***Государственное бюджетное образовательное учреждение*** ***высшего образования Московской области  
Университет «Дубна»***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРСКАЯ РАБОТА**

**Тема** Разработка алгоритма оценки адекватности модели объекту исследования в пространстве состояний и его параллельная реализация

**ФИО студента** Соколов Иван Александрович

**Группа** 4012 **Направление подготовки** 09.03.01 – Информатика и вычислительная техника \_\_\_

**Профильная направленность образовательной программы:** Автоматизированные системы обработки информации и управления

**Выпускающая кафедра** Системного анализа и управления

**Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /**ст. пр., Миловидова А. А.**/**

|  |  |
| --- | --- |
| **Консультант(ы)** | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**/** |
| **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**/** |

**Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /**к.ф.-м.н., Стрельцова О. И.**/**

**Выпускная квалификационная работа допущена к защите «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.**

(дата)

**Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** /д.т.н. Черемисина Е. Н./

***г. Дубна, 2017***

***Государственное бюджетное образовательное учреждение*** ***высшего образования Московской области  
Университет «Дубна»***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

**УТВЕРЖДАЮ**

**Заведующий кафедрой**

**\_\_\_\_\_\_\_\_\_ /**проф. Черемисина Е. Н. **/**

(Подпись) (Ф И О )

**«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г.**

**З а д а н и е**

***на бакалаврскую работу***

**Тема** Разработка алгоритма оценки адекватности модели объекту исследования в пространстве состояний и его параллельная реализация

**Утверждена приказом № \_\_\_\_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ФИО студента** Соколов Иван Александрович

**Группа** \_4012 Направление: 09.03.01 - Информатика и вычислительная техника, Профиль обучения: Автоматизированные системы обработки информации и управления

**Выпускающая кафедра** Системного анализа и управления

Дата выдачи задания \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата завершения бакалаврской работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Аннотация**

Цель работы – разработка алгоритма решения задачи оценки адекватности модели объекту исследования в пространстве состояний и его параллельная реализация.

Оценка адекватности модели объекту исследования – одна из основных задач при моделировании различных процессов и явлений: природных, социальных, технологических, научных и т.д.

В работе рассматриваются различные методы оценки адекватности модели объекту исследования. Предлагается постановка задачи и алгоритм решения задачи оценки адекватности модели объекту исследования в пространстве состояний. Результаты работы алгоритма апробируются на примере модели «Хищник-жертва» в *MS Excel*.На примере модели «Хищник-жертва» создана параллельная реализация алгоритма с помощью технологии *OpenMP*. Проведены исследование эффективности параллельной реализации.

***Abstract***

*The purpose of the work is to design and develop an algorithm of the problem solution of the model adequacy assessment to research object in space of states and to realize it in parallel.*

*The model adequacy assessment to the research object is one of the main objectives in modeling various processes and the phenomena: natural, social, technological, scientific, etc.*

*Various methods of model adequacy assessment to the research object are considered in the work. The problem is formulated and an algorithm is offered of the problem solution of model adequacy assessment to research object in the space of states. The results of work of the algorithm are confirmed on the example of the «Predator-Prey» model in MS Excel. On the example of the «Predator-Prey» model a parallel realization of the algorithm is created using OpenMP technology. The effectiveness of the parallel implementation is investigated.*

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc485047617)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7](#_Toc485047618)

[1.1 Понятие адекватности 7](#_Toc485047619)

[1.2 Существующие методы оценки адекватности модели объекту исследования 8](#_Toc485047620)

[1.3 Технологии параллельных вычислений 10](#_Toc485047621)

[1.3.1 Технология *MPI* 10](#_Toc485047622)

[1.3.2 Технология *CUDA* 11](#_Toc485047623)

[1.3.3 Технология *OpenMP* 12](#_Toc485047624)

[1.4 Формальная постановка задачи оценки адекватности 13](#_Toc485047625)

[1.5 Вычислительный алгоритм 16](#_Toc485047626)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 18](#_Toc485047627)

[2.1 Реализация алгоритма в среде *MS Excel* 18](#_Toc485047628)

[2.2 Программная реализация алгоритма на *С/С++* 25](#_Toc485047629)

[2.2.1 Последовательная реализация алгоритма 25](#_Toc485047630)

[2.2.2 Параллельная реализация алгоритма 30](#_Toc485047631)

[2.2.3 Проведение вычислений на кластере «*HybriLIT*» 33](#_Toc485047632)

[2.3 Исследование эффективности параллельной реализации алгоритма 35](#_Toc485047633)

[ЗАКЛЮЧЕНИЕ 39](#_Toc485047634)

[СПИСОК ЛИТЕРАТУРЫ 40](#_Toc485047635)

[ПРИЛОЖЕНИЕ 1. Листинг кода параллельной реализации алгоритма 42](#_Toc485047636)

# ВВЕДЕНИЕ

Какой бы сложной не была модель, тем не менее она является приближенным отображение реального объекта и отражает только те свойства, которые нам необходимы. Однако до тех пор, пока не доказана адекватность модели тому объекту, на основе которого она строилась, нельзя с уверенностью утверждать, что с ее помощью получаются те результаты, которые реально отражают состояние и свойства исследуемого объекта. Оценка адекватности и точности модели любого типа, в том числе и имитационной, является важнейшей задачей моделирования, так как любые исследования на неадекватной модели теряют смысл и в некоторых случаях, их применение может привести к убыткам или даже к катастрофам. Например, если управлять реактором атомной электростанции используя неадекватную модель объекта управления.

С ростом адекватности и точности модели возрастает как ее стоимость, так и ценность для исследования, в связи с чем приходится решать вопрос о компромиссе между стоимостью модели и последствиями ошибочных решений из-за ее неадекватности исследуемому процессу или объекту. Поэтому на практике построение модели представляет собой итеративный процесс усовершенствования системы моделей, а, следовательно, и исследования объекта до тех пор, пока это считается разумным. Поэтому и оценка адекватности и точности модели представляет собой непрерывный процесс, начинающийся с началом исследования.

Существует множество алгоритмов оценки адекватности модели объекту исследования, но они требуют значительного объема ресурсов. Но современные информационные технологии не стоят на месте, ведутся разработки в области параллельных вычислений, которые позволяют в значительной степени ускорить расчеты.

На данный момент основными технологиями параллельных вычислений являются:

* *OpenMP*;
* *MPI*;
* *CUDA*.

Каждая из технологий обладает своими особенностями и недостатками и будут рассмотрены в данной работе.

Таким образом, **целью** данной работы является разработка алгоритма оценки адекватности модели объекту исследования в пространстве состояний, его апробация и параллельная реализация.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

* разработка алгоритма оценки адекватности модели объекту исследования в пространстве его состояний;
* апробация работы алгоритма на примере модели «Хищник-жертва» в *MS Excel*;
* реализация последовательной версии алгоритма на примере модели «Хищник-жертва» на языке *C/С++;*
* реализация параллельной версии алгоритма на примере модели «Хищник-жертва» на языке *C/С++;*
* проведение вычислений на кластере «*HybriLIT*»;
* исследование эффективности параллельной реализации алгоритма.

**Исходные данные**:

* наблюдения за состоянием численности популяцией «хищников» и «жертв» на временном интервале;
* научные публикации по теме работы;
* документация по технологии *OpenMP*.

**Результат:** алгоритм оценки адекватности модели объекту исследования в пространстве состояний и его параллельная реализация.

**Критерий оценки результата:** после применения алгоритма будет найден вектор (а) параметров модели с мерами различия;повышение скорости работы алгоритма за счет применения технологии *OpenMP* в 10 раз.

**Используемые методы и средства:** *MS Excel, С/C++* , программа для создания [двух-](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D1%83%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) и [трёхмерных](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D1%91%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) [графиков](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D0%BA_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8) *Gnuplot,* технологияпараллельных вычислений *OpenMP,* вычислительный кластер ОИЯИ«*HybriLIT*»*.*

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Понятие адекватности

Рассмотрим различные толкования понятия «адекватность» с целью выявления его смыслового значения.

Адекватное (лат. *Adaequatus –* приравненный, равный) – вполне соответствующее, соразмерное, согласующееся, верное, точное, тождественное [1].

В теории познания соответствие, сходство идеального образа и объекта.

Адекватность – соответствие, равенство, эквивалентность. Синонимами «адекватность» являются термины: соответствие, идентичность, тождественность, нормальность, соразмерность, равноценность, сообразность, «валидность» и т.д.

Из всего перечисленного можно обобщить понятие «адекватность» и поставить в соответствие слова: сходство, близость, «валидность».

Несмотря на то, что значение слова «валидность» – это мера соответствия методик и результатов исследования, его можно рассматривать как синоним сходства.

Противоположным слову «адекватность» можно поставить: различие, не соответствие.

С точки зрения формального определения «сходства» – мера сходства, (коэффициент близости) существуют различные подходы, сводящиеся по существу к следующему.

Меру сходства определяют, как безразмерную величину для количественного определения степени похожести (сходства) объектов, либо групп объектов. Так в биологии рассматриваются различные типы нормированных [0,1] коэффициентов сходства. Основанием типов сходства является число рассматриваемых объектов [2]:

* унарные – рассматривается один объект. В эту группу входят [меры разнообразия](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D0%B0_%D1%80%D0%B0%D0%B7%D0%BD%D0%BE%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%B8%D1%8F), меры концентрации;
* бинарные – рассматривается два объекта. Это наиболее известная группа – коэффициентов;
* *n*-арные (многоместные) – рассматривается *n* объектов.

В литературе рассматриваются различные количественные коэффициенты сходства (близости), из которых следует: коэффициент сходства как математический объект характеризуется определёнными свойствами, такими, например, как, шкалой измерения, возможностью сравнения одного коэффициента близости с другим. С точки зрения математического аппарата меры близости рассматриваются: в статистике, в кластерном анализе [3], изучаются в векторных пространствах.

## Существующие методы оценки адекватности модели объекту исследования

Модель считается адекватной, если отражает заданные свойства объекта с приемлемой точностью. Точность определяется как степень совпадения значений выходных величин модели и объекта. В настоящее время применяются следующие способы определения адекватности:

1. экспертный метод;
2. экспериментальный метод;
3. результат расчета на основе корреляционных зависимостей;
4. результат определения с помощью общих теорий на основе принятых допущений и аксиом (сравнение с данными «хорошо» себя зарекомендовавшей, «похожей» модели).

Экспертный метод – назначение качественных, сравнительно-количественных или количественных показателей на основе их принятия или соглашения. Часто применяется для математических моделей, «имитирующих» поведение системы путем показа заранее подготовленных данных. Часто оценивается экспертами, имеющими опыт работы с моделируемым объектом [4].

