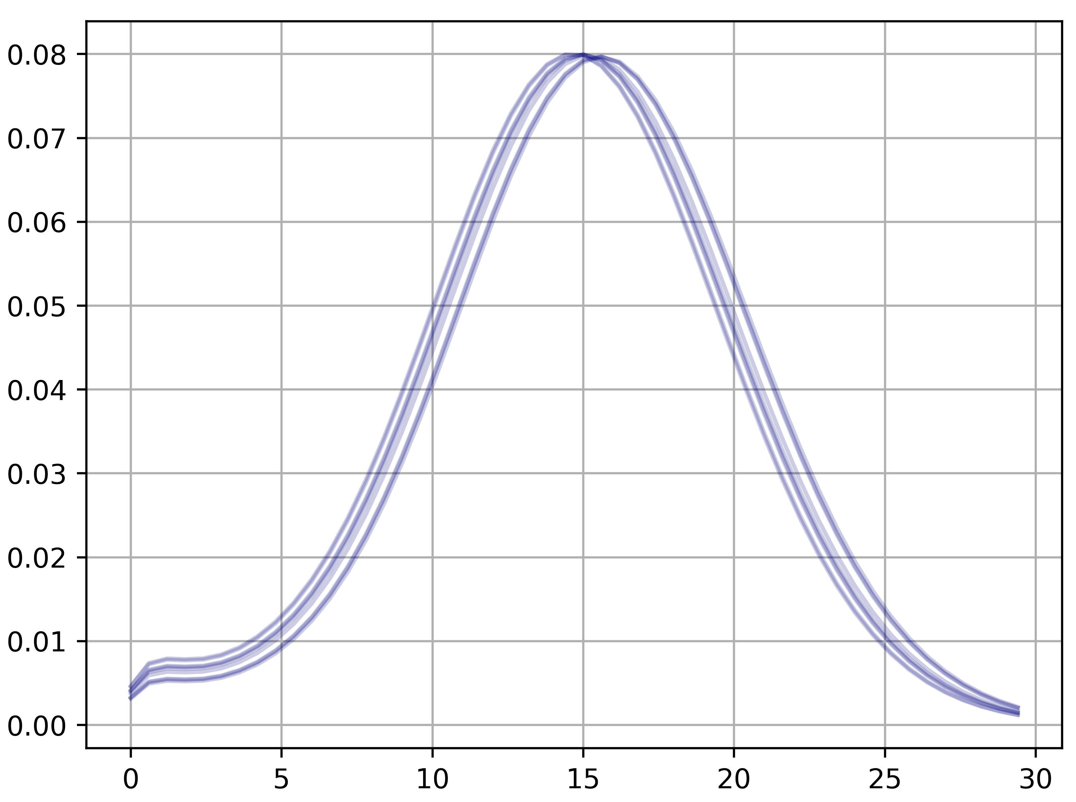


Рисунок 11 – Набор траекторий 

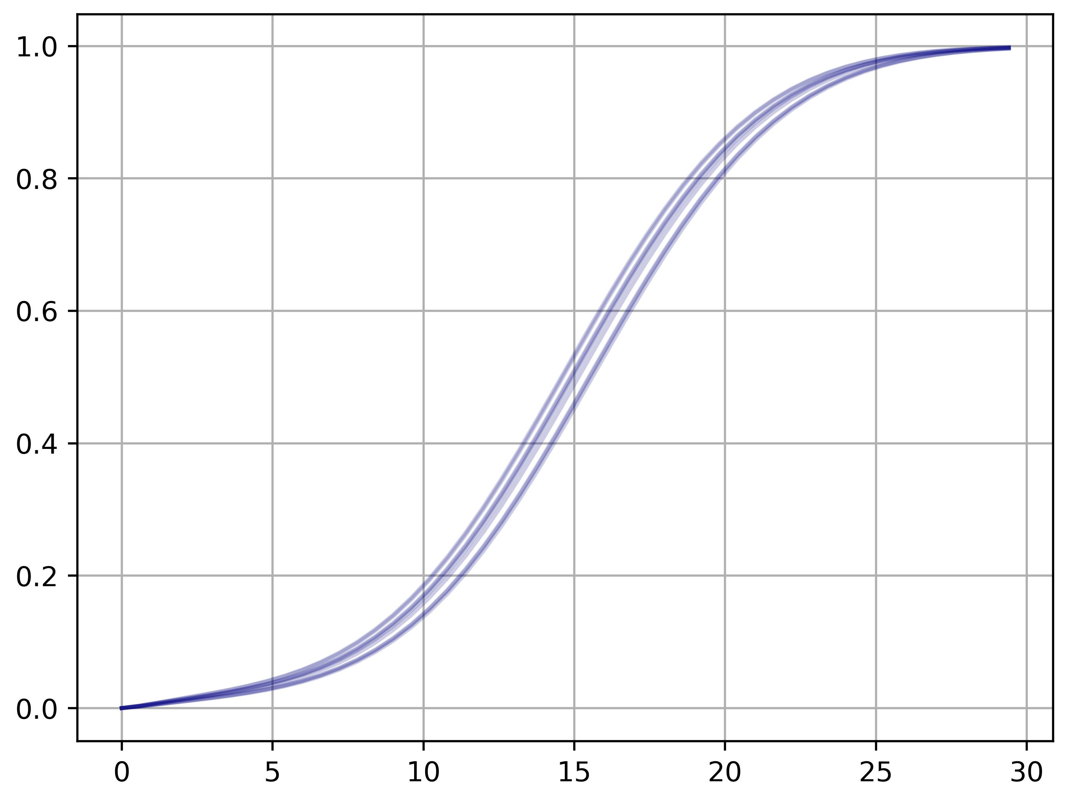


Рисунок 12 – Набор траекторий 

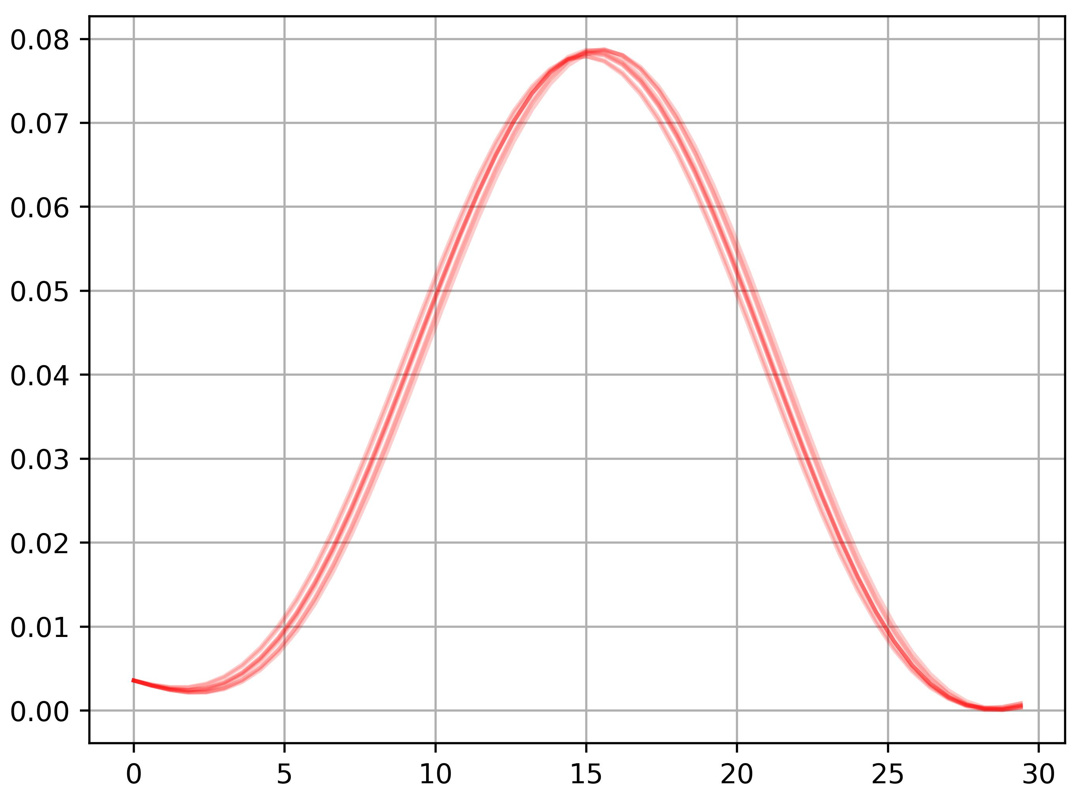


Рисунок 13 – Набор траекторий 

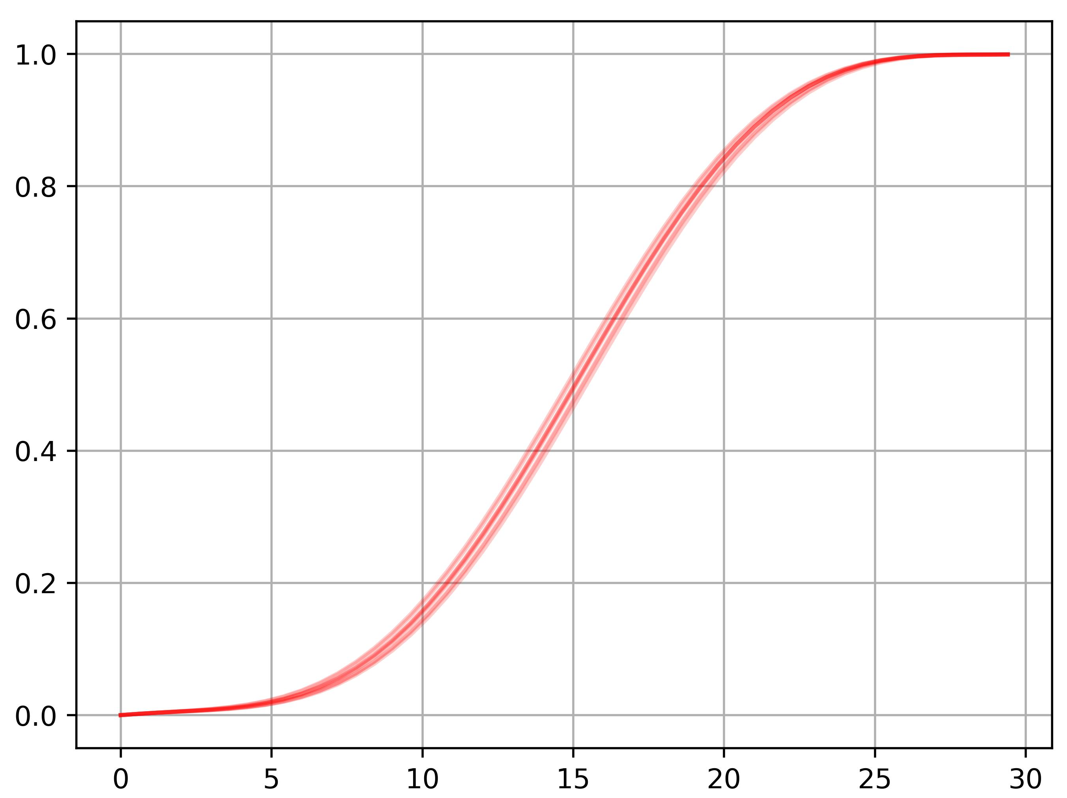


Рисунок 14 – Набор траекторий 

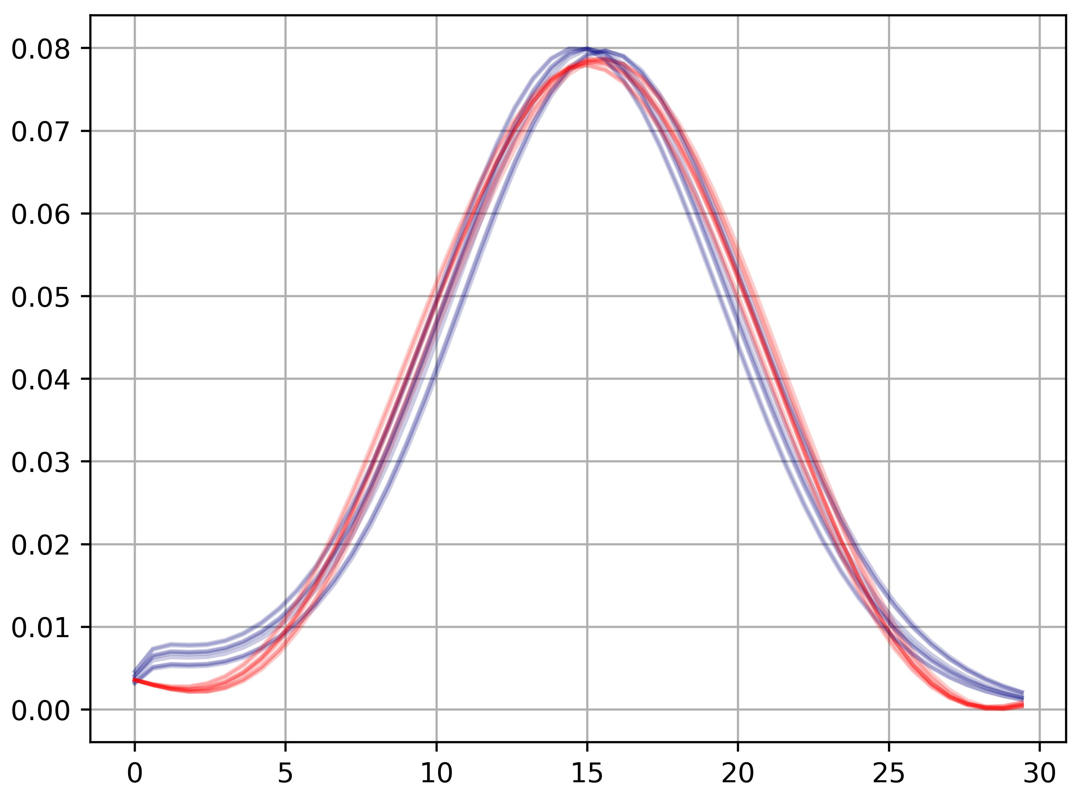


Рисунок 15 – Совмещённые оценки для плотности вероятности эволюции системы

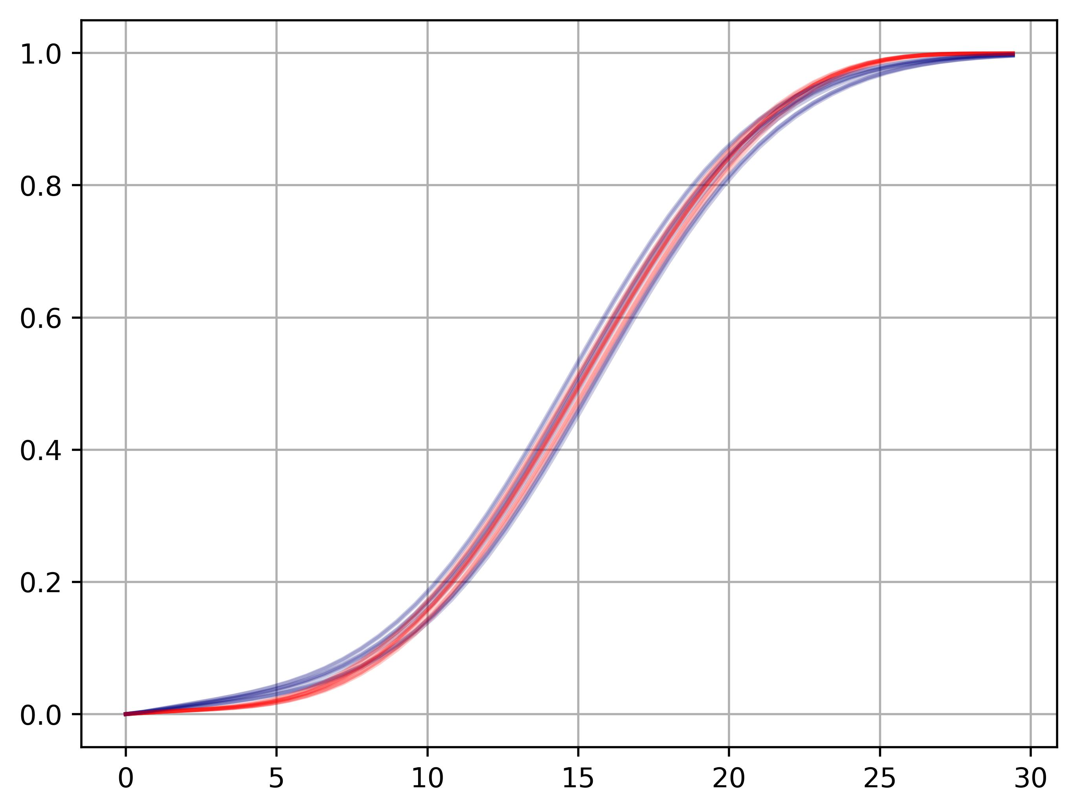


Рисунок 16 – Совмещённые оценки для функции распределения эволюции системы

Таким образом, стохастический метод Галёркина наиболее хорошо подходит для оценки интегральных параметров функции, а не значений функции в отдельных точках. Также стохастический метод Галёркина эффективен для анализа поведения системы, если параметры системы зависят от случайных параметров, законы распределения которых известны, так как метод даёт возможность варьировать эти параметры, не пересчитывая аппроксимирующую функцию. Возможность эффективного анализа как раз и демонстрируется рисунками выше.

## **2.3 Задача о линейном затухающем осцилляторе**

Очень часто в реальных физических задачах многие постоянные в рамках этих задач параметры системы бывают измерены реальными приборами с некоторыми погрешностями. В тех случаях, когда приборы обеспечивают достаточную точность, можно игнорировать стохастические эффекты, возникающие в силу случайности заданных параметров. В том случае, если игнорировать данный факт нельзя, необходимо прибегать к определённым методам, которые позволяют решать задачи подобного рода. Стохастический метод Галёркина как раз и является одним из таких методов, позволяя рассматривать измеренные с погрешностью параметры как функции некоторых случайных величин, законы распределения которых известны. В качестве примера подобной реальной задачи была выбрана задача о линейном затухающем осцилляторе, потому что эта задача позволяет продемонстрировать возможность применения стохастического метода Галёркина для решения обыкновенных дифференциальных уравнений со случайными параметрами, при этом данная задача имеет простое аналитическое решение, с которым можно было бы сравнить результаты, полученные при применении метода.

Рассмотрим модель линейного затухающего осциллятора:

|  |  |
| --- | --- |
|  | (2.54) |

где  – масса тела,  – коэффициент затухания колебаний,  – жёсткость пружины,  – амплитуда вынуждающей силы,  – циклическая частота вынуждающей силы,  – начальное положение,  – начальная скорость. Примем также следующие ограничения на параметры системы:

|  |  |
| --- | --- |
|  | (2.55) |

где  – собственная частота колебаний системы, 

При данных ограничениях в системе будет установлен колебательный режим без эффектов резонанса.

Выпишем общее решение дифференциального уравнения (2.54):

|  |  |
| --- | --- |
| , | (2.56) |

где:

|  |  |
| --- | --- |
|  | (2.57) |

Решая задачу Коши (2.54), получаем:

|  |  |
| --- | --- |
|  | (2.58) |

Зададим конкретные значения характеристик системы:

|  |  |
| --- | --- |
|  | (2.59) |

Найдём решение задачи при детерминированных параметрах (2.59), найдя предварительно все необходимые коэффициенты и параметры:

|  |  |
| --- | --- |
|  | (2.60) |
|  |  |



Рисунок 17 – Решение задачи о линейном затухающем осцилляторе при детерминированных параметрах (2.59)

Рассмотрим теперь тот случай, когда коэффициент затухания колебаний измерен с нормально распределённой ошибкой, то есть пусть , а остальные параметры при этом возьмём из (2.59). Поскольку модель (2.54) является динамической, коэффициенты в разложении полиномиального хаоса являются скалярными функциями от времени:

|  |  |
| --- | --- |
| . | (2.61) |

Решим данную задачу методом Галёркина. Пусть:

|  |  |
| --- | --- |
| ,. | (2.62) |

Теперь представим функцию  в виде разложения полиномиального хаоса:

|  |  |
| --- | --- |
| . | (2.63) |

где 

Подставим при этом начальные условия, получим следующие соотношения:

|  |  |
| --- | --- |
| . | (2.64) |

Заметим, при этом, что, с одной стороны:

|  |  |
| --- | --- |
| , | (2.65) |

а с другой стороны:

|  |  |
| --- | --- |
| . | (2.66) |

Отсюда следует, что:

|  |  |
| --- | --- |
| ,. | (2.67) |

Аналогично:

|  |  |
| --- | --- |
| ,. | (2.68) |

Подставим теперь функцию  в виде РПХ в (2.54):

|  |  |
| --- | --- |
| , | (2.69) |

при этом .