Например, в работе Бориско С.Н., Королев М.Р., Лобейко В.И. «Оценка адекватности имитационного моделирования на основе элементов теории нечетких множеств» [5], авторы используют экспертные методы оценки адекватности для описания общего алгоритма оценки адекватности модели на основе нечетких множеств. Также в работе [6] авторы применяют данный метод для оценки адекватности тренажера оператора эрготехнических радиоэлектронных средств.

Достоинства метода: не требует наличия метрического пакета, нет необходимости в проведении замеров (эксперимента), статистической обработки и т. д.

Недостатки метода: субъективность оценки, значительная зависимость точности оценки от компетенции экспертов, большой риск обнаружения ошибок во время эксплуатации тренажера.

Экспериментальный метод – результат единичного непосредственного измерения или результат статистической обработки нескольких измерений. Хорошо подходит для математических моделей, построенных на имеющихся статистических данных, и для тех случаев, когда можно провести реальный эксперимент [4].

В работе [7] используется экспериментальный метод для оценки адекватности работы плавильной печи. В диссертации [8] изложен экспериментальный подход к проверке адекватности разработанной математической модели динамики автомобиля на стенде с беговыми барабанами во всем скоростном диапазоне исследований.

Достоинства метода: объективность оценки.

Недостатки метода: значительная зависимость от достоверности исходных данных и адекватности метода статистической обработки, высокая сложность при многофакторном анализе.

Результат расчета на основе корреляционных зависимостей, при наличии «очень похожих» моделей и их аппроксимаций.

Результат определения с помощью общих теорий на основе принятых допущений и аксиом хорошо подходит для математических моделей, построенных на основе широко известных законов (закон Ома, Кирхгофа, Ньютона и т.д.).

Достоинства метода: объективность оценки.

Недостатки метода: значительная зависимость от достоверности теоретического аппарата, высокая сложность при многофакторном анализе.

Точность модели может изменятся, в зависимости от условий функционирования объекта, обусловленных разными внешними воздействиями (нагрузками, температурой внешней среды и др.) на моделируемый объект [4].

Кроме того, широкое распространение имеют статистические методы:

1. по средним значениям откликов модели и системы;
2. по дисперсиям отклонений откликов модели от среднего значения откликов системы;
3. по максимальному значению относительных отклонений откликов модели от откликов системы [9].

В работе [10] используется статистический метод, основанный на регрессионном анализе, для оценки адекватности имитационной модели транспортного обслуживания строительных объектов. В работе [11] используется метод по средним значениям откликов модели и системы для оценки адекватности экономических моделей.

Общим недостатком методов является субъективность «выборки» данных для сравнения, и, как следствие, отсутствие абсолютной уверенности в правильности работы модели.

Проведя анализ существующих методов было принято решение разработать такой алгоритм оценки адекватности, который позволит учесть собранные данные о состоянии объекта и оценивать адекватность модели объекта при фиксированной ее структуре, меняя только модельные параметры.

## Технологии параллельных вычислений

### Технология *MPI*

Наиболее распространенной технологией программирования для параллельных компьютеров с распределенной памятью в настоящее время является *MPI*. Основным способом взаимодействия параллельных процессов в таких системах является передача сообщений друг другу. Это и отражено в названии данной технологии Message Passing Interface (интерфейс передачи сообщений). Стандарт *MPI* фиксирует интерфейс, который должен соблюдаться как системой программирования на каждой вычислительной платформе, так и пользователем при создании своих программ.

Интерфейс *MPI* поддерживает создание параллельных программ в стиле *MIMD* (*Multiple Instruction Multiple Data*), что подразумевает объединение процессов с различными исходными текстами. Однако писать и отлаживать такие программы очень сложно, поэтому на практике программисты гораздо чаще используют *SPMD*-модель *(Single Program Multiple Data)* параллельного программирования, в рамках которой для всех параллельных процессов используется один и тот же код. В настоящее время все больше и больше реализаций *MPI* поддерживают работу с нитями.

*MPI*-программа – это множество параллельных взаимодействующих процессов. Все процессы порождаются один раз, образуя параллельную часть программы. В ходе выполнения *MPI*-программы порождение дополнительных процессов или уничтожение существующих не допускается. Каждый процесс работает в своем адресном пространстве, никаких общих переменных или данных в *MPI* нет. Основным способом взаимодействия между процессами является явная посылка сообщений [12].

*MPI*-программа часто существенно отличается от программы последовательной, а в некоторых случаях даже не может выполняться в однопроцессном варианте. И конечно, создание и отладка *MP*I-программ требует значительно больших усилий, чем создание последовательной программы, решающей ту же задачу.

В настоящее время *MPI* – наиболее широко используемый и динамично развивающийся интерфейс из своего класса. Используется в основном для создания мобильных библиотек для [кластеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2)) и [суперкомпьютеров](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80). К положительным моментам *MPI* можно отнести:

* Переносимость – *MPI* позволяет в значительной степени снизить остроту проблемы переносимости параллельных программ между разными компьютерными системами – параллельная программа, разработанная на алгоритмическом языке *C* или *Fortran* с использованием библиотеки *MPI*, как правило, будет работать на любых вычислительных платформах, для которых имеется реализация стандарта.
* Повышение эффективности – *MPI* содействует повышению эффективности параллельных вычислений, поскольку в настоящее время практически для каждого типа вычислительных систем существуют реализации стандарта, в максимальной степени учитывающие возможности используемого компьютерного оборудования.

Реализации стандарта *MPI* имеются практически для всех суперкомпьютерных платформ, а также кластеров на основе рабочих станций *UNIX\Linux* и *Windows* [13].

### Технология *CUDA*

Технология *CUDA* появилась в 2006 году и представляет из себя программно-аппаратный комплекс производства компании *Nvidia*, позволяющий эффективно писать программы под графические адаптеры. Программная часть, в свою очередь, содержит в себе всё необходимое для разработки программы: расширения языка *С*, компилятор, *API* для работы с графическими адаптерами и набор библиотек [14].

*CUDA* использует для программирования стандартный язык *C*, что является одним из основных ее преимуществ для разработчиков. Изначально *CUDA* включает библиотеки *BLAS* (базовый пакет программ линейной алгебры) и *FFT* (расчёт преобразований Фурье). Также *CUDA* имеет возможность взаимодействия с графическими *API OpenGL* или *DirectX*, возможность разработки на низком уровне, характеризуется оптимизированным распределением потоков данных между *CPU* и *GPU*. Вычисления *CUDA* выполняются одновременно с графическими, в отличие от аналогичной платформы *AMD*, где для расчетов на *GPU* вообще запускается специальная виртуальная машина. Но такое «сожительство» чревато и возникновением ошибок в случае создания большой нагрузки графическим *API* при одновременной работе *CUDA*– ведь графические операции имеют все же более высокий приоритет. Платформа совместима с 32- и 64-битными операционными системами *Windows XP*, *Windows Vista*, *MacOS X* и различными версиями *Linux* [15].

Трудоёмкость программирования *GPU* при помощи *CUDA* довольно велика. Программы, написанные на *CUDA*, требуют разбиения приложения между несколькими мультипроцессорами подобно *MPI* программированию, но без разделения данных, которые хранятся в общей видеопамяти. И так как *CUDA* программирование для каждого мультипроцессора подобно *OpenMP* программированию, оно требует хорошего понимания организации памяти. Но, конечно же, сложность разработки и переноса на *CUDA* сильно зависит от приложения. Один из немногочисленных недостатков *CUDA* – слабая переносимость. Эта архитектура работает только на видеочипах этой компании, да ещё и не на всех [16].

Вычисления с использованием графических адаптеров показывают максимальную эффективность в задачах, не требующих интенсивного обращения к памяти. Чуть хуже решаются задачи, для которых памяти требуется много, но при этом есть возможность использовать разделяемую память. Меньшее ускорение будет получена, когда требуется доступ по случайному адресу, но есть пере использование данных (используется текстурная память). Совсем плохо будут решаться те задачи, для которых не выполняется ни одно из этих требований. Кроме того, если задача требует большого количества памяти (несколько гигабайт), то, скорее всего, на данном этапе развития технологии *CUDA* её вообще не целесообразно решать при помощи *GPU* [14].

### Технология *OpenMP*

*OpenMP* (*Open Multi-Processing*) – это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью (*SMP*-системах) [17].

Первый стандарт *OpenMP* был разработан в 1997 году как *API*, ориентированный на написание легко переносимых многопоточных приложений. Сначала он был основан на языке *Fortran*, но позднее включил в себя и языки *С* и *С*++.

Интерфейс *OpenMP* стал одной из наиболее популярных технологий параллельного программирования. *OpenMP* успешно используется как при программировании суперкомпьютерных систем с большим количеством процессоров, так и в настольных пользовательских системах.

В *OpenMP* используется модель параллельного выполнения «ветвление-слияние». Программа *OpenMP* начинается как единственная нить выполнения, называемой начальной нитью. Когда нить встречает параллельную конструкцию, она создает новую группу нитей, состоящую из себя и некоторого числа дополнительных нитей, и становится главной в новой группе. Все члены новой группы (включая главную) выполняют код внутри параллельной конструкции. В конце параллельной конструкции имеется неявный барьер. После параллельной конструкции выполнение пользовательского кода продолжает только главная нить. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждая нить первоначального региона становится основным для своей группы нитей. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности [17].

Число нитей в группе, выполняющихся параллельно, можно контролировать несколькими способами. Один из них – использование переменной окружения *OMP\_NUM\_THREADS*. Другой способ, вызов процедуры *omp\_set\_num\_threads* (). Еще один способ использование выражения *num\_threads* в сочетании с директивой *parallel*.

*OpenMP* можно рассматривать как высокоуровневую надстройку над *Pthreads* (или аналогичными библиотеками нитей). Она дает следующие преимущества:

1. возможность быстро распараллелить вычислительные программы с большими параллельными циклами. То есть не создает новая параллельная программа, а просто последовательно добавляются в текст последовательной программы *OpenMP*-директивы;
2. *OpenMP* достаточно гибкий механизм, предоставляющий большие возможности контроля над поведением параллельного приложения;
3. Предполагается, что *OpenMP*-программа на однопроцессорной платформе может быть использована в качестве последовательной программы, т.е. нет необходимости поддерживать последовательную и параллельную версии;
4. одним из достоинств *OpenMP* поддержка так называемых «*orphan*» (оторванных) директив, то есть директивы синхронизации и распределения работы могут не входить непосредственно в лексический контекст параллельной области [17].

На данный момент технология *OpenMP* поддерживается большинством компиляторов языка *С/С++*.

Для распараллеливания алгоритма была выбрана технология параллельных вычислений *OpenMP* [4], поскольку она позволяет быстро разрабатывать параллельные версии программ, с помощью добавления соответствующих директив в последовательный код. Таким образом, отсутствует потребность создавать новую программу, а просто нужно дополнить существующую *OpenMP*-директивами, то есть нет необходимости писать программу ориентируясь на конкретную библиотеку. В дальнейшем для увеличения эффективности расчетов планируется использовать комбинирования описанных технологий совместно, тем самым найти наилучшее решение.