Упростим выражение (2.69), используя свойства полиномов Эрмита. Прежде всего, из свойства (1.29) следует, что:

|  |  |
| --- | --- |
| . | (2.70) |

Из этого следует, что:

|  |  |
| --- | --- |
| . | (2.71) |

Умножим теперь каждое слагаемое в (2.70) на базисный многочлен ,:

|  |  |
| --- | --- |
| , | (2.72) |
|  |  |
| , | (2.73) |
|  |  |
| , | (2.74) |
|  |  |
| , | (2.75) |
|  |  |
| , | (2.76) |
|  |  |
| . | (2.77) |
|  |  |

Наконец, запишем итоговую систему, отдельно расписав случаи для  и :

|  |  |
| --- | --- |
|  | (2.78) |

Численно решая эту систему методом Рунге-Кутта третьего порядка, получаем в итоге следующие результаты

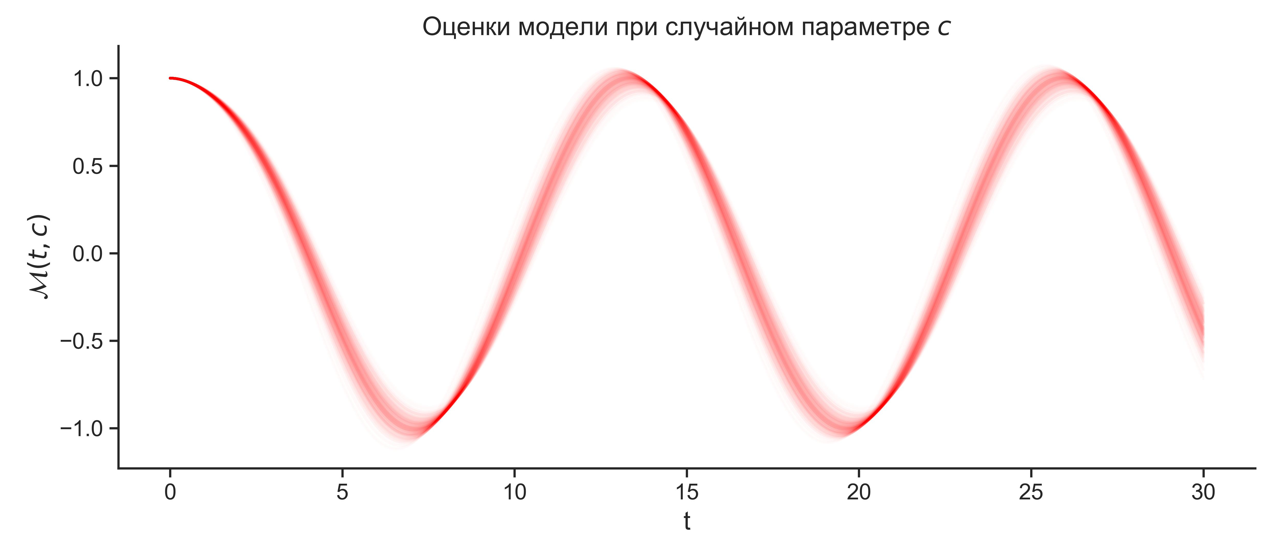


Рисунок 18 – Набор траекторий  при вариации коэффициента затухания колебаний

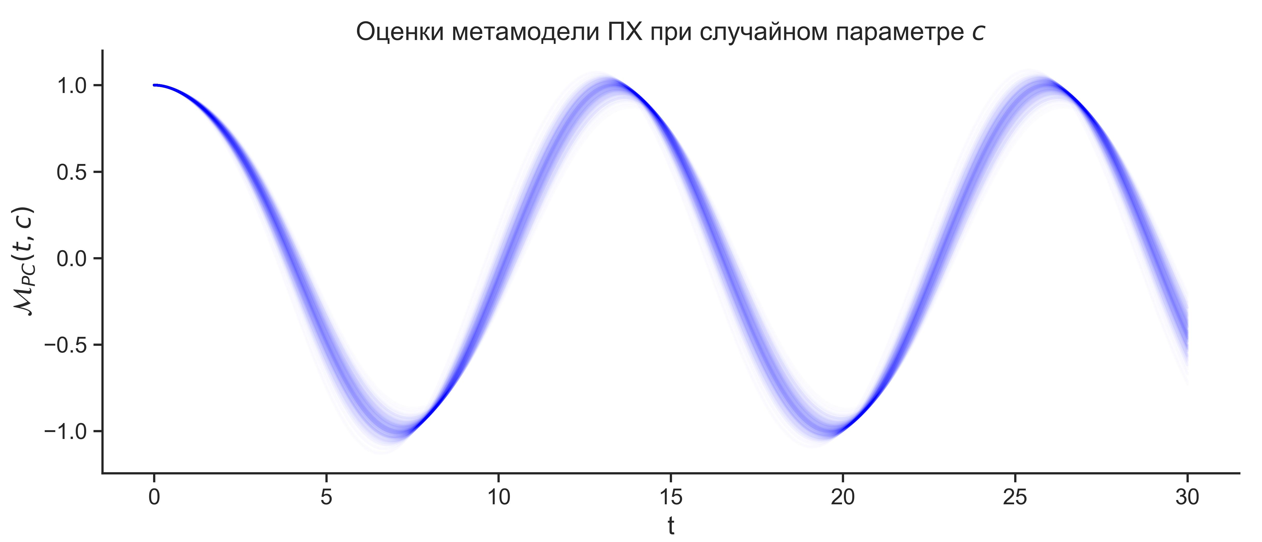
**

Рисунок 19 – Набор траекторий  при вариации коэффициента затухания колебаний

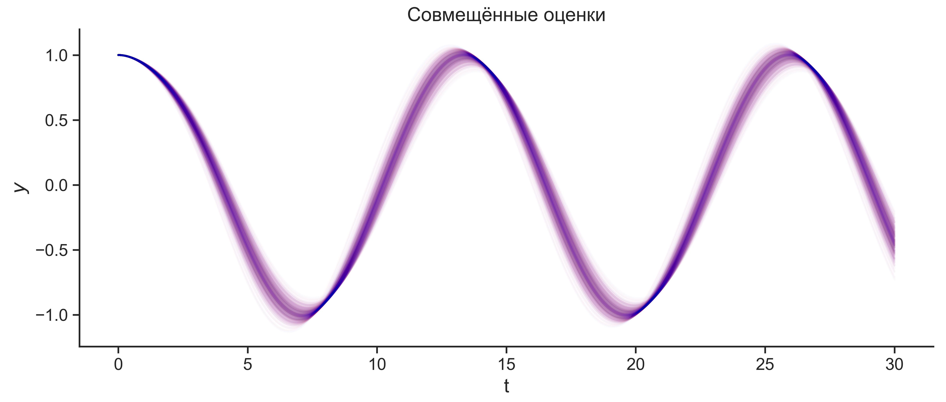
**

Рисунок 20 – Совмещённые траектории модели и метамодели ПХ при вариации коэффициента затухания колебаний

Приведём также зависимость математического ожидания и стандартного отклонения траекторий  от времени.

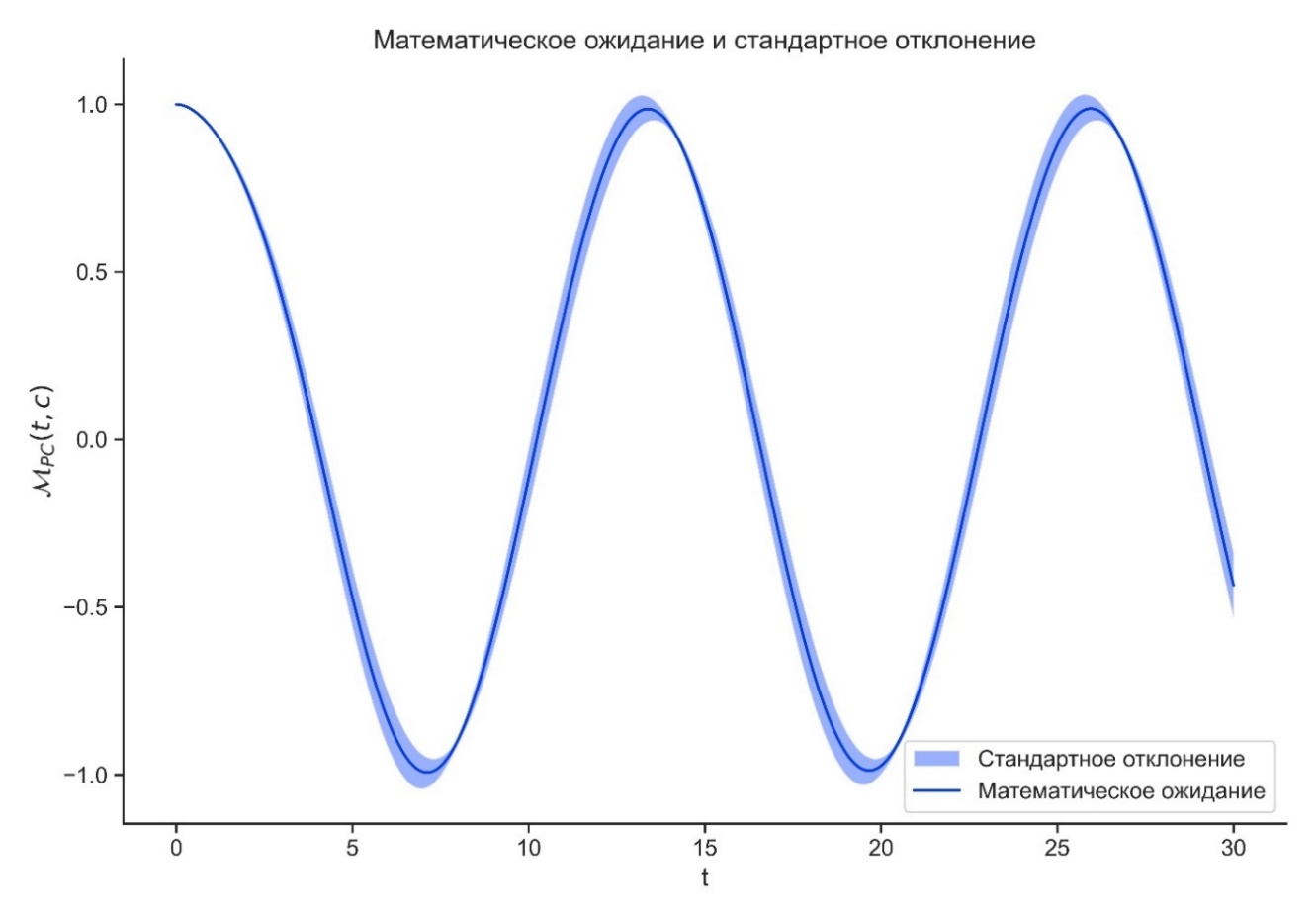
**

Рисунок 21 – Математическое ожидание и стандартное отклонение траекторий метамодели ПХ

Ошибка аппроксимации при данном подходе составила 

Таким образом, стохастический метод Галёркина можно эффективно использовать для решения дифференциальных уравнений в том случае, если характеристики данного уравнения являются случайными величинами, которые можно выразить как функции от случайных величин с известным ортонормированным базисом. Более того, стохастический метод Галёркина остаётся достаточно точным даже в том случае, когда среднеквадратичное отклонение  оказывается достаточно большим.

# ЗАКЛЮЧЕНИЕ

В теоретической части работы всесторонне рассмотрена проблема подбора модели, описывающей изменения состояния системы под действием различных факторов стохастической природы. Подробно описан подход, основанный на разложении полиномиального хаоса, который позволяет не только использовать знания или предположения о законе распределения случайных входных данных, но и в интрузивной постановке учитывать устройство модели, что особенно важно в не имеющих аналитических решений задачах механики. Изучены свойства ортогональной системы полиномов Эрмита, соответствующие нормальному закону распределения входных данных. Уделено внимание сравнение стохастического метода Галёркина со спектральными методами анализа регрессионных моделей, которые, являясь неинтрузивными, воспринимают модель как чёрный ящик, внутреннее устройство которого не влияет на результат аппроксимации. Коэффициенты разложения считаются численно, с помощью, например, метода наименьших квадратов.

В практической части работы реализован программный код для вычисления стохастической проекции Галёркина. На тестовых примерах проведён сравнительный анализ качества полученной модели с моделью МНК на основе полиномов Колмогорова-Габора. Установлено, что при моделировании аналитических и специальных тестовых функций от случайной нормальной величины, меньшая среднеквадратичная ошибка получается методом МНК при малом разбросе входных значений, в то время как при больших среднеквадратичных отклонениях предпочтение следует отдать стохастической проекции Галёркина.

При исследовании модели Шлёгля установлено, что метод лучше всего подходит для оценки интегральных параметров системы. Также метод эффективен для анализа поведения системы, если параметры системы зависят от случайных параметров, законы распределения которых известны, так как он даёт возможность варьировать эти параметры, не пересчитывая аппроксимирующую функцию.

На примере задачи о линейном затухающем осцилляторе со случайным коэффициентом затухания колебаний показано, что метод стохастический проекций Галёркина крайне эффективен для решения дифференциальных уравнений, поскольку даёт достаточно точную аппроксимацию решения, а также, в силу интрузивного характера, позволяет при этом максимально учитывать структуру функциональной зависимости.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Соболь И. М. Численные методы Монте-Карло. – М., Наука, 1967 г.
2. Пупков К. А. Вероятностная неопределённость в стохастических технических системах управления. – Инженерный журнал: наука и инновации, 2013 г., №10 (22).
3. Sudret B., Mai C. Computing derivative-based global sensitivity measures using polynomial chaos expansions. – 2015.
4. Parekh J., Verstappen R. Intrusive polynomial chaos for CFD using OpenFOAM. – Computational Science, vol 12143. Springer, 2020.
5. Kaintura A., Dhaene T., Spina D. Review of polynomial chaos-based methods for uncertainty quantification in modern integrated circuits. – Electronics, 2018.
6. Alekseev A. K., Navon I. M., Zelentsov M. E. The estimation of functional uncertainty using polynomial chaos and adjoint equations. – Int. J. Numer. Meth. Fluids, 67, 2011.
7. Berveiller M., Sudret B., Lemaire M. Stochastic finite element: A non-intrusive approach by regression. – European Journal of Computational Mechanics, 2006.
8. Канторович Л.В., Крылов В.И. Приближённые методы высшего анализа. — 5-е изд. — Л.-М., 1962.
9. Neckel T. Lecture 7, Polynomial Chaos Approximation 2: The stochastic Galerkin approach. – Algorithms for Uncertainty Quantification, Technische Universität München, 2018.
10. Сергиенко А.Б. Тестовые функции для глобальной оптимизации. – Красноярск: Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева, 2015. – 112 с.
11. А. В. Калинкин, Схемы взаимодействий: детерминированные и стохастические модели: Методические указания к выполнению типового расчета по курсу «Дополнительные главы теории случайных процессов». М.: Изд-во МГТУ им. Н.Э. Баумана, 2009. – 44 с.
12. Эмануэль Н.М., Кнорре Д.Г. Курс химической кинетики. М.: Высш. шк., 1974. 400 с.
13. Математические методы в теории надёжности / Г.Д. Карташов, О.И. Тескин, О.А. Бархатова, С.М. Швартин. М.: Изд-во МГТУ им. Н.Э. Баумана, 1982. 32 с.
14. А. В. Калинкин, Ланге А.М., Мастихин А.В., Шаповников А.А., Численные методы Монте-Карло для моделирования схем взаимодействий при дискретных состояниях // Вестник МГТУ им. Н.Э. Баумана. Сер. «Естественные науки». 2005. № 2. С. 53–74
15. Falk J, Mendler M, Drossel B, A minimal model of burst-noise induced bistability, PLoS ONE 12(4): e0176410, 2017. – 15 с.
16. И. К. Волков, С. М. Зуев, Г. М. Цветкова, Случайные процессы: Учеб. для вузов / Под ред. B.C. Зарубина, А.П. Крищенко. - М.: Изд-во МГТУ им. Н.Э. Баумана, 1999. – 448 с. (Сер. Математика в техническом университете; Вып. XVIII).
17. Рассказова М.Н. Имитационное моделирование систем: учебное пособие   
    / М. Н. Рассказова. – Омск: Омский государственный институт сервиса, 2010. – 80 с.
18. Gardiner C. Stochastic Methods – A Handbook for the Natural and Social. Springer; 2009.

# ПРИЛОЖЕНИЕ А Программный код на языке Python

galerkin.py:

import numpy as np

import matplotlib.pyplot as plt

import scipy.integrate as integrate

class Solver:

@staticmethod

def He(n, x):

def factorial(num):

res = 1

for k in range(2, num + 1):

res \*= k

return res

S = 0

for j in range(int(0.5 \* n) + 1):

S += (-0.5) \*\* j \* np.sqrt(factorial(n)) / (factorial(j) \* factorial(n - 2 \* j)) \* x \*\* (n - 2 \* j)

return S

def draw(self):

pass

class Solver\_1D(Solver):

def \_\_init\_\_(self, f, X, \*, acc=0, p=0, silent=False):

self.f = f

self.X = X

self.X.sort()

self.m = np.mean(self.X)

self.s = np.sqrt(np.var(self.X))

self.acc = acc

self.p = p

self.silent = silent

def galerkin(self):

def a(j):

def h(k):

return self.X[k + 1] - self.X[k]

def I(k):

return self.f(self.X[k]) \* Solver.He(j, (self.X[k] - self.m) / self.s) \* np.exp(-(self.X[k] - self.m) \*\* 2 / (2 \* self.s \* self.s))

S = 0

for k in range(len(self.X) - 1):

S += 0.5 \* h(k) \* (I(k) + I(k + 1))

return S / (np.sqrt(2 \* np.pi) \* self.s)

A = []

j = 0

if self.p > 0:

while j <= self.p:

A.append(a(j))

j += 1

elif self.acc > 0:

k = 0

while k < 6:

x = a(j)

A.append(x)

if np.abs(x) < self.acc:

k += 1

else:

k = 0

j += 1

else:

raise ValueError("p или acc должны быть положительным числом")

if not self.silent:

print("Коэффициенты в разложении полиномиального хаоса: ", A, end='\n\n')

print(f"Разложение производится до {len(A) - 1} порядка", end='\n\n')

def Fest(x):

S = 0

for j in range(len(A)):

S += A[j] \* Solver.He(j, (x - self.m) / self.s)

return S

return Fest

def least\_squares(self):

if self.p == 0:

flag = False

N = 6

while not flag:

M = [[sum([self.X[i] \*\* (k + j) for i in range(len(self.X))]) for k in range(N + 1)] for j in range(N + 1)]

V = [sum([self.f(self.X[i]) \* self.X[i] \*\* j for i in range(len(self.X))]) for j in range(N + 1)]

A = np.linalg.solve(M, V)

if max(map(abs, A[-6:])) < self.acc:

flag = True

else:

flag = False

N += 1

else:

M = [[sum([self.X[i] \*\* (k + j) for i in range(len(self.X))]) for k in range(self.p + 1)] for j in range(self.p + 1)]

V = [sum([self.f(self.X[i]) \* self.X[i] \*\* j for i in range(len(self.X))]) for j in range(self.p + 1)]

A = np.linalg.solve(M, V)

if not self.silent:

print("Коэффициенты ряда: ", A, end='\n\n')

print(f"Разложение производится до {len(A) - 1} порядка", end='\n\n')

def Fest(x):

S = 0

for k in range(len(A)):

S += A[k] \* x \*\* k

return S

return Fest

def compare\_errors(self):

D = np.linspace(self.m - 3 \* self.s, self.m + 3 \* self.s, len(self.X))

g = self.galerkin()

ls = self.least\_squares()

'''plt.plot(D, [g(x) for x in D], color="red")

plt.plot(D, [ls(x) for x in D], color="green")

plt.plot(D, [self.f(x) for x in D], color="navy")

err\_galerkin = sum([(self.f(x) - g(x)) \*\* 2 for x in D]) / len(D)

err\_ls = sum([(self.f(x) - ls(x)) \*\* 2 for x in D]) / len(D)'''

Ih = lambda x: integrate.quad(g, 0, x)[0]

Ik = lambda x: integrate.quad(ls, 0, x)[0]

If = lambda x: integrate.quad(self.f, 0, x)[0]

err\_galerkin = sum([(If(x) - Ih(x)) \*\* 2 for x in D]) / len(D)

err\_ls = sum([(If(x) - Ik(x)) \*\* 2 for x in D]) / len(D)

if not self.silent:

print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Эрмита равна {err\_galerkin}", end='\n\n')

print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Колмогорова-Габора равна {err\_ls}", end='\n\n')

return (err\_galerkin, err\_ls)

def compare(self):

plt.grid(True)

D = np.linspace(self.m - 3 \* self.s, self.m + 3 \* self.s, len(self.X))

g = self.galerkin()

ls = self.least\_squares()

plt.plot(D, [g(x) for x in D], color="red")

plt.plot(D, [ls(x) for x in D], color="green")

plt.plot(D, [self.f(x) for x in D], color="navy")

err\_galerkin = sum([(self.f(x) - g(x)) \*\* 2 for x in D]) / len(D)

err\_ls = sum([(self.f(x) - ls(x)) \*\* 2 for x in D]) / len(D)

'''Ih = lambda x: integrate.quad(g, 0, x)[0]

Ik = lambda x: integrate.quad(ls, 0, x)[0]

If = lambda x: integrate.quad(self.f, 0, x)[0]

plt.plot(D, [Ih(x) for x in D], color="red")

plt.plot(D, [Ik(x) for x in D], color="green")

plt.plot(D, [If(x) for x in D], color="navy")

plt.xlabel("X")

plt.ylabel("Y")

err\_galerkin = sum([(If(x) - Ih(x)) \*\* 2 for x in D]) / len(D)

err\_ls = sum([(If(x) - Ik(x)) \*\* 2 for x in D]) / len(D)'''

print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Эрмита равна {err\_galerkin}", end='\n\n')

print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Колмогорова-Габора равна {err\_ls}", end='\n\n')

plt.show()

def draw(self):

plt.grid(True)

D = np.linspace(self.m - 3 \* self.s, self.m + 3 \* self.s, len(self.X))

g = self.galerkin()

plt.plot(D, [g(x) for x in D], color="red")

plt.plot(D, [self.f(x) for x in D], color="navy")

plt.xlabel("X")

plt.ylabel("Y")

err\_galerkin = sum([(self.f(x) - g(x)) \*\* 2 for x in D]) / len(D)

print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Эрмита равна {err\_galerkin}", end='\n\n')

plt.show()

class Solver\_2D(Solver):

def \_\_init\_\_(self, f, X, Y, \*, acc=0, p=0, silent=False):

self.f = f

self.X = X

self.X.sort()

self.Y = Y

self.Y.sort()

self.mx = np.mean(self.X)

self.my = np.mean(self.Y)

self.sx = np.sqrt(np.var(self.X))

self.sy = np.sqrt(np.var(self.Y))

self.nx = len(self.X)

self.ny = len(self.Y)

self.acc = acc

self.p = p

self.silent = silent

def galerkin(self):

def acalc(k):

def a(i, j):

def hx(kx):

return self.X[kx + 1] - self.X[kx]

def hy(ky):

return self.Y[ky + 1] - self.Y[ky]

def I(kx, ky):

return self.f(self.X[kx], self.Y[ky]) \* Solver.He(i, (self.X[kx] - self.mx) / self.sx) \* Solver.He(j, (self.Y[ky] - self.my) / self.sy) \* np.exp(-(self.X[kx] - self.mx) \*\* 2 / (2 \* self.sx \* self.sx)) \* np.exp(-(self.Y[ky] - self.my) \*\* 2 / (2 \* self.sy \* self.sy))