## Формальная постановка задачи оценки адекватности

1. Задано описание конечного класса (*K*) процессов (), .
2. Задано описание класса () моделей процессов (),.
3. Задано описание соответствия между объектом и моделью .
4. Задано описание общего пространства состояния (*U*) объекта и модели.
5. Определено текущее состояние объекта исследования () при определённых условиях *R* для .
6. Задано ( – множество допустимых начальных значений).
7. Определен вектор параметр .
8. *R* определяет допустимые значения параметров в виде системы ограничений



1. При произвольных начальных значениях вектора состояния и вектора параметров имеем множество всевозможных состояний объекта (см. рисунок 1), т.е..

Процесс

Рисунок 1. Схема получения различных состояний объекта

1. Определено состояния модели объекта , () при определённых условиях для .
2. Задано (множество допустимых начальных значений).
3. Определён вектор параметров модели .
4. определяет допустимые значения параметров в виде системы ограничений



1. При произвольных начальных значениях вектора состояния и вектора параметра имеем множество всевозможных состояний модели объекта (см. рисунок 2) т.е. ).

Модель   
технологического процесса

Рисунок 2. Схема получения различных состояний модели объекта

Пусть задан некоторый процесс и модель этого процесса. Мера сходства модели *M* объекту *Р* (обозначим как  – это процедура, определяющая численную величину, которая характеризует близость модели объекту. Поскольку объект и модель характеризуется своими состояниями, определим меру сходства следующим образом.

Мера сходства состояния модели *M* и объекта *P*, для фиксированного вектора  – это величина , которая характеризует близость состояния модели *М* объекту *P* (см. рисунок 3).

Вычисление меры сходства между *М* и *Р* 

Рисунок 3. Схема вычисления меры сходства объекта и модели

Согласно сказанному для определения меры сходства необходимо привести реальное время, на котором задано состояние объекта к модельному.

Для множества строится сетка: *j*-ый узел сетки .

Для узла определено значение т.е. .

Для узла и начального состояния определено состояние модели .

Так же для определения меры сходства необходимо задать шаг дискретизации времени , *N* – число точек дискретизации.

Найдя меру сходства можно получить меру различия. Данная величина вычисляется, как .

В данной работе модель будет считается адекватной объекту исследования в пространстве состояний, если для заданной некой количественной величины *δ*, мера различия (*M*,*P*) меньше или равна *δ*. Данная величина определяется экспертом.

## Вычислительный алгоритм

1. В пространстве параметров модели определяется матрица квадратичных невязок для всех пар состояния модели и объекта (см. таблица 1).

Таблица 1. Матрица квадратичных невязок

|  |  | … |  | ... |  |
| --- | --- | --- | --- | --- | --- |
|  |  | ... |  | ... |  |
|  |  | ... |  | ... |  |
| ... | ... | ... | ... | ... | ... |
|  |  | .... |  | .... |  |
| ... | ... | ... | ... | ... | ... |
|  |  | ... |  | ... |  |

1. В пространстве параметров модели определяется вектор минимальных квадратичных невязок (см. таблица 2).

Таблица 2. Определение вектора минимальных квадратичных невязок

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | *...* |  | *...* |  |
|  |  | *...* |  | *...* |  |
| *…* | *...* | *…* | *...* | *...* | *...* |
|  |  | *…* |  | *...* |  |
|  |  | *...* |  | *...* |  |

1. В пространстве параметров модели формируется нуль-единичная матрица отклонений квадратичных невязок от минимальных (см. таблица 3).

Таблица 3. Нуль-единичная матрица отклонений квадратичных невязок от минимальных

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *…* |  |
|  |  | *…* |  |
| ... | *…* | *…* | *…* |
|  |  | *…* |  |

1. В пространстве параметров модели формируется вектор коэффициентов близости () и различия () модели и объекта в пространстве состояний (см. таблица 4).

Таблица 4. Формирование векторов близости и различия

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | *...* |  | (0) | (0) |
|  |  | *...* |  |  |  |
| *...* | *...* | *...* | *...* | *...* | *…* |
|  |  | *...* |  |  |  |

1. В векторе коэффициентов различия определяется наилучший, для этого экспертом задается количественная величина *δ*, принимающая значения от 0 до 1. Будем считать, что модель *М* адекватна объекту *P,* если найдётся такое значение вектора , для которого выполняется условие .

Если возникает ситуация, что не найдено ни одной меры различия удовлетворяющей , то алгоритм необходимо уточнить за счет уменьшения сетки области параметров и /или шага дискретизации интервала времени.

Данный алгоритм применим для объекта в том случае, если в пространстве параметров присутствует оптимальная точка, при котором модель будет близка по состоянии объекту исследования.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Реализация алгоритма в среде *MS Excel*

Рассмотрим в качестве объекта исследования процесс взаимодействия двух видов, находящиеся в определённом отношении между собой. Например, в качестве отношения выберем «хищничество» (один виде поедает другой). Пусть в течение определённого времени за видами, при известных условиях (*U*), осуществлялось наблюдение. Пространство, в котором наблюдаются виды, характеризуется двумя переменными – численностью «жертв» и численностью «хищников». Пусть наблюдения проводились в моменты времени , ; при , . Все *n* наблюдений сведены в следующую таблицу 5.

Таблица 5. Наблюдения за объектом

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Номер наблюдения | Момент наблюдения | Численность «жертв» | Численность «хищников» | Прирост «жертв» | Число погибших «жертв» | Прирост «хищников» | Число умерших «хищников» |
| 1 | *t*1 | *y*11 | *z*11 |  |  |  |  |
| 2 | *t*2*= t*1*+h* | *y*21 | *z*21 |  |  |  |  |
| … | … | … | … | … | … | … | … |
| *i* | *ti=t*1*+h*(*i-*1) | *yi*1 | *zi*1 |  |  |  |  |
| … | … | … | … | … | … | … | … |
| *n* | *tn=t*1*+h*(*n-*1) | *yn*1 | *zn*1 |  |  |  |  |

Пусть динамика изменения численности видов представлена моделью Вольтера-Лотка:

  (1)

где *a*, *b*, *c*, *d* > 0; .

На основе (1), используя метод Рунге-Кутта 4-го порядка, построим компьютерную модель в *MS Excel*.

На рисунке 4 представлены параметры модели и начальные условия для построения модели, где  – начальное время измерений, *T* – конечное значение времени измерения, *N* – количество точек измерений, *h* – шаг измерений,  – начальное количество «жертв» в популяции,  – начальное количество «хищников» в популяции. Переменные *a, b, c, d* – коэффициенты, отражающие взаимодействия между видами.

|  |  |  |  |
| --- | --- | --- | --- |
| *t*0 | *T* | *N* | *h* |
| 0 | 100 | 1000 | 0,01 |

|  |  |
| --- | --- |
| *y(*0*)* | *z(*0*)* |
| 100 | 20 |

|  |  |  |  |
| --- | --- | --- | --- |
| *a* | *b* | *c* | *d* |
| 10 | 0,1 | 35 | 0,5 |

Рисунок 4. Параметры модели и начальные условия

В таблице 6 представлен фрагмент расчета состояний численности популяций в модели при заданных значениях параметров *a*, *b*, *c*, *d*.

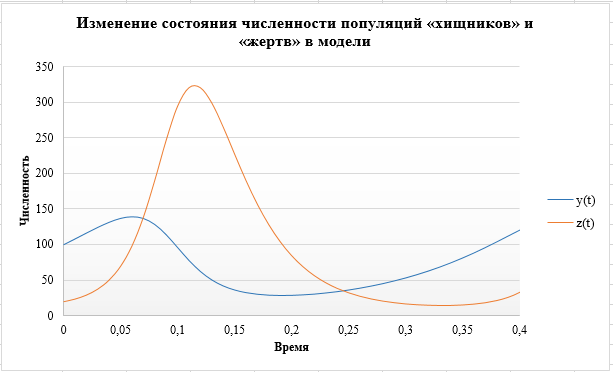
Таблица 6. Фрагмент рассчета состояний численности популяций в модели

| *t* | *yi*(*t*) | *zi*(*t*) |
| --- | --- | --- |
| 0 | 100 | 20 |
| 0,01 | 108,1415636 | 23,71202467 |
| 0,02 | 116,4111433 | 29,29361936 |
| 0,03 | 124,4519789 | 37,70225289 |
| 0,04 | 131,6697992 | 50,4267108 |
| 0,05 | 137,1279393 | 69,64969999 |
| 0,06 | 139,4782251 | 98,1594328 |
| 0,07 | 137,0921434 | 138,4206454 |
| 0,08 | 128,6629336 | 190,0545513 |
| 0,09 | 114,3056397 | 246,3831703 |
| 0,1 | 96,27986142 | 294,1784542 |
| 0,11 | 78,08855145 | 320,3762831 |
| 0,12 | 62,51426048 | 320,3998392 |
| 0,13 | 50,61446425 | 299,1187692 |
| 0,14 | 42,15248739 | 265,4580011 |
| 0,15 | 36,40632719 | 227,4375399 |
| 0,16 | 32,65736243 | 190,3432195 |
| 0,17 | 30,34996724 | 156,9359782 |
| 0,18 | 29,09597113 | 128,2622589 |
| 0,19 | 28,63583887 | 104,3884397 |
| … | … | … |

На рисунке 5 представлен фазовый портрет состояний численности популяций «хищников» и «жертв», построенный на основе данных, полученных из построенной модели.

Рисунок 5. Фазовый портрет состояний численности популяций «хищников» и «жертв» в модели

На рисунке 6 представлены графики изменения состояния численности популяций «жертв» и «хищников» со временем, построенных на основе данных из модели.

Рисунок 6. Изменение состояния численности популяций «хищников» и «жертв» в модели   
на интервале [0, 0.4]

Пусть имеются реальные наблюдения за состоянием численности популяций «хищников» и «жертв» во времени. Фрагмент наблюдений за состоянием численности популяций представлен в таблице 7, а графики состояний и фазовый портрет – на рисунках 7 и 8 соответственно.