S = 0

for kx in range(self.nx - 1):

for ky in range(self.ny - 1):

S += hx(kx) \* hy(ky) \* 0.25 \* (I(kx, ky) + I(kx + 1, ky) + I(kx, ky + 1) + I(kx + 1, ky + 1))

return S / (2 \* np.pi \* self.sx \* self.sy)

return [[a(i, j) for j in range(k + 1)] for i in range(k + 1)]

A = []

L = 0

if self.p > 0:

A = acalc(self.p)

L = self.p

elif self.acc != 0:

k = 0

while k < 6:

data = acalc(L)

if max(map(abs, [elem for elem in sum(data, []) if elem not in sum(A, [])])) < self.acc:

k += 1

else:

k = 0

A = data

L += 1

else:

raise ValueError("p или acc должны быть положительным числом")

if not self.silent:

print(f"Разложение производится до {max([len(M) for M in A]) - 1} порядка", end='\n\n')

def Fest(x, y):

S = 0

for i in range(L):

for j in range(L):

S += A[i][j] \* Solver.He(i, (x - self.mx) / self.sx) \* Solver.He(j, (y - self.my) / self.sy)

return S

return Fest

def draw(self):

fig = plt.figure()

axes = fig.add\_subplot(projection='3d')

Dx = np.arange(self.mx - 3 \* self.sx, self.mx + 3 \* self.sx, 6 \* self.sx / (self.nx - 1))

Dy = np.arange(self.my - 3 \* self.sy, self.my + 3 \* self.sy, 6 \* self.sy / (self.ny - 1))

xgrid, ygrid = np.meshgrid(Dx, Dy)

g = self.galerkin()

axes.plot\_surface(xgrid, ygrid, g(xgrid, ygrid), color="red")

axes.plot\_surface(xgrid, ygrid, self.f(xgrid, ygrid), color="navy")

axes.set\_xlabel("X")

axes.set\_ylabel("Y")

axes.set\_zlabel("Z")

err\_galerkin = sum([(self.f(x, y) - g(x, y)) \*\* 2 for x in Dx for y in Dy]) / (len(Dx) \* len(Dy))

print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Эрмита равна {err\_galerkin}", end='\n\n')

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

np.random.seed(100000)

def f(x):

return 10 + x \* x - 10 \* np.cos(2 \* np.pi \* x)

err\_galerkin = 0

err\_ls = 0

N = 20

for i in range(N):

X = [np.random.normal(0, 0.5) for \_ in range(800)]

s1 = Solver\_1D(f, X, p=11, silent=True)

pr = s1.compare\_errors()

err\_galerkin += pr[0]

err\_ls += pr[1]

err\_galerkin /= N

err\_ls /= N

print(err\_galerkin, err\_ls)

FP.py:

from galerkin import Solver\_2D

import numpy as np

import matplotlib.pyplot as plt

import scipy.integrate as integrate

class Solver\_FP:

def \_\_init\_\_(self, f, xmax, m, s, n, acc=0, p=0, path=''):

self.xmax = xmax

self.arrX = [x for x in range(xmax + 1)]

self.arrK = [np.random.normal(m, s) for \_ in range(n)]

print(f"ksi = {self.arrK}", end='\n\n')

self.m = np.mean(self.arrK)

self.s = np.sqrt(np.var(self.arrK))

self.f = f

self.Dx = np.arange(0, xmax, 0.02 \* xmax)

solver = Solver\_2D(f, self.arrX, self.arrK, acc=acc, p=p)

self.galerkin = solver.galerkin()

def wP(ksi):

N = 1 / integrate.quad(lambda y: f(y, ksi), 0, np.inf)[0]

return lambda x: N \* f(x, ksi)

self.wP = wP

def wG(ksi):

N = 1 / integrate.quad(lambda y: self.galerkin(y, ksi), 0, self.xmax)[0]

return lambda x: N \* self.galerkin(x, ksi)

self.wG = wG

def wF(ksi):

P = self.wP(ksi)

return lambda x: integrate.quad(lambda y: P(y), 0, x)[0]

self.wF = wF

def wGF(ksi):

G = self.wG(ksi)

return lambda x: integrate.quad(lambda y: G(y), 0, x)[0]

self.wGF = wGF

N = 1 / integrate.quad(lambda y: f(y, self.m), 0, np.inf)[0]

self.mP = lambda x: N \* f(x, self.m)

print(f"N = {N}", end='\n\n')

gN = 1 / integrate.quad(lambda y: self.galerkin(y, self.m), 0, self.xmax)[0]

self.mG = lambda x: gN \* self.galerkin(x, self.m)

print(f"gN = {gN}", end='\n\n')

self.mF = lambda x: integrate.quad(lambda y: self.mP(y), 0, x)[0]

self.mGF = lambda x: integrate.quad(lambda y: self.mG(y), 0, x)[0]

self.path = path

def check(self, \*, acc):

for x in self.arrX:

if self.mG(x) < 0 or self.mGF(x) < 0 or self.mGF(x) > 1:

return False

return sum([(self.mGF(x) - self.mF(x)) \*\* 2 for x in self.Dx]) / len(self.Dx) < acc

def draw\_middle\_section(self):

#err\_hermite = sum([(self.G(x, ksi) - self.P(x, ksi)) \*\* 2 for ksi in self.arrK for x in self.arrX]) / (len(self.arrX) \* len(self.arrK))

#print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Эрмита равна {err\_hermite}", end='\n\n')

print(f"mksi = {self.m}", end='\n\n')

print(f"sksi = {self.s}", end='\n\n')

'''Dk = np.arange(self.m - 3 \* self.s, self.m + 3 \* self.s, 6 \* self.s / (len(self.arrK) - 1))

xgrid, kgrid = np.meshgrid(Dx, Dk)

fig = plt.figure()

axes = fig.add\_subplot(projection='3d')

axes.plot\_surface(xgrid, kgrid, self.G(xgrid, kgrid), color="red")

axes.plot\_surface(xgrid, kgrid, self.P(xgrid, kgrid), color="navy")

axes.set\_xlabel("X")

axes.set\_ylabel("ksi")

axes.set\_zlabel("Z")

plt.show()'''

plt.grid(True)

plt.plot(self.Dx, [self.mG(x) for x in self.Dx], color="red", label="Аппроксимирующая функция")

plt.plot(self.Dx, [self.mP(x) for x in self.Dx], color="navy", label="Исходная функция")

err2 = sum([(self.mG(x) - self.mP(x)) \*\* 2 for x in self.arrX]) / len(self.arrX)

print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Эрмита (2) равна {err2}", end='\n\n')

plt.legend(loc='best')

plt.savefig(self.path + 'density\_middle\_section.jpg', dpi=500)

plt.close()

print(f"Интеграл от исходной функции равен {self.mF(self.xmax)}", end='\n\n')

print(f"Интеграл от аппроксимации равен {self.mGF(self.xmax)}", end='\n\n')

err3 = sum([(self.mGF(x) - self.mF(x)) \*\* 2 for x in self.arrX]) / len(self.arrX)

print(f"Среднеквадратичная ошибка аппроксимации при использовании полиномов Эрмита (3) равна {err3}", end='\n\n')

plt.grid(True)

plt.plot(self.Dx, [self.mF(x) for x in self.Dx], color="navy", label="Исходная функция")

plt.plot(self.Dx, [self.mGF(x) for x in self.Dx], color="red", label="Аппроксимирующая функция")

plt.legend(loc='best')

plt.savefig(self.path + 'distribution\_middle\_section.jpg', dpi=500)

plt.close()

def draw\_model(self):

arrK = [ksi for ksi in self.arrK if np.abs(ksi - self.m) < self.s]

plt.grid(True)

for ksi in arrK:

P = self.wP(ksi)

plt.plot(self.Dx, [P(x) for x in self.Dx], alpha=0.2, color="navy")

plt.savefig(self.path + 'density\_model.jpg', dpi=500)

plt.close()

plt.grid(True)

for ksi in arrK:

F = self.wF(ksi)

plt.plot(self.Dx, [F(x) for x in self.Dx], alpha=0.2, color="navy")

plt.savefig(self.path + 'distribution\_model.jpg', dpi=500)

plt.close()

def draw\_PC(self):

arrK = [ksi for ksi in self.arrK if np.abs(ksi - self.m) < self.s]

plt.grid(True)

for ksi in arrK:

G = self.wG(ksi)

plt.plot(self.Dx, [G(x) for x in self.Dx], alpha=0.2, color="red")

plt.savefig(self.path + 'density\_PC.jpg', dpi=500)

plt.close()

plt.grid(True)

for ksi in arrK:

GF = self.wGF(ksi)

plt.plot(self.Dx, [GF(x) for x in self.Dx], alpha=0.2, color="red")

plt.savefig(self.path + 'distribution\_PC.jpg', dpi=500)

plt.close()

def draw\_combined(self):

arrK = [ksi for ksi in self.arrK if np.abs(ksi - self.m) < self.s]

plt.grid(True)

for ksi in arrK:

P = self.wP(ksi)

G = self.wG(ksi)

plt.plot(self.Dx, [P(x) for x in self.Dx], alpha=0.2, color="navy")

plt.plot(self.Dx, [G(x) for x in self.Dx], alpha=0.2, color="red")

plt.savefig(self.path + 'density\_combined.jpg', dpi=500)

plt.close()

plt.grid(True)

for ksi in arrK:

F = self.wF(ksi)

GF = self.wGF(ksi)

plt.plot(self.Dx, [F(x) for x in self.Dx], alpha=0.2, color="navy")

plt.plot(self.Dx, [GF(x) for x in self.Dx], alpha=0.2, color="red")

plt.savefig(self.path + 'distribution\_combined.jpg', dpi=500)

plt.close()

class Solver\_BFP(Solver\_FP):

def \_\_init\_\_(self, X, k4, \*, xmax, m, s, n, acc=0, p=0):

G = lambda ksi: 2 \* X[0](ksi) \* (X[0](ksi) + X[1](ksi)) \* (X[0](ksi) + X[2](ksi)) / ((X[0](ksi) - X[1](ksi)) \* (X[0](ksi) - X[2](ksi)))

H = lambda ksi: 2 \* X[1](ksi) \* (X[0](ksi) + X[1](ksi)) \* (X[1](ksi) + X[2](ksi)) / ((X[0](ksi) - X[1](ksi)) \* (X[1](ksi) - X[2](ksi)))

M = lambda ksi: 2 \* X[2](ksi) \* (X[0](ksi) + X[2](ksi)) \* (X[1](ksi) + X[2](ksi)) / ((X[0](ksi) - X[2](ksi)) \* (X[2](ksi) - X[1](ksi)))

print(f"G(1) \* m = {G(1) \* m}", end='\n\n')

print(f"H(1) \* m = {H(1) \* m}", end='\n\n')

print(f"M(1) \* m = {M(1)\* m}", end='\n\n')

B = lambda x, ksi: k4 \* (x + X[0](ksi)) \* (x + X[1](ksi)) \* (x + X[2](ksi))

f = lambda x, ksi: np.exp(2 \* (G(ksi) \* np.log(1 + x / X[0](ksi)) - H(ksi) \* np.log(1 + x / X[1](ksi)) - M(ksi) \* np.log(1 + x / X[2](ksi)) - x)) / B(x, ksi)

super().\_\_init\_\_(f, xmax, m, s, n, acc, p, path='results/BFP/')

class Solver\_MFP(Solver\_FP):

def \_\_init\_\_(self, K, \*, xmax, m, s, n, acc=0, p=0):

def f(x, ksi):

A = lambda y: K[0](ksi) - K[1](ksi) \* y + K[2](ksi) \* y \* y - K[3](ksi) \* y \*\* 3

B = lambda y: K[0](ksi) + K[1](ksi) \* y + K[2](ksi) \* y \* y + K[3](ksi) \* y \*\* 3

g = lambda y: 2 \* integrate.quad(lambda u: A(u) / B(u), 0, y)[0]

return np.exp(g(x)) / B(x)

super().\_\_init\_\_(f, xmax, m, s, n, acc, p, path='results/MFP/')

self.M = lambda ksi: integrate.quad(lambda x: x \* self.wP(ksi)(x), 0, np.inf)

self.GM = lambda ksi: integrate.quad(lambda x: x \* self.wG(ksi)(x), 0, self.xmax)

def draw\_expectation(self):

plt.grid(True)

Dksi = np.arange(self.m - 2 \* self.s, self.m + 2 \* self.s, 0.1 \* 4 \* self.s)

plt.plot(Dksi, [self.M(ksi) for ksi in Dksi], color="navy", label="Математическое ожидание исходной функции")

plt.plot(Dksi, [self.GM(ksi) for ksi in Dksi], color="red", label="Математическое ожидание аппроксимирующей функции")

plt.legend(loc='best')

plt.savefig(self.path + 'expectation.jpg', dpi=500)

plt.close()

class Solver\_BFP2(Solver\_FP):

def \_\_init\_\_(self, K, \*, xmax, m, s, n, acc=0, p=0):

def f(x, ksi):

A = lambda y: K[0](ksi) - K[1](ksi) \* y + K[2](ksi) \* y \* y - K[3](ksi) \* y \*\* 3

B = lambda y: K[0](ksi) + K[1](ksi) \* y + K[2](ksi) \* y \* y + K[3](ksi) \* y \*\* 3

g = lambda y: 2 \* integrate.quad(lambda u: A(u) / B(u), 0, y)[0]

return np.exp(g(x)) / B(x)

super().\_\_init\_\_(f, xmax, m, s, n, acc, p, path='results/BFP2/')

if \_\_name\_\_ == "\_\_main\_\_":

seed = 100009

np.random.seed(seed)

'''def x1(ksi):

return ksi

def x2(ksi):

return 2 \* ksi

def x3(ksi):

return 4 \* ksi

X = [x1, x2, x3]

k4 = 0.001

fp = Solver\_BFP(X, k4, xmax=50, m=10, s=1, n=10, p=18)'''

def k1(ksi):

return 0.1 \* (1 + 4 \* ksi)

def k2(ksi):

return 0.1 \* (1 + 2 \* ksi)

def k3(ksi):

return 0.1 \* (1 + ksi)

def k4(ksi):

return 0.01 \* ksi

K = [k1, k2, k3, k4]

fp = Solver\_MFP(K, xmax=30, m=1, s=0.1, n=20, p=18)

fp.draw\_expectation()

'''def k1(ksi):

return 8 \* ksi

def k2(ksi):

return 1.4 \* ksi

def k3(ksi):

return 0.07 \* ksi

def k4(ksi):

return 0.001

K = [k1, k2, k3, k4]

fp = Solver\_BFP2(K, xmax=50, m=1, s=0.1, n=20, p=18)'''

'''while not fp.check(acc=0.01):

print(seed, end='\n\n')

seed += 1

np.random.seed(seed)

fp = Solver\_MFP(K, xmax=30, m=1, s=0.1, n=20, p=18)

print(f"seed = {seed}", end='\n\n')'''

fp.draw\_middle\_section()

fp.draw\_model()

fp.draw\_PC()

fp.draw\_combined()

print("Завершено!", end='\n\n')

deterministic.py:

import numpy as np

import matplotlib.pyplot as plt

m = 1

c = 0.8

k = 1.16

f = 0.9881

p = 0.5

x0 = 1

x1 = 0

if c \* c >= 4 \* k \* m:

raise ValueError("Квадрат коэффициента затухания колебаний не должен превышать или быть равным произведению массы тела и жёсткости пружины, умноженному на 4")

w = np.sqrt(4 \* k \* m - c \* c) / (2 \* m)

C1 = f \* (k - m \* p \* p) / (c \* c \* p \* p + (k - m \* p \* p) \*\* 2)

C2 = c \* f \* p / (c \* c \* p \* p + (k - m \* p \* p) \*\* 2)

A = x0 - C1

B = (x1 + c / (2 \* m) \* A - C2 \* p) / w

def f(t):

return np.exp(-c / (2 \* m) \* t) \* (A \* np.cos(w \* t) + B \* np.sin(w \* t)) + C1 \* np.cos(p \* t) + C2 \* np.sin(p \* t)

def g(t):

return -c / (2 \* m) \* np.exp(-c / (2 \* m) \* t) \* (A \* np.cos(w \* t) + B \* np.sin(w \* t)) + np.exp(-c / (2 \* m) \* t) \* w \* (B \* np.cos(w \* t) - A \* np.sin(w \* t)) + p \* (C2 \* np.cos(p \* t) - C1 \* np.sin(p \* t))

plt.grid(True)

T = np.linspace(0, 30, 1024)

plt.plot(T, f(T), color="navy", label="Перемещение x(t)")

plt.plot(T, g(T), color = "red", label="Скорость x'(t)")

plt.legend(loc='best')

plt.show()

main.py:

import time

import chaospy

import warnings

import numpy as np

import seaborn as sns

from matplotlib import pyplot as plt

from scipy.integrate import solve\_ivp

from sklearn.model\_selection import train\_test\_split

import metrics

import oscillator

sns.set\_context('notebook', font\_scale=1.2)

sns.set\_style('ticks')

sns.set\_palette('bright')

plt.style.use("https://raw.githubusercontent.com/NeuromatchAcademy/content-creation/main/nma.mplstyle")

warnings.filterwarnings("ignore", category=UserWarning)

def example1(\*\*kwargs):

"""

Example 1.1: Damped linear oscillator model. PC

expansion using the least squares method.

"""

# Define default values

default\_values = {'train\_size': 750, 'test\_size':

250, 'mu': 0.8, 'sigma': 0.2, 't\_min': 0,

't\_max': 30, 'coord\_num': 1000, 'max\_order': 2,

'plot': True, 'quantiles': True}

# Define and initialize local variables

for key, value in default\_values.items():

if key not in kwargs:

kwargs[key] = default\_values[key]

# Define the damped linear oscillator model with default settings

model = oscillator.DampedOscillator()

model.solve()

# Define the distribution of the damping factor

c\_dist = chaospy.Normal(kwargs['mu'], kwargs['sigma'])

# Sample from the distribution using latin hypercube experimental design

c\_sample = c\_dist.sample(kwargs['train\_size'] +

kwargs['test\_size'],

rule='latin\_hypercube')

# Define the time point sequence

coordinates = np.linspace(kwargs['t\_min'],

kwargs['t\_max'],

kwargs['coord\_num'])

# Evaluate the model at the experimental design points

model\_eval = np.array([model.compute\_displacement(p,

coordinates) for p in c\_sample])

# Create train and test sets

train\_sample, test\_sample, model\_train\_eval, model\_test\_eval = train\_test\_split(c\_sample,

model\_eval,

test\_size=kwargs['test\_size'],

train\_size=kwargs['train\_size'])

# Define the PC basis

chaos\_basis = chaospy.generate\_expansion(

kwargs['max\_order'], c\_dist,

normed=True)

# Define the PC metamodel

metamodel = chaospy.fit\_regression(chaos\_basis,

train\_sample.reshape(1, -1),

model\_train\_eval)

# Evaluate the metamodel at the testing points

metamodel\_test\_eval = metamodel(test\_sample).T

# Make comparative plots

if kwargs['plot']:

fig, ax = plt.subplots(nrows=3, ncols=1,

figsize=(10, 13))

ax[0].plot(coordinates, model\_test\_eval.T,

alpha=0.01, color='red')

ax[0].set\_xlabel('t')

ax[0].set\_ylabel('$\\mathcal{M}(t, c)$')

ax[0].set\_title('Оценки модели при случайном параметре $c$')

ax[1].plot(coordinates, metamodel\_test\_eval.T,

alpha=0.01, color='blue')

ax[1].set\_xlabel('t')

ax[1].set\_ylabel('$\\mathcal{M}\_{PC}(t, c)$')

ax[1].set\_title('Оценки метамодели ПХ при случайном параметре $c$')

ax[2].plot(coordinates, model\_test\_eval.T,

alpha=0.01, color='red')

ax[2].plot(coordinates, metamodel\_test\_eval.T,

alpha=0.01, color='blue')

ax[2].set\_xlabel('t')

ax[2].set\_ylabel('$y$')

ax[2].set\_title('Совмещённые оценки')

plt.savefig('results/Oscillator evaluation.jpg',

dpi=500)

# Compute the errors

e\_loo = metrics.leave\_one\_out\_error(test\_sample,

chaos\_basis, model\_test\_eval,

metamodel\_test\_eval)

r\_sqr = metrics.r\_squared(model\_test\_eval,

metamodel\_test\_eval)

e\_mae = metrics.mean\_absolute\_error(model\_test\_eval,

metamodel\_test\_eval)

e\_rmae = metrics.relative\_mean\_absolute\_error(

model\_test\_eval, metamodel\_test\_eval)

# Make comparative plots for the errors

if kwargs['plot']:

fig, ax = plt.subplots(2, 2, figsize=(10, 7))

ax[0, 0].plot(coordinates, e\_loo, linewidth=2,

color='forestgreen')

ax[0, 0].set\_xlabel('t')

ax[0, 0].set\_ylabel('$E\_{LOO}$')

ax[0, 0].set\_title('Ошибка leave-one-out')

ax[0, 1].plot(coordinates, r\_sqr, linewidth=2,

color='forestgreen')

ax[0, 1].set\_xlabel('t')

ax[0, 1].set\_ylabel('$R^2$')

ax[0, 1].set\_title('Коэффициент детерминации')

ax[0, 1].set\_yticklabels(

[0.8, 0.85, 0.9, 0.95, 1.0])

ax[1, 0].plot(coordinates, e\_mae, linewidth=2,

color='forestgreen')

ax[1, 0].set\_xlabel('t')

ax[1, 0].set\_ylabel('$MAE$')

ax[1, 0].set\_title('Средняя абсолютная ошибка')

ax[1, 1].plot(coordinates, e\_rmae, linewidth=2,

color='forestgreen')

ax[1, 1].set\_xlabel('t')

ax[1, 1].set\_ylabel('$rMAE$')

ax[1, 1].set\_title('Относительная средняя абсолютная ошибка')

plt.savefig('results/Oscillator errors.jpg',

dpi=500)

# Compute the mean and the standard deviation of the metamodel output

metamodel\_eval\_mean = chaospy.E(metamodel, c\_dist)

metamodel\_eval\_std = chaospy.Std(metamodel, c\_dist)

# Plot the mean and the standard deviation in a demonstrative form

if kwargs['plot']:

fig, ax = plt.subplots(figsize=(10, 7))

ax.fill\_between(coordinates, metamodel\_eval\_mean

- metamodel\_eval\_std,

metamodel\_eval\_mean +

metamodel\_eval\_std,

alpha=0.4,

label='Стандартное отклонение')

ax.plot(coordinates, metamodel\_eval\_mean,

label='Математическое ожидание')

ax.set\_title('Математическое ожидание и стандартное отклонение')

ax.set\_xlabel('t')

ax.set\_ylabel('$\\mathcal{M}\_{PC}(t, c)$')

plt.legend()

plt.savefig('results/Oscillator expectation.jpg',

dpi=500)

# Measure the time for computing the model and metamodel responses

if kwargs['quantiles']:

model\_time = np.array([])

metamodel\_time = np.array([])

for p in c\_dist.sample(1000,

rule='latin\_hypercube'):

start1 = time.time()

model.compute\_displacement(p, coordinates)

finish1 = time.time()

model\_time = np.append(model\_time, finish1 -

start1)

start2 = time.time()

metamodel(p)

finish2 = time.time()

metamodel\_time = np.append(metamodel\_time,

finish2 - start2)

# Estimate the 95% quantiles

with open('results/Oscillator quantiles.txt',

'w') as res\_file:

print(np.quantile(model\_time,

q=0.025).round(10),

np.quantile(model\_time,

q=0.975).round(10), file=res\_file)

print(np.quantile(metamodel\_time,

q=0.025).round(10),

np.quantile(metamodel\_time,

q=0.975).round(10), file=res\_file)

"""

Example 1.2: Damped linear oscillator model. PC

expansion using the Galerkin method.