Таблица 7. Фрагмент наблюдений за численностью популяций

| *t* | *yi*(*t*) | *zi*(*t*) |
| --- | --- | --- |
| 0 | 100 | 20 |
| 0,01 | 106,1728 | 20,30821 |
| 0,02 | 112,6552 | 21,28454 |
| 0,03 | 119,3708 | 23,05703 |
| 0,04 | 126,2019 | 25,83906 |
| 0,05 | 132,9685 | 29,96076 |
| 0,06 | 139,4011 | 35,91022 |
| 0,07 | 145,1049 | 44,3756 |
| 0,08 | 149,5256 | 56,25721 |
| 0,09 | 151,9365 | 72,57282 |
| 0,1 | 151,4973 | 94,11608 |
| 0,11 | 147,4537 | 120,7249 |
| 0,12 | 139,5045 | 150,2793 |
| 0,13 | 128,1851 | 178,1964 |
| 0,14 | 114,9204 | 198,5597 |
| 0,15 | 101,5394 | 206,8372 |
| 0,16 | 89,56497 | 202,1243 |
| 0,17 | 79,81369 | 187,0411 |
| 0,18 | 72,45225 | 165,8395 |
| 0,19 | 67,26955 | 142,5201 |
| … | … | … |

Рисунок 7. Изменение состояния численности популяций «хищников» и «жертв» объекта исследования на интервале [0, 0.4]

Рисунок 8. Фазовый портрет состояний численности популяций «хищников» и «жертв» объекта исследования

Для оценки адекватности модели «Хищник-жертва» объекту исследования задается *δ*, например, эксперт задал ее равную 0,3. Если найдется хотя бы один вектор параметров модели, при котором коэффициент различия будет меньше *δ*, то можно сказать, что модель адекватна объекту исследования в пространстве состояний с полученной мерой различия , и следовательно, алгоритм работает правильно.

Модельное время приводится ко времени объекта исследования. Для модели определяется множество допустимых векторов параметров. В таблице 8 представлен фрагмент значений этих векторов.

Таблица 8. Фрагмент множества допустимых векторов параметров модели

| *N* | *a* | *b* | *c* | *d* |
| --- | --- | --- | --- | --- |
| 1 | 7 | 0,05 | 45 | 0,3 |
| 2 | 7 | 0,05 | 45 | 0,5 |
| 3 | 7 | 0,05 | 45 | 0,8 |
| 4 | 7 | 0,05 | 50 | 0,3 |
| … | … | … | … | … |
| 81 | 12 | 0,8 | 55 | 0,8 |

Подставляя различные вектора параметров в модель, формируются различные состояния модели. Получив различные состояния модели, к ним применяется алгоритм оценки адекватности модели объекту исследования в пространстве состояний, рассмотренный в теоретической части.

Согласно алгоритму, в конце формируется вектор коэффициентов различия. Фрагмент расчетов показан в таблице 9.

Таблица 9. Фрагмент расчета вектора коэффициентов различия

| *Ai* | *Y* | *Z* |  |
| --- | --- | --- | --- |
| *A*1 | 1 | 0 | 0,5 |
| *A*2 | 0 | 0 | 1 |
| *A*3 | 1 | 1 | 0 |
| *A*4 | 0 | 0 | 1 |
| *A*5 | 0 | 0 | 1 |
| *A*6 | 0 | 0 | 1 |
| *A*7 | 0 | 0 | 1 |
| *A*8 | 0 | 0 | 1 |
| *A*9 | 0 | 0 | 1 |
| *A*10 | 0 | 0 | 1 |
| *A*11 | 0 | 0 | 1 |
| *A*12 | 0 | 0 | 1 |
| *A*13 | 0 | 0 | 1 |
| *A*14 | 0 | 0 | 1 |
| *A*15 | 0 | 0 | 1 |
| *A*16 | 0 | 0 | 1 |
| *A*17 | 0 | 0 | 1 |
| *A*18 | 0 | 0 | 1 |
| *A*19 | 0 | 0 | 1 |
| *A*20 | 0 | 0 | 1 |
| *A*21 | 0 | 0 | 1 |
| … | … | … | … |
| *A*81 | 0 | 0 | 1 |

На основе этой таблицы находится наилучший коэффициент различия состояния модели с заданным вектором параметров, то есть самый минимальный. В результате эксперимента определяется вектор параметров модели, при котором состояние модели и состояния объекта исследования наиболее близки. Минимальным оказался вектор под номером 3. Он имеет коэффициент различия равный нулю. Полученные результаты представлены в графической форме на рисунке 9, 10.

Рисунок 9. Изменение состояния численности популяций «хищников» и «жертв»   
на интервале [0, 0,4] объекта исследования и модели

Рисунок 10. Фазовый портрет состояний численности популяций «хищников» и «жертв» объекта исследования и модели

По рисункам видно, что фазовый портрет модели и объекта исследования, а также графики изменения состояния численности популяций, совпали. Расчеты и совпадение графиков говорит о том, что, правильно рассчитан и найден коэффициент различия модели и объекта исследования. Следовательно, найден вектор параметров модели, при котором модель наиболее близка по состоянию объекту исследованию и у которого мера различия минимальна среди всех и меньше *δ*. Это говорит о том, что модель адекватна исследуемому объекту в пространстве состояний и алгоритм отработал правильно.

## Программная реализация алгоритма на *С/С++*

### Последовательная реализация алгоритма

Если объект исследования будет содержать много переменных, характеризующих его состояние, а модель множество векторов с параметрами, то работа алгоритма может занять значительное время. Поэтому необходимо провести ускорение расчетов с помощью параллельных вычислений.

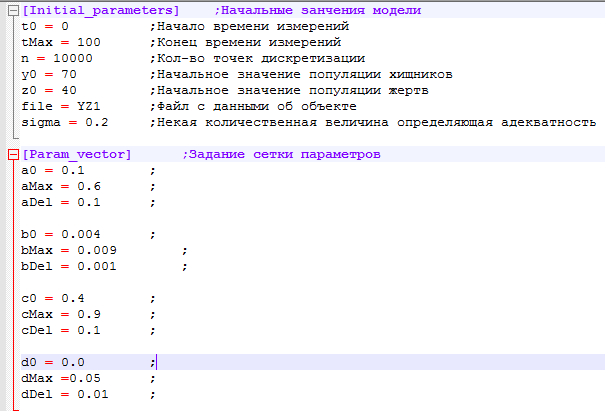
Отметим, что особенностью данного алгоритма является то, что он основан на независимом вычислении в пространстве параметров, уравнений, описывающих рассматриваемую модель, что в свою очередь, позволяет их находить параллельно.

*MS Excel* не обладает хорошей точностью расчетов и допускает слишком большую погрешность. Это было выявлено после сравнения результатов вычислений, полученных из *MS Excel*, с расчётами, представленными в книге Эдвардс Ч., Пенни Д. «Дифференциальные уравнения и краевые задачи» [5, с. 549]. Поэтому в дальнейшем было принято решение использовать для расчетов язык *С/С++*, что позволит повысить точность вычислений, а также дает возможность применить технологии параллельных вычислений. Так как описанные в теоретической части технологии поддерживают написание на этом языке программ.

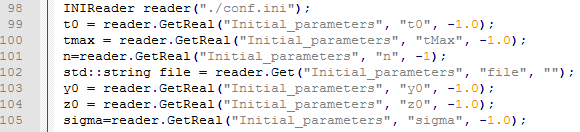
Для рассматриваемой модели «Хищник-жертва» была написана последовательная версия реализации алгоритма на языке *С/C++*. Листинг всей программы представлен в приложении 1.

В программе предусмотрены три функции: *rk*4, *fy*, *fz*. Функция *rk*4осуществляет расчет состояний модели на каждом временном промежутке с заданным вектором параметров. Функция *fy* описывает уравнения изменения численности состояния популяции «хищников», а функция *fz,* соответственно, изменения численности состояния популяции «жертв». Все остальные вычисления и реализация алгоритма осуществляются в методе *main*.

Передача начальных данных и всех параметров осуществляется с помощью конфигурационного файла *conf.ini* (*ini*-файл – *Initialization file*). Файлы конфигурации могут использоваться в любой операционной системе. Несложная структура этого формата позволяет легко обрабатывать их программно и имеет достаточно понятный вид для чтения и изменения пользователем. Содержание данного файла представлено на рисунке 11.

Рисунок 11. Конфигурационный файл conf.ini

Для того, чтобы считать данные из файла *conf.ini* в программу была подключена библиотека *INIReader.h*.Данная библиотека была написана для парсинга данных различных типов из файлов, имеющих формат как у *conf.ini*. В коде сначала инициализируется переменная типа *INIReader*, у которой указывается путь к конфигурационному файлу. После этого вызывая метод *GetReal* и *Get,* переменной *INIReader,* осуществляется чтение значений переменных из файла*.* Код чтения начальных данных модели из файла *conf.ini* показано на рисунке 12.

Рисунок 12. Чтение начальных данных модели из файла conf.ini

Считав значения всех параметров из конфигурационного файла, производится расчет количества векторов параметров модели и формирование данных векторов параметров (см. рисунок 13).



Рисунок 13. Расчет количества векторов параметров модели и формирование   
данных векторов параметров

После того, как были сформированы вектора параметров модели, осуществляется чтение значений состояния объекта исследования. Эти данные хранятся в файле *YZ*. Код чтения представлен на рисунке 14, а фрагмент файла на рисунке 15.

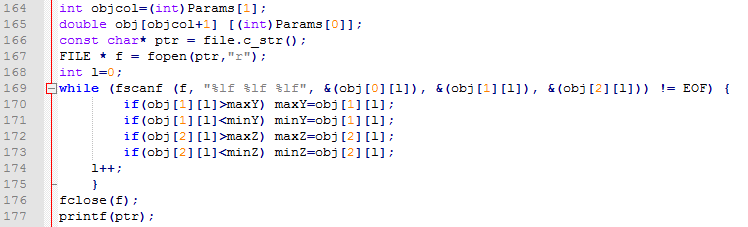
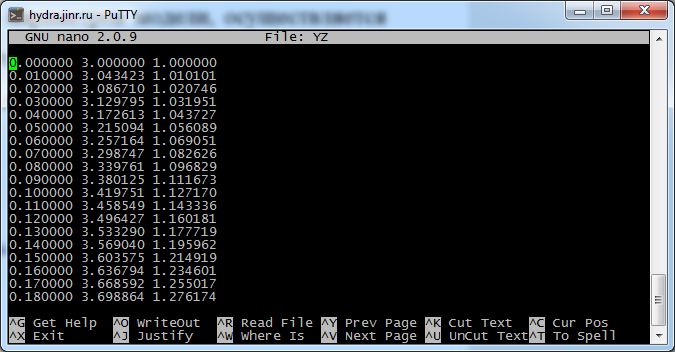


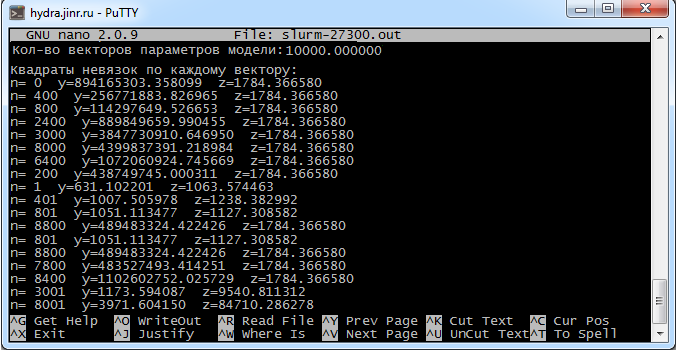
Рисунок 14. Чтение значений состояния объекта исследования из файла *YZ*