"""

# Define the mean and the standard deviation of the damping factor

c\_mu = kwargs['mu']

c\_sigma = kwargs['sigma']

# Define the initial condition for the Galerkin method

ic = oscillator.galerkin\_ic(model.y\_0, model.y\_1,

chaos\_basis)

# Compute the expansion coefficients using the Galerkin method

galerkin\_coeffs = solve\_ivp(

fun=oscillator.galerkin\_system,

t\_span=(kwargs['t\_min'],

kwargs['t\_max']),

y0=ic, method='RK23',

t\_eval=coordinates,

args=(c\_dist, c\_mu, c\_sigma,

chaos\_basis, model))['y']

# Define the intrusive metamodel from the result

intrusive\_metamodel = chaospy.sum(chaos\_basis \*

galerkin\_coeffs.T[:, :3], -1)

res = {'model': model, 'chaos\_basis': chaos\_basis,

'metamodel': metamodel, 'intrusive\_metamodel':

intrusive\_metamodel, 'e\_loo': e\_loo,

'r\_sqr': r\_sqr, 'e\_mae': e\_mae,

'e\_rmae': e\_rmae}

return res

np.random.seed(100000)

res = example1()

with open("results/galerkin\_coefficients.txt", "w") as f:

print(res, file=f)

print("Completed!")

oscillator.py:

import chaospy

import sympy as sp

import numpy as np

from typing import Any, Iterable, Union

class DampedOscillator:

def \_\_init\_\_(self, \*\*kwargs):

"""

Creates the damped linear oscillator model.

"""

if len(kwargs) == 0:

kwargs = {'m': 1, 'k': 1.16, 'f': 0.9881,

'omega': 0.5, 'y\_0': 1, 'y\_1': 0}

else:

if not all(map(lambda x: isinstance(x, (int,

float)), kwargs.values())):

raise TypeError("Parameter values of the model must be of the int or float type.")

if any(map(lambda x: np.isnan(x) or

np.isinf(x), kwargs.values())):

raise ValueError("All parameters of the system should be well-defined.")

self.m = kwargs['m']

self.c = sp.Symbol('c', real=True, positive=True)

self.k = kwargs['k']

self.f = kwargs['f']

self.omega = kwargs['omega']

self.y\_0 = kwargs['y\_0']

self.y\_1 = kwargs['y\_1']

self.displacement = None

self.velocity = None

if any(map(lambda x: x <= 0, [self.m, self.k,

self.f, self.omega])):

raise ValueError("Certain parameters of the system should be positive.")

def solve(self) -> tuple:

"""

Solves the damped linear oscillator model.

:return: tuple, real parts of displacement and

velocity

"""

t = sp.Symbol('t', real=True, positive=True)

y = sp.Function('y')

eq = self.m \* sp.diff(y(t), (t, 2)) + self.c \* sp.diff(y(t), t) + self.k \* y(t) - \

self.f \* sp.cos(self.omega \* t)

sol = sp.dsolve(eq, func=y(t),

ics={y(0): self.y\_0,

sp.diff(y(t), t).subs(t, 0): self.y\_1})

self.displacement = sp.re(sol.rhs)

self.velocity = sp.re(sp.diff(self.displacement,

t))

return self.displacement, self.velocity

def lambdify\_displacement(self, c: float = 0.1) -> callable:

"""

Creates a Python function out of the displacement

expression.

:param c: float, value for the undetermined

damping factor

:return: function, for the displacement time

dependence computation

"""

if self.displacement is None:

raise AttributeError("The displacement attribute wasn't found, solve the model first.")

if not isinstance(c, (int, float)):

raise TypeError("The damping factor must be of the int or float type.")

if not(c >= 0): # to catch NaN

raise ValueError("The value of the damping factor should greater than or equal to zero.")

t = sp.Symbol('t', real=True, positive=True)

return sp.lambdify(t,

self.displacement.subs(self.c, c,

real=True), 'numpy')

def lambdify\_velocity(self, c: float = 0.1) -> callable:

"""

Creates a Python function out of the velocity

expression.

:param c: float, value for the undetermined

damping factor

:return: function, for the velocity time

dependence computation

"""

if self.velocity is None:

raise AttributeError("The velocity attribute wasn't found, solve the model first.")

if not isinstance(c, (int, float)):

raise TypeError("The damping factor must be of the int or float type.")

if not(c >= 0): # to catch NaN

raise ValueError("The value of the damping factor should greater than or equal to zero.")

t = sp.Symbol('t', real=True, positive=True)

return sp.lambdify(t, self.velocity.subs(self.c,

c, real=True), 'numpy')

def compute\_displacement(self, c: float = 0.1,

coordinates: Iterable[float]

= None) -> np.ndarray:

"""

Computes the displacement values at the

coordinate points using a certain

value of the damping factor.

:param c: float, damping factor

:param coordinates: Iterable[float], time points

:return: np.ndarray, displacement values at the

coordinates

"""

if self.displacement is None:

raise AttributeError("The displacement attribute wasn't found, solve the model first.")

if not isinstance(c, (int, float)):

raise TypeError("The damping factor must be of the int or float type.")

if not(c >= 0): # to catch NaN

raise ValueError("The value of the damping factor should greater than or equal to zero.")

if isinstance(coordinates, Iterable):

try:

coordinates = np.array(coordinates,

dtype=float)

except BaseException:

raise TypeError("The coordinates must be of the iterable type over floats.")

else:

raise TypeError("The coordinates must be of the iterable type over floats.")

if len(coordinates) == 0:

raise ValueError("The input is empty.")

if not all(map(lambda x: x >= 0, coordinates)):

raise ValueError("The time points should be greater than or equal to zero.")

displ\_func = self.lambdify\_displacement(c)

return displ\_func(coordinates)

def compute\_velocity(self, c: float = 0.1,

coordinates: Iterable[float]

= None) -> np.ndarray:

"""

Computes the velocity values at the coordinate

points using a certain value of the damping

factor.

:param c: float, damping factor

:param coordinates: Iterable[float], time points

:return: np.ndarray, velocity values at the

coordinates

"""

if self.velocity is None:

raise AttributeError("The velocity attribute wasn't found, solve the model first.")

if not isinstance(c, (int, float)):

raise TypeError("The damping factor must be of the int or float type.")

if not(c >= 0): # to catch NaN

raise ValueError("The value of the damping factor should greater than or equal to zero.")

if isinstance(coordinates, Iterable):

try:

coordinates = np.array(coordinates,

dtype=float)

except BaseException:

raise TypeError("The coordinates must be of the iterable type over floats.")

else:

raise TypeError("The coordinates must be of the iterable type over floats.")

if len(coordinates) == 0:

raise ValueError("The input is empty.")

if not all(map(lambda x: x >= 0, coordinates)):

raise ValueError("The time points should be greater than or equal to zero.")

vel\_func = self.lambdify\_velocity(c)

return vel\_func(coordinates)

def galerkin\_ic(y\_0: Union[int, float], v\_0: Union[int,

float], basis: Any) -> np.ndarray:

"""

Computes the initial condition of the variables for

the Galerkin method.

:param y\_0: Union[int, float], displacement initial

value

:param v\_0: Union[int, float], velocity initial value

:param basis: Any, PC basis

:return: np.ndarray, general initial conditions array

"""

if not isinstance(y\_0, (int, float)):

raise TypeError("y\_0 must be of the int ot float type.")

if not isinstance(v\_0, (int, float)):

raise TypeError("v\_0 must be of the int ot float type.")

if np.isnan([y\_0, v\_0]).any() or np.isinf([y\_0,

v\_0]).any():

raise ValueError("The input contains NaN or Inf.")

init\_y = y\_0 \* np.eye(N=1, M=len(basis)).reshape(-1)

init\_v = v\_0 \* np.eye(N=1, M=len(basis)).reshape(-1)

return np.hstack((init\_y, init\_v))

def galerkin\_system(t: Union[int, float], x: np.ndarray,

dist: Any, c\_mu: Union[int, float], c\_sigma:

Union[int, float], basis: Any, osc: DampedOscillator) -> np.ndarray:

"""

Computes the system of ordinary linear explicit first

order differential equations

:param t: Union[int, float], time point

:param x: np.ndarray, variables of the differential

equation system

:param dist: Any, transformed distribution the PC

basis is built upon

:param c\_mu: Union[int, float], mean of the

untransformed initial distribution

:param c\_sigma: Union[int, float], standard

deviation of the untransformed

initial distribution

:param basis: Any, PC basis

:param osc: DampedOscillator, damped oscillator model

:return: np.ndarray, ODE system

"""

if not isinstance(t, (int, float)):

raise TypeError("The time point must be of the int ot float type.")

if not(t >= 0): # to catch NaN

raise ValueError("The time point should greater than or equal to zero.")

if len(x) != 2 \* len(basis):

raise ValueError("The array of variables must be twice longer than the PC basis.")

if not(c\_mu >= 0) or not(c\_sigma >= 0): # to catch NaN

raise ValueError("The distribution parameters must be greater than or equal to zero.")

if not isinstance(osc, DampedOscillator):

raise TypeError("The damped oscillator model should be provided.")

delta = np.eye(len(basis))

y = x[:len(basis)]

v = x[len(basis):]

dydt = v

dvdt = np.zeros(len(basis))

for n in range(len(basis)):

dvdt[n] = (osc.f \* np.cos(osc.omega \* t) \*

delta[0, n] - c\_mu \* v[n] - osc.k \*

y[n]) / osc.m

summa = 0

for i in range(len(basis)):

summa += v[i] \* chaospy.E(basis[1] \* basis[i]

\* basis[n], dist)

dvdt[n] += -c\_sigma / osc.m \* summa

return np.hstack((dydt, dvdt))

metrics.py:

import warnings

import numpy as np

from typing import Any, Iterable, Union

warnings.filterwarnings("ignore", category=RuntimeWarning)

def empirical\_error(y\_true: Iterable[float], y\_pred: Iterable[float]) -> Union[float, np.ndarray]:

"""

Computes the empirical error between the ground truth

values and the predicted ones.