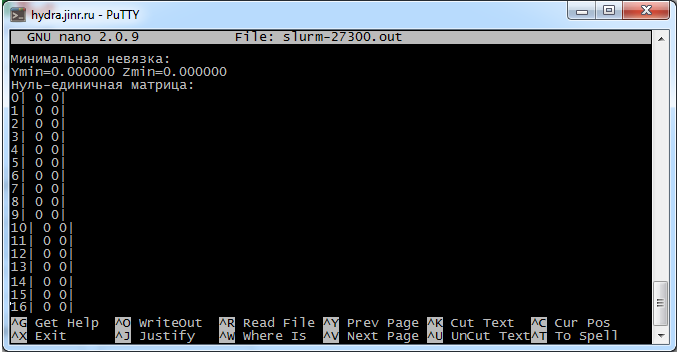
Рисунок 15. Фрагмент файла YZ

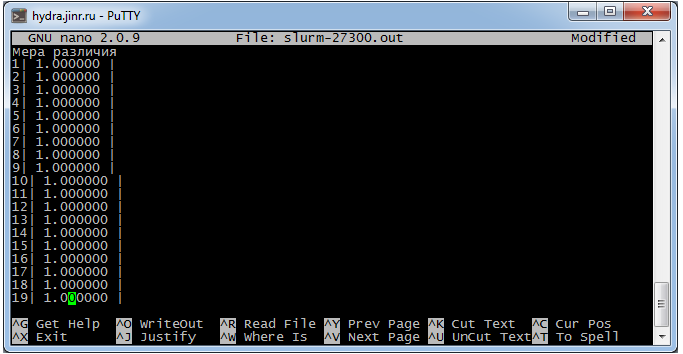
Получив все необходимые данные для построения модели, производятся вычисления алгоритма, описанного в теоретической части.

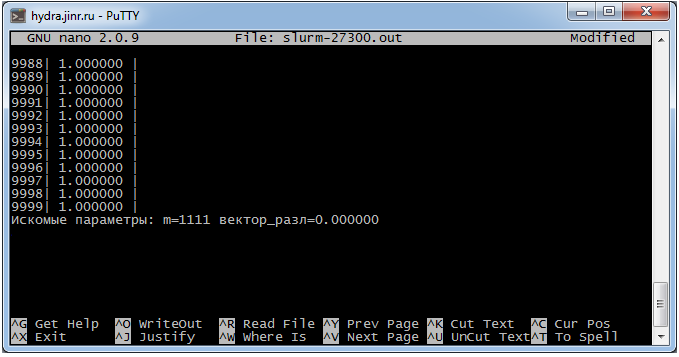
Результатом выполнения программы (*output*) является:

* количество векторов параметров (см. рисунок 16);
* расчет квадратов невязок по каждому вектору (см. рисунок 16);
* минимальная невязка из всех векторов (см. рисунок 17);
* сформированная нуль-единичная матрица (см. рисунок 17);
* мера различия по каждому вектору (см. рисунок 18);
* искомый вектор параметров и его мера различия (см. рисунок 19).

Рисунок 16. Количество векторов параметров модели и фрагмент расчета квадратов невязок

**Рисунок 17. Минимальная невязки и фрагмент нуль-единичной матрицы

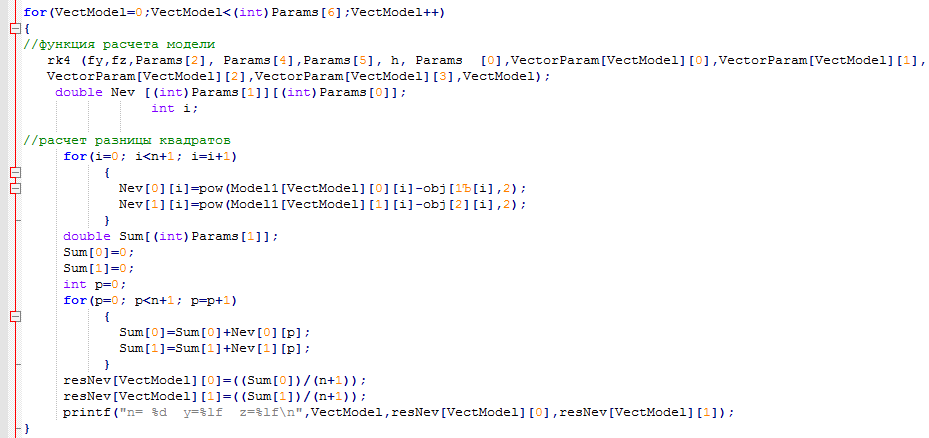
Рисунок 18. Мера различия

Рисунок 19. Искомый вектор параметров и его мера различия

### Параллельная реализация алгоритма

Для ускорения расчетов разработана параллельная реализация алгоритма оценки адекватности модели исследования в пространстве состояний, на основе последовательной версии. Параллельная реализация проводилась с помощью, выбранной в теоретической части технологии *OpenMP*.

Перед применением технологии необходимо определить этапы алгоритма, которые можно распараллелить. Проведенный анализ показал, что вычисления по всем векторам параметров модели могут быть проведены параллельно. Для каждого сформированного вектора параметров осуществляется расчет состояний численности популяций «хищников» и «жертв», а также квадратичных невязок пар состояния модели и объекта исследования. Все эти расчеты организуются в цикле представленном на рисунке 20.

Рисунок 20. Цикл расчета для вектора параметров модели состояний численности популяций «хищников» и «жертв», квадратичных невязок пар состояния модели и объекта исследования

Для использования технологии *OpenMP* необходимо подключить библиотеку *omp.h*, которая содержит все методы, переменные окружения, директивы, процедуры данной технологии.

Распараллеливание описанного выше цикла происходит с помощью двух *OpenMP*-директив: *parallel* и *for*.

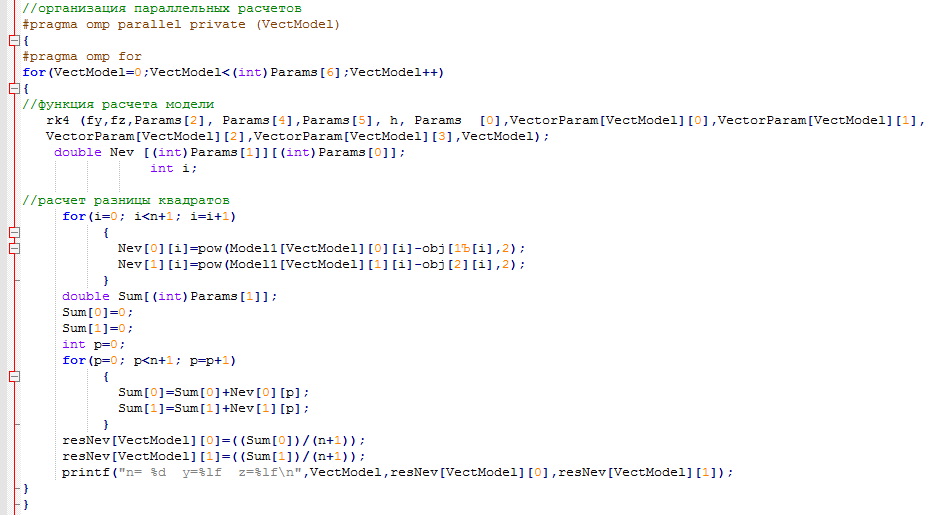
Директива *#pragma omp parallel* определяет параллельную область программы. При входе в эту область порождаются новые (*N*-1), образуется группа из *N* нитей, а порождающая нить получает номер 0 и становится основной нитью команды (т.н. «*master thread*»). *N* это число ядер процессора. *N* можно задавать тремя способами:

* количество нитей определяется до выполнения программы, определив значение переменной среды *OMP\_NUM\_THREADS*;
* задание количества нитей перед параллельной областью в коде программы с помощью вызова функции *omp\_set\_num\_threads* ();
* с помощью опции *NUM\_THREADS*.

Каждая из нитей в группе выполняет следующую за директивой команду (или блок команд, определённый в {}-скобках). При выходе из параллельной области основная нить дожидается завершения остальных нитей и продолжает выполнение в одном экземпляре. То есть применив данную директиву, каждая итерация цикла будет обсчитываться в стольких потоках, сколько их было выделено при компиляции программы. При освобождении нити в ней будет помещена следующая итерация [19].

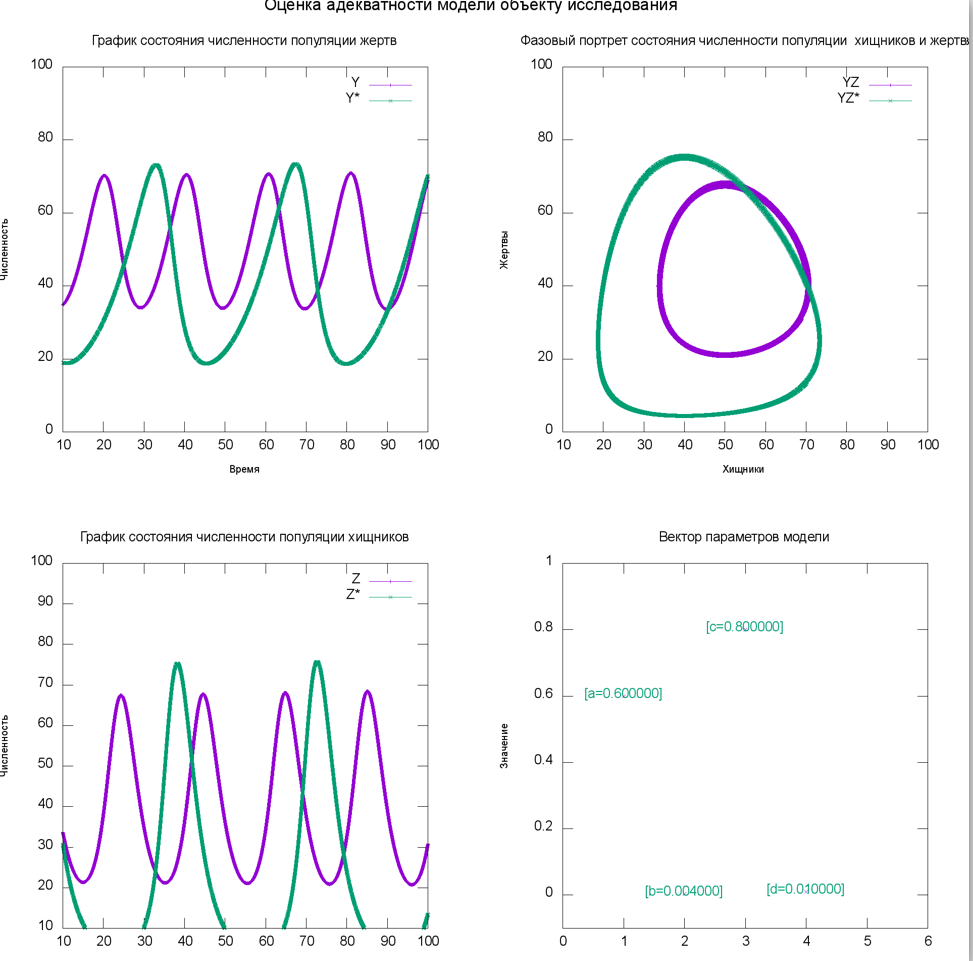
Директива *#pragma omp for* распределяет вычислительную работу цикла *for* на части, и отдаёт каждую часть нити из текущей группы, между всеми нитями. Данная директива не создаёт новых нитей, она всего лишь делит работы между нитями текущей группы [19].

Применение данных директив в коде программы показано на рисунке 21.

Рисунок 21. Применение директив *parallel* и *for* в коде программы

Для расчета времени, которое необходимо для проведения вычислений оценки адекватности модели объекту исследования, в программе используется библиотечная функция *OpenMP*, называющаяся *omp\_get\_wtime* () и возвращающая время в секундах. Данная функция вызывается в самом начале и конце программы.

Для визуального представления результатов расчетов в программу был добавлен код формирования скрипта для программы *Gnuplot* (см. приложение 1). После запуска данного скрипта в программе *Gnuplot* создается файл анимации, в котором осуществляется перебор различных векторов параметров модели, отображение графиков численности популяций объекта исследования и модели при определенном векторе параметров и их фазовый портрет. Один из кадров анимации представлен на рисунке 22.

Рисунок 22. Кадр файла анимации перебора векторов параметров модели

### Проведение вычислений на кластере «*HybriLIT*»

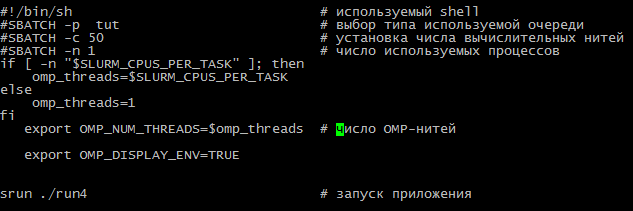
Тестирование проводилось на вычислительных узлах гетерогенного кластера *«HybriLIT*»[20]*.* На каждом из них установлен следующий набор программного обеспечения:

* операционная система *Scientific Linux* 6.8;
* пакет *MODULE* 3.2.10 позволяет пользователю изменять список компиляторов для сборки приложений с поддержкой основных языков программирования (*C/C++*, *FORTRAN*, *Java*), технологий параллельного программирования (*OpenMP*, *MPI*, *OpenCL*, *CUDA*) и использовать установленные на кластере пакеты программ;
* *SLURM* 16.05 – менеджер ресурсов и планировщик заданий для вычислительных кластеров, управляющий доступными вычислительными узлами. Настройки планировщика *SLURM* позволяют управлять очередью для заданий, общей загрузкой ресурсов в процессе выполнения заданий, распределять нагрузку по выделенным узлам.

Каждый узел оснащен двумя вычислительными процессорами ***Intel Xeon***с 12 или 14 ядрами. Один вычислительный узел может предоставить для расчетов 48 или 56 вычислительных ядер. Так как технология *OpenMP* является средством программирования для многопроцессорных систем с общей памятью, то число нитей, которые мы можем организовать при проведении вычислений, определяется возможностями вычислительного узла. Вычисления производились в типе очереди«*tut*», содержащую один вычислительный узел, оснащенный двумя 12-ти ядерными процессорами *Intel Xeon*. Максимальное количество используемых нитей – 48.

Компиляция *OMP* реализации программы осуществляется с помощью компилятора *Intel*. Перед компиляцией с использованием Intel-компиляторов необходимо загрузить соответствующий модуль: *module add hlit/intel/*2017.2.050. Соответственно для компиляции программ на языке *С/C++* необходимо выполнить команду: *icpc -qopenmp ruge\_ex.cpp -o run*, где –*openmp –* ключ компилятора, поддерживающего *OpenMP*; *ruge\_ex.cpp –* код программы*; run –* выходной исполняемый бинарный файл.

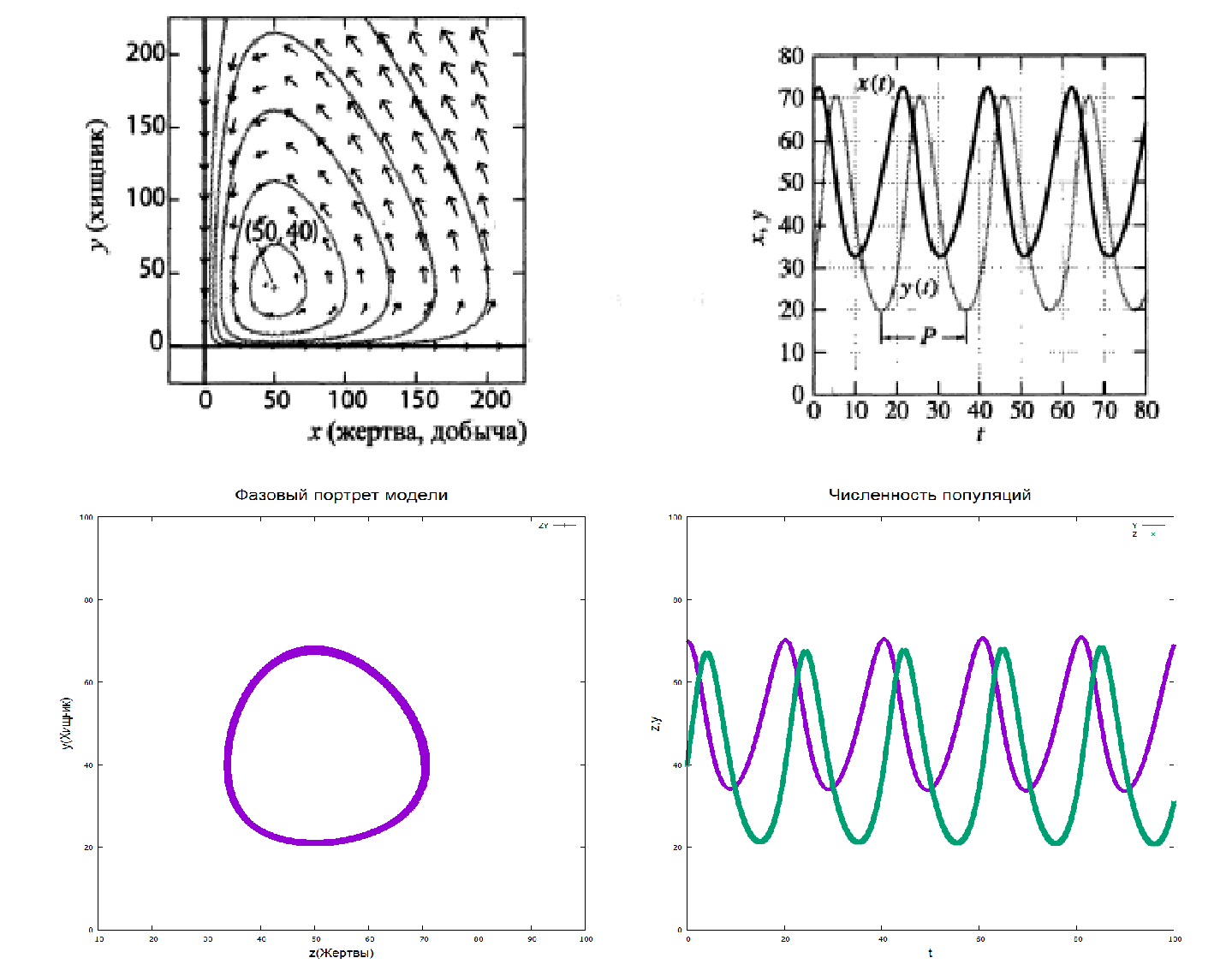
ЗапускOpenMP-приложений осуществляться с помощью *script*-файла, содержащего следующую информацию, представленную на рисунке 23.

Рисунок 23. *Script*-файла для запуска *OpenMP*-приложения на кластере

Для запуска приложения используется следующая команда: *sbatch имя скрипта.* Команда *sbatch* является основной командой планировщика SLURM. После запуска приложению присваивается персональный номер ***jobid*,** по которому его можно найти в списке запущенных приложений. Результат записывается в файл с именем ***slurm-jobid.out*.**

## Исследование эффективности параллельной реализации алгоритма

Для тестирования правильности работы программной реализации били проведены расчеты на задачах, имеющих аналитическое решение, а также было проведено сравнение с результатами, полученными в книге Эдвардс Ч., Пенни Д. «Дифференциальные уравнения и краевые задачи» [5, с. 549] для модели «Хищник-жертва». После проведения расчетов были построены графики фазового портрета и численности состояний популяций. Графики построенные на основе модели и в книге представлены на рисунке 24.

Рисунок 24. Графики фазового портрета и состояния численности популяций из книги и построенные с помощью программы *Gnuplot*

Для определения на сколько был ускорен алгоритм был сформирован следующий конфигурационный файл для модели (см. рисунок 25).

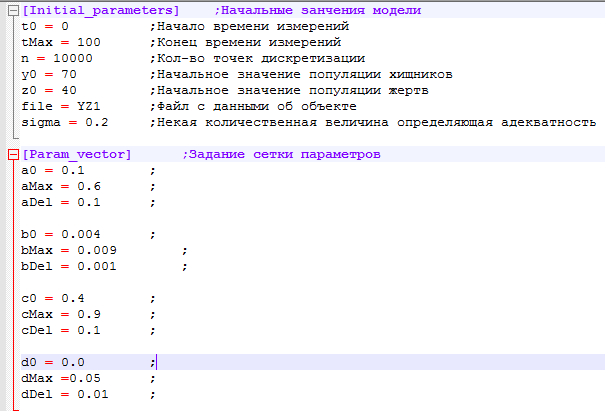


Рисунок 25. Конфигурационный файл для проведения исследования эффективности алгоритма

На основе заданных параметров модели формируется 10000 векторов параметров модели.

Программа определения оценки адекватности модели объекту исследования в пространстве состояний была запущена на разном количестве нитей. Время, затраченное на расчёты оценки в зависимости от количества нитей, представлено в таблице 10.