:param y\_true: Iterable[float], ground truth values

:param y\_pred: Iterable[float], predicted values

:return: Union[float, np.ndarray], empirical error

"""

if isinstance(y\_true, Iterable) and isinstance(y\_pred, Iterable):

try:

y\_true = np.array(y\_true, dtype=float)

y\_pred = np.array(y\_pred, dtype=float)

except BaseException:

raise TypeError("The input must be of the iterable type over floats.")

else:

raise TypeError("The input must be of the iterable type over floats.")

if (len(y\_true) == 0) or (len(y\_pred) == 0):

raise ValueError("The input is empty.")

if len(y\_true) != len(y\_pred):

raise ValueError("The inputs must be of the same size.")

if np.isnan(y\_true).any() or np.isnan(y\_pred).any():

raise ValueError("The input contains NaN.")

if np.isinf(y\_true).any() or np.isinf(y\_pred).any():

raise ValueError("The input contains Inf.")

return np.mean((y\_true - y\_pred) \*\* 2, axis=0)

def r\_squared(y\_true: Iterable[float], y\_pred:

Iterable[float]) -> Union[float,

np.ndarray]:

"""

Computes the R^2 regression score.

:param y\_true: Iterable[float], ground truth values

:param y\_pred: Iterable[float], predicted values

:return: Union[float, np.ndarray], R^2 score

"""

if isinstance(y\_true, Iterable) and isinstance(y\_pred, Iterable):

try:

y\_true = np.array(y\_true, dtype=float)

y\_pred = np.array(y\_pred, dtype=float)

except BaseException:

raise TypeError("The input must be of the iterable type over floats.")

else:

raise TypeError("The input must be of the iterable type over floats.")

if (len(y\_true) == 0) or (len(y\_pred) == 0):

raise ValueError("The input is empty.")

if len(y\_true) != len(y\_pred):

raise ValueError("The inputs must be of the same size.")

if np.isnan(y\_true).any() or np.isnan(y\_pred).any():

raise ValueError("The input contains NaN.")

if np.isinf(y\_true).any() or np.isinf(y\_pred).any():

raise ValueError("The input contains Inf.")

if len(y\_true) == 1:

raise RuntimeError("R^2 score is not well-defined with less than two samples.")

e\_emp = empirical\_error(y\_true, y\_pred)

var = np.sum((y\_true - np.mean(y\_true)) \*\* 2, axis=0) / (len(y\_true) - 1)

# If y\_true is constant, then the R^2 score isn't finite: it is either NaN

# (perfect predictions) or -Inf (imperfect predictions). To prevent

# such non-finite numbers, y default these cases are replaced with 1.0

# (perfect predictions) or 0.0 (imperfect predictions) respectively.

res = 1 - e\_emp / var

nan\_res = np.isnan(res)

inf\_res = np.isinf(res)

if np.sum(nan\_res) > 0:

if isinstance(res, (int, float)):

return 1.

else:

res[nan\_res] = 1.

if np.sum(inf\_res) > 0:

if isinstance(res, (int, float)):

return 0.

else:

res[inf\_res] = 0.

return res

def mean\_absolute\_error(y\_true: Iterable[float], y\_pred:

Iterable[float]) -> Union[float, np.ndarray]:

"""

Computes the Mean Absolute Error (MAE) between the

ground truth values and the predicted ones.

:param y\_true: Iterable[float], ground truth values

:param y\_pred: Iterable[float], predicted values

:return: Union[float, np.ndarray], Mean Absolute

Error

"""

if isinstance(y\_true, Iterable) and isinstance(y\_pred, Iterable):

try:

y\_true = np.array(y\_true, dtype=float)

y\_pred = np.array(y\_pred, dtype=float)

except BaseException:

raise TypeError("The input must be of the iterable type over floats.")

else:

raise TypeError("The input must be of the iterable type over floats.")

if (len(y\_true) == 0) or (len(y\_pred) == 0):

raise ValueError("The input is empty.")

if len(y\_true) != len(y\_pred):

raise ValueError("The inputs must be of the same size.")

if np.isnan(y\_true).any() or np.isnan(y\_pred).any():

raise ValueError("The input contains NaN.")

if np.isinf(y\_true).any() or np.isinf(y\_pred).any():

raise ValueError("The input contains Inf.")

return np.mean(np.abs(y\_true - y\_pred), axis=0)

def relative\_mean\_absolute\_error(y\_true: Iterable[float],

y\_pred: Iterable[float]) -> Union[float, np.ndarray]:

"""

Computes the relative Mean Absolute Error (rMAE)

between the ground truth values and the predicted

ones.

:param y\_true: Iterable[float], ground truth values

:param y\_pred: Iterable[float], predicted values

:return: Union[float, np.ndarray], relative Mean

Absolute Error

"""

if isinstance(y\_true, Iterable) and isinstance(y\_pred, Iterable):

try:

y\_true = np.array(y\_true, dtype=float)

y\_pred = np.array(y\_pred, dtype=float)

except BaseException:

raise TypeError("The input must be of the iterable type over floats.")

else:

raise TypeError("The input must be of the iterable type over floats.")

if (len(y\_true) == 0) or (len(y\_pred) == 0):

raise ValueError("The input is empty.")

if len(y\_true) != len(y\_pred):

raise ValueError("The inputs must be of the same size.")

if np.isnan(y\_true).any() or np.isnan(y\_pred).any():

raise ValueError("The input contains NaN.")

if np.isinf(y\_true).any() or np.isinf(y\_pred).any():

raise ValueError("The input contains Inf.")

if np.isclose(y\_true, 0.).any():

raise ValueError("rMAE is not well-defined when any ground truth value equals zero.")

return np.mean(np.abs(1 - y\_pred/y\_true), axis=0)

def leave\_one\_out\_error(sample: Iterable[float],

basis: Any,

y\_true: Iterable[float],

y\_pred: Iterable[float]) -> Union[float, np.ndarray]:

"""

Computes the leave-one-out error between the ground

truth values and the predicted ones using an

information matrix of the PC expansion.

:param sample: Iterable[float], experimental design

samples

:param basis: Any, PC basis

:param y\_true: Iterable[float], ground truth values

:param y\_pred: Iterable[float], predicted values

:return: Union[float, np.ndarray], leave-one-out

error

"""

if isinstance(y\_true, Iterable) and isinstance(y\_pred, Iterable) and\

isinstance(sample, Iterable):

try:

sample = np.array(sample, dtype=float)

y\_true = np.array(y\_true, dtype=float)

y\_pred = np.array(y\_pred, dtype=float)

except BaseException:

raise TypeError("The input must be of the iterable type over floats.")

else:

raise TypeError("The input must be of the iterable type over floats.")

if (len(sample) == 0) or (len(basis) == 0) or\

(len(y\_true) == 0) or (len(y\_pred) == 0):

raise ValueError("The input is empty.")

if len(y\_true) != len(y\_pred):

raise ValueError("The inputs must be of the same size.")

if np.isnan(sample).any() or np.isnan(y\_true).any() or np.isnan(y\_pred).any():

raise ValueError("The input contains NaN.")

if np.isinf(sample).any() or np.isinf(y\_true).any() or np.isinf(y\_pred).any():

raise ValueError("The input contains Inf.")

psi\_mat = basis(sample).T

h\_diag = np.diag(psi\_mat @ np.linalg.inv(psi\_mat.T @

psi\_mat) @ psi\_mat.T)

if np.sum(np.isinf(h\_diag) | np.isnan(h\_diag) |

np.isclose(h\_diag, 1.)) > 0:

raise RuntimeError("The diagonal of the regression matrix contains either Inf, NaN or ones.")

return np.mean((y\_true - y\_pred) \*\* 2 /

(1 - h\_diag.reshape(-1, 1)) \*\* 2, axis=0)

def relative\_leave\_one\_out\_error(sample: Iterable[float],

basis: Any, y\_true: Iterable[float], y\_pred:

Iterable[float]) -> Union[float, np.ndarray]:

"""

Computes the relative leave-one-out error between the

ground truth values and the predicted ones using an

information matrix of the PC expansion.

:param sample: Iterable[float], experimental design

samples

:param basis: Any, PC basis

:param y\_true: Iterable[float], ground truth values

:param y\_pred: Iterable[float], predicted values

:return: Union[float, np.ndarray], relative leave

one-out error

"""

if isinstance(y\_true, Iterable) and isinstance(y\_pred, Iterable) and \

isinstance(sample, Iterable):

try:

sample = np.array(sample, dtype=float)

y\_true = np.array(y\_true, dtype=float)

y\_pred = np.array(y\_pred, dtype=float)

except BaseException:

raise TypeError("The input must be of the iterable type over floats.")

else:

raise TypeError("The input must be of the iterable type over floats.")

if (len(sample) == 0) or (len(basis) == 0) or \

(len(y\_true) == 0) or (len(y\_pred) == 0):

raise ValueError("The input is empty.")

if len(y\_true) != len(y\_pred):

raise ValueError("The inputs must be of the same size.")

if np.isnan(sample).any() or np.isnan(y\_true).any() or np.isnan(y\_pred).any():

raise ValueError("The input contains NaN.")

if np.isinf(sample).any() or np.isinf(y\_true).any() or np.isinf(y\_pred).any():

raise ValueError("The input contains Inf.")

if len(y\_true) == 1:

raise RuntimeError("Relative E\_LOO score is not well-defined with less than two samples.")

e\_loo = leave\_one\_out\_error(sample, basis,

y\_true, y\_pred)

var = np.sum((y\_true - np.mean(y\_true)) \*\* 2, axis=0) / (len(y\_true) - 1)

# If y\_true is constant, then the score isn't finite:

# it is either NaN(perfect predictions) or –Inf

# (imperfect predictions). To prevent such non-finite

# numbers, y default these cases are replaced with

# 1.0 (perfect predictions) or 0.0 (imperfect

# predictions) respectively.

res = e\_loo / var

nan\_res = np.isnan(res)

inf\_res = np.isinf(res)

if np.sum(nan\_res) > 0:

if isinstance(res, (int, float)):

return 1.

else:

res[nan\_res] = 1.

if np.sum(inf\_res) > 0:

if isinstance(res, (int, float)):

return 0.

else:

res[inf\_res] = 0.

return res

def q\_squared(sample: Iterable[float], basis: Any,

y\_true: Iterable[float], y\_pred: Iterable[float]) -> Union[float, np.ndarray]:

"""

Computes the Q^2 score.

:param sample: Iterable[float], experimental design

samples

:param basis: Any, PC basis

:param y\_true: Iterable[float], ground truth values

:param y\_pred: Iterable[float], predicted values

:return: Union[float, np.ndarray], Q^2 score

"""

return 1 - relative\_leave\_one\_out\_error(sample,

basis, y\_true, y\_pred

# ПРИЛОЖЕНИЕ Б Графическая часть ВКР

В графическую часть выпускной квалификационной работы входят слайды презентации в количестве 17 шт.