Таблица 10. Время, затраченное на расчёты оценки в зависимости от количества нитей

|  |  |
| --- | --- |
| *N* (Количество нитей) | *Tn* (Время расчетов) |
| 1 | 7,994464 |
| 5 | 1,730559 |
| 10 | 0,921059 |
| 15 | 0,67147 |
| 20 | 0,582573 |
| 25 | 0,48462 |
| 30 | 0,452749 |
| 35 | 0,407689 |
| 40 | 0,386883 |
| 45 | 0,360766 |
| 48 | 0,353584 |

На основе полученных замеров времени, построен график времени расчетов, ускорения расчетов и эффективности расчетов в зависимости от количества нитей (см. рисунок 26-28). Эффективность рассчитывается как отношение времени рассветов с использованием одной нити к времени расчетов с использованием другого количества нитей. А ускорение как отношение времени рассветов с использованием одной нити к времени расчетов с использованием другого количества нитей умноженного на количество используемых нитей.

Рисунок 26. График времени расчетов в зависимости от количества нитей

Рисунок 27. График ускорения расчетов в зависимости от количества нитей

Рисунок 28. График эффективности расчетов в зависимости от количества нитей

Таким образом, из представленных результатов видно, что в параллельном режиме время расчётов сократилось в 25 раз при использовании двух 12-ти ядерных процессоров *Intel Xeon*. Однако, эффективность параллельных вычислений падает, так как на организацию нитей уходит больше времени, чем на сами вычисления.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской дипломной работы был разработан алгоритм оценки адекватности модели объекту исследования в пространстве состояний. Работа данного алгоритма была проверена на модели «Хищник-жертва». С помощью алгоритма был найден такой вектор параметров модели, при котором модель была наиболее близка объекту исследования в пространстве состояний. При использовании разработанного алгоритма для более сложных моделей, в том числе содержащих большее количество параметров, время вычислений существенно возрастет. Поэтому в работе были рассмотрены различные технологии параллельных вычислений и на основе одной из них разработан параллельный алгоритм для проведения вычислений на вычислительных системах с общей памятью, содержащих современные многоядерные процессоры.

В процессе исследований была изучена технология параллельных вычислений *OpenMP*. C помощью данной технологии была разработана параллельная реализация алгоритма и проведены вычисления на примере модели «Хищник-жертва». Применение данной технологии позволило сократить время вычислений в 25 раз при использовании вычислительных узлов, оснащенных двумя 12-ти ядерными процессорами *Intel Xean*.

В дальнейшем, несмотря на распараллеливание данного алгоритма в связи с многообразием параметров объектов исследования, планируется рассмотреть методы сокращения полного перевода. Так же применение других технологий параллельных вычислений в данном алгоритме. Данный алгоритм планируется к практическому применению для оценки адекватности модели технологического процесса переработки железорудного концентрата ОАО «Стойленского горно-обогатительного комбината» и других сложных технологических объектов в рамках внедрения интеллектуальных систем принятия решений в управлении производством.

# СПИСОК ЛИТЕРАТУРЫ

1. Адекватность: большая советская энциклопедия / Под ред. О.Ю. Шмидт. – М.: Советская энциклопедия, 1969–1978. – C.821.
2. [Magurran A.E.](https://en.wikipedia.org/wiki/Anne_E._Magurran) Measuring biological diversity. – Oxford, UK.: Blackwell Publishing, 2004. – 256 p.
3. Коврижных О.Е., Синельник С.А. Определение меры сходства объектов в кластерном анализе [Электронный ресурс] – Электрон. текст. – 2015. – Режим доступа: http://www.konspekt.biz/index.php?text=51064, свободный (дата обращения – 01.03.2017).
4. Гаммер М. Адекватность и универсальность математической модели [Электронный ресурс] – Электрон. текст. – 2012. – Режим доступа: http://www.professionalgroup.ru/resheniya/ispolzuemyie-texnologii/adekvatnost-i-universalnost-matematicheskoj-modeli.html, свободный (дата обращения – 20.03.2017).
5. Бориско С.Н., Королев М. Р., Лобейко В. И. Оценка адекватности имитационного моделирования на основе элементов теории нечетких множеств // Известия волгоградского государственного технического университета. 2011.– Т.9. – C. 10-13.
6. Абу-Абед Ф.Н., Мартынов Д.В., Потапов А.Н., Угловский Е.П. Реализация субъективных методов вероятностной оценки адекватности тренажеров на основе комплекса программ с привлечением экспертов и анкетированием обучаемых // Программные продукты и системы. – 2016.– N3. – С. 121-128.
7. Усачев А.Б. Разработка теоретических и технологических основ производства чугуна процессом жидкофазного восстановления POMEJIT: Диссертация … на кандидата технических наук /МИСиС.– М., 2003. – 357 с.
8. Захаров А.Ю. Технологии исследований колесных транспортных средств на стендах с имитацией качения движителей по опорной поверхности: Диссертация … на кандидата технических наук /МГТУ им. Н.Э. Баумана.– М., 2016. – 171 с.
9. Васильев К. К. Служивый Н. Математическое моделирование систем связи. –Ульяновск: УлГТУ, 2008. – 170 с.
10. Курзаева Л.В., Новикова Т.Б. Оценка адекватности имитационной модели транспортного обслуживания строительных объектов // Международный журнал прикладных и фундаментальных исследований. – 2016. – N12-2. – С. 206-209.
11. Лычкина Н.Н. Имитационное моделирование экономических процессов. – М.: МГУУ, 2005. – 164 с.
12. Антонов А.С. [Параллельное программирование с использованием технологии MPI](https://parallel.ru/tech/tech_dev/MPI/). – М.: МГУ им. М.В. Ломоносова, 2004. – 72 с.
13. Параллельное программирование в системах с общей памятью. Инструментальная поддержка: учеб. Пособие / А.В. Сысоев, И.Б. Мееров, А.Н. Свистунов, А.Л. Курылев, А.В. Сенин, А.В. Шишков, К.В. Корняков, А.А. Сиднев. – Нижний Новгород: ННГУ, 2007. – 110 с.
14. Казённов А. М. Основы технологии CUDA // Компьютерные исследования и моделирование. – 2010. – Т.2, N3. – С. 295–308.
15. Знакомство с технологией CUDA [Электронный ресурс] – Электрон. текст. – 2008. – Режим доступа: https://www.overclockers.ua/software/nvidia-cuda/, свободный (дата обращения – 10.04.2017).
16. Берилло А. Nvidia CUDA – неграфические вычисления на графических процессорах [Электронный ресурс] – Электрон. текст. – 2008. – Режим доступа: http://www.ixbt.com/video3/cuda-1.shtml, свободный (дата обращения – 12.04.2017).
17. Официальный сайт OpenMP Architecture Review Board [Электронный ресурс] – Электрон. текст. – 2017. – Режим доступа: <http://www.openmp.org/>, свободный (дата обращения – 15.04.2017).
18. Эдвардс Ч., Пенни Д. Дифференциальные уравнения и краевые задачи. – М.: ООО «Вильямс», 2008. – 1004 с.
19. Антонов А.С. [Параллельное программирование с использованием технологии OpenMP](https://parallel.ru/tech/tech_dev/MPI/). – М.: МГУ им. М.В. Ломоносова, 2009. – 77 с.
20. Официальный сайт HybriLit ОИЯИ [Электронный ресурс] – Электрон. текст. – 2017. – Режим доступа: http://www.[hybrilit.jinr.ru/](http://www.openmp.org/), свободный (дата обращения – 27.04.2017).

# ПРИЛОЖЕНИЕ 1. Листинг кода параллельной реализации алгоритма

#include <fstream>

#include <cstring>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <malloc.h>

#include <algorithm>

#include <omp.h>

#include "INIReader.h"

///константы

const int perem=2;

const int colParams=20;

const int colParamModel=4;

double \*Params;

double \*\*\*Model1;

int maxY=-10000;

int maxZ=-10000;

int minY=100000;

int minZ=100000;

double minV=10000;

double maxV=-10000;

double start\_time, end\_time;

double fy(double y, double z, double a, double b);

double fz(double z, double y, double c, double d);

//уравнение хищников

double fy(double y, double z, double a, double b)

{

return ((a\*y-b\*y\*z));

}

//уравнение жертв

double fz(double z, double y, double c, double d)

{

return ((-c\*z+d\*y\*z));

}

//Рунге-Кутта 4-го порядка

void rk4(double fy(double, double,double,double),double fz(double, double, double, double),double t, double y, double z, double h, int n, double a, double b, double c, double d,int Vec)

{

int k;

double F1y, F2y, F3y, F4y, ta;

double F1z, F2z, F3z, F4z;

Model1[Vec][0][0]=y;

Model1[Vec][1][0]=z;

ta = t;

for (k = 1; k < n; k++)

{

F1y = h \* fy(y,z,a,b);

F2y = h \* fy(y + 0.5 \* h, z + F1y \* 0.5,a,b);

F3y = h \* fy(y + 0.5 \* h, z + F2y \* 0.5,a,b);

F4y = h \* fy(y + h, z + F3y,a,b);

F1z = h \* fz(z,y,c,d);

F2z = h \* fz(z + 0.5 \* h, y + F1z \* 0.5,c,d);

F3z = h \* fz(z + 0.5 \* h, y + F2z \* 0.5,c,d);

F4z = h \* fz(z + h, y + F3z,c,d);

y += (F1y + 2\*F2y +2\*F3y + F4y) / 6;

z += (F1z + 2\*F2z +2\*F3z + F4z) / 6;

Model1[Vec][0][k]=y;

Model1[Vec][1][k]=z;

t = ta + h\*k;

}

}

void main()

{

//начало замера времени расчета

start\_time = omp\_get\_wtime();

//инициализация параметров

double t0, tmax;

int n ;

double y0,z0;

double a0,b0,c0,d0;// начальные занчения парметров

double aMax,bMax,cMax,dMax; // верхняя граница параметров

double aDel,bDel,cDel,dDel;// шаг изменения параметров

double sigma;

double colmod;

//чтение начальных значений модели из файла

INIReader reader("./conf.ini");

t0 = reader.GetReal("Initial\_parameters", "t0", -1.0);

tmax = reader.GetReal("Initial\_parameters", "tMax", -1.0);

n=reader.GetReal("Initial\_parameters", "n", -1);

std::string file = reader.Get("Initial\_parameters", "file", "");

y0 = reader.GetReal("Initial\_parameters", "y0", -1.0);

z0 = reader.GetReal("Initial\_parameters", "z0", -1.0);

sigma=reader.GetReal("Initial\_parameters", "sigma", -1.0);

//чтение из файла параметов векторов

double MinMAx[8];

a0=reader.GetReal("Param\_vector", "a0", -1.0);

b0=reader.GetReal("Param\_vector", "b0", -1.0);

c0=reader.GetReal("Param\_vector", "c0", -1.0);

d0=reader.GetReal("Param\_vector", "d0", -1.0);

//нахождение минимума

MinMAx[0]=a0;

MinMAx[1]=b0;

MinMAx[2]=c0;

MinMAx[3]=d0;

aMax=reader.GetReal("Param\_vector", "aMax", -1.0);

bMax=reader.GetReal("Param\_vector", "bMax", -1.0);

cMax=reader.GetReal("Param\_vector", "cMax", -1.0);

dMax=reader.GetReal("Param\_vector", "dMax", -1.0);

MinMAx[4]=aMax;

MinMAx[5]=bMax;

MinMAx[6]=cMax;

MinMAx[7]=dMax;

aDel=reader.GetReal("Param\_vector", "aDel", -1.0);

bDel=reader.GetReal("Param\_vector", "bDel", -1.0);

cDel=reader.GetReal("Param\_vector", "cDel", -1.0);

dDel=reader.GetReal("Param\_vector", "dDel", -1.0);

//расчет количества моделей

double colA, colB,colC,colD;

colA=((aMax-a0)/aDel)+1;

colB=((bMax-b0)/bDel)+1;

colC=((cMax-c0)/cDel)+1;

colD=((dMax-d0)/dDel)+1;

double colM;

colM=(colA\*colB\*colC\*colD);

colmod=colM;

printf("Кол-во а=%f b=%f c=%f d=%f colM=%f \n",colA,colB,colC,colD,colmod);

//генерация трехмерного массива для хранения данных о модели

Model1=(double\*\*\*)malloc(colmod \* sizeof(double\*\*));

int m1,m2;

for (m1 = 0; m1 < colmod; m1++) Model1[m1] =(double\*\*)malloc(perem \* sizeof(double\*));

for (m1 = 0; m1 < colmod; m1++)

{

for (m2 = 0; m2 < perem; m2++) Model1[m1][m2] = (double\*)malloc((n+1) \* sizeof(double));

}

///массив параметров модели

Params = (double\*)malloc(colParams \* sizeof(double));

Params[0]=n+1;Params[1]=perem;Params[2]=t0;Params[3]=tmax;Params[4]=y0;Params[5]=z0;Params[6]=colmod;Params[7]=colParamModel;Params[8]=a0;Params[9]=aMax;Params[10]=aDel;Params[11]=b0;Params[12]=bMax;Params[13]=bDel;Params[14]=c0;Params[15]=cMax;Params[16]=cDel;Params[17]=d0;Params[18]=dMax;Params[19]=dDel;

// вычисляемые параметры

double h,t;

h = (Params[3]-Params[2]) /(Params[0]-1);

// чтение из файла Y И Z

int objcol=(int)Params[1];

double obj[objcol+1] [(int)Params[0]];

const char\* ptr = file.c\_str();

FILE \* f = fopen(ptr,"r");

int l=0;

while (fscanf (f, "%lf %lf %lf", &(obj[0][l]), &(obj[1][l]), &(obj[2][l])) != EOF) {

if(obj[1][l]>maxY) maxY=obj[1][l];

if(obj[1][l]<minY) minY=obj[1][l];

if(obj[2][l]>maxZ) maxZ=obj[2][l];

if(obj[2][l]<minZ) minZ=obj[2][l];

l++;

}

fclose(f);

printf(ptr);

///массив векторов параметров

double VectorParam [(int)Params[6]][(int)Params[7]];

//вызов расчета векторов параметров модели

double w,q,v,g ;

printf("Параметры модели:\n");

w=Params[8];

do

{

for(q=Params[11];q<=Params[12]+Params[13];q=q+Params[13])

{

for(v=Params[14];v<=Params[15];v=v+Params[16])

{

for(g=Params[17];g<=Params[18];g=g+Params[19])

{

VectorParam [m][0]=w;

VectorParam [m][1]=q;

VectorParam [m][2]=v;

VectorParam [m][3]=g;

}

}

}

w=w+Params[10];

}while(w<=Params[9]+0.02);

FILE \* vec = fopen("Vecmodel","w+b");

//Заполенение файла Vecmodel

int trap=0;

int rap=0;

for(trap=0;trap<=3;trap=trap+1)

{

for(rap=0;rap<colmod;rap=rap+1)

{

fprintf(vec,"%f ",VectorParam[rap][trap]);

}

fprintf(vec,"\n");

}

fclose(vec);

int VectModel;

//массив вектора квадрата невязок(

double resNev [(int)Params[6]][(int)Params[1]];

printf("Квадраты невязок по каждому вектору:\n");

//организация параллельных расчетов

#pragma omp parallel private (VectModel)

{

#pragma omp for

for(VectModel=0;VectModel<(int)Params[6];VectModel++)

{

//функция расчета модели

rk4 (fy,fz,Params[2], Params[4],Params[5], h, Params [0],VectorParam[VectModel][0],VectorParam[VectModel][1],VectorParam[VectModel][2],VectorParam[VectModel][3],VectModel);

double Nev [(int)Params[1]][(int)Params[0]];

int i;

//расчет разницы квадратов

for(i=0; i<n+1; i=i+1)

{

Nev[0][i]=pow(Model1[VectModel][0][i]-obj[1Ъ[i],2);

Nev[1][i]=pow(Model1[VectModel][1][i]-obj[2][i],2);

}

double Sum[(int)Params[1]];

Sum[0]=0;

Sum[1]=0;

int p=0;

for(p=0; p<n+1; p=p+1)

{

Sum[0]=Sum[0]+Nev[0][p];

Sum[1]=Sum[1]+Nev[1][p];

}

resNev[VectModel][0]=((Sum[0])/(n+1));

resNev[VectModel][1]=((Sum[1])/(n+1));

printf("n= %d y=%lf z=%lf\n",VectModel,resNev[VectModel][0],resNev[VectModel][1]);

}

}

//нахождение минимальной невязки

double Min[(int)Params[1]];

int cnt;

for(cnt=0;cnt<2;cnt++)

{

Min[cnt]=1000000;

}

int u;

for(u=0; u<(int)Params[6]; u=u+1)

{

if(resNev[u][0]<Min[0]) Min[0]=resNev[u][0];

if(resNev[u][1]<Min[1]) Min[1]=resNev[u][1];

}

printf("Минимальная невязка:\n");

printf("Ymin=%lf Zmin=%lf\n" ,Min[0],Min[1]);

//Формирование нуль-единичной матрицы

printf("Нуль-единичная матрица:\n");

int ZeroMatrix[(int)Params[6]][(int)Params[1]];

int x;

for(x=0; x<(int)Params[6]; x=x+1)

{

if((resNev[x][0]-Min[0])==0.0) {ZeroMatrix[x][0]=1;}

else ZeroMatrix[x][0]=0;

if((resNev[x][1]-Min[1])==0.0) {ZeroMatrix[x][1]=1;}

else ZeroMatrix[x][1]=0;

printf("%d| %d %d|\n",x,ZeroMatrix[x][0],ZeroMatrix[x][1]);

}

printf("Мера различия\n");

//сравнение с критерием дельта

double Mu[(int)Params[6]];

int r,model;

double MinRes=sigma;

for(r=0; r<(int)Params[6]; r=r+1)

{

Mu[r]=1-((ZeroMatrix[r][0]+ZeroMatrix[r][1])/2);

if(Mu[r]<MinRes) {MinRes=Mu[r]; model=r;}

printf("%d| %lf |\n",r,Mu[r]);

}

printf("Искомые параметры: m=%d вектор\_разл=%lf \n",model,MinRes);

//запись данных в файла result

FILE \* res = fopen("result","w+b");

int tmp1, tmp2;

for(tmp1=0;tmp1<Params[0];tmp1=tmp1+1)

{

fprintf(res,"%lf %lf %lf ",obj[0][tmp1],obj[1][tmp1],obj[2][tmp1]);

for(tmp2=0;tmp2<colmod;tmp2=tmp2+1)

{

fprintf(res,"%lf %lf ",Model1[tmp2][0][tmp1],Model1[tmp2][1][tmp1] );

}

fprintf(res,"\n");

}

fclose(res);

..создание

int tp1;

for(tp1=0;tp1<4;tp1=tp1+1)

{

if(minV>MinMAx[tp1]) minV=MinMAx[tp1];

}

int tp2;

for(tp2=4;tp2<8;tp2=tp2+1)

{

if(maxV<MinMAx[tp2]) maxV=MinMAx[tp2];

}

printf("adsdas %lf\n",maxV);

///Запись файла отрисовки

FILE \* gr = fopen("graphic.p","w+b");

fprintf(gr,"set term gif \\\n");

fprintf(gr," animate \\\n");

fprintf(gr," optimize \\\n");

fprintf(gr," delay 100 \\\n");

fprintf(gr," size 2000, 2000 \\\n");

fprintf(gr," crop \\\n");

fprintf(gr," font \"Times-Roman,20\"\n");

fprintf(gr,"set output \"graphic.gif\"\n");

fprintf(gr," y=4\n");

fprintf(gr," x=5\n");

fprintf(gr,"do for [i=%d:%f] { \n",1,colmod);

fprintf(gr,"set multiplot \\\n");

fprintf(gr," title \"Оценка адекватности модели объекту исследования\" font \"arial,25\" \\\n");

fprintf(gr," layout 2, 2 \\\n");

fprintf(gr," scale 1.1 \\\n");

fprintf(gr," columnsfirst scale 1.1,0.9 \n");

fprintf(gr,"set tmargin 2 \n");

fprintf(gr,"####################################График состояния численности популяции жертв\n");

fprintf(gr," set autoscale\n");

fprintf(gr," set size square\n");

fprintf(gr," unset log\n");

fprintf(gr," unset label\n");

fprintf(gr," set xtic auto \n");

fprintf(gr," set ytic auto\n");

fprintf(gr," set title \"График состояния численности популяции жертв\" font \"arial,35\"\n");

fprintf(gr," set xlabel \"Время\" font \"arial,25\"\n");

fprintf(gr," set ylabel \"Численность\" font \"arial,25\"\n");

fprintf(gr," set yrange[%d:%d]\n",minY-30,maxY+30);

fprintf(gr," set xrange[%d:%f] \n",0,tmax);

fprintf(gr," plot \"result\" using 1:2 title \'Y\' with linespoints,\"result\" using 1:y title \'Y\*\' with linespoints \n");

fprintf(gr,"####################################график численности хищников\n");

fprintf(gr," set autoscale\n");

fprintf(gr," set size square\n");

fprintf(gr," unset log\n");

fprintf(gr," unset label\n");

fprintf(gr," set xtic auto \n");

fprintf(gr," set ytic auto\n");

fprintf(gr," set title \"График состояния численности популяции хищников\" font \"arial,35\"\n");

fprintf(gr," set xlabel \"Время\" font \"arial,25\"\n");

fprintf(gr," set ylabel \"Численность\" font \"arial,25\"\n");

fprintf(gr," set xrange[%d:%f]\n",0,tmax);

fprintf(gr," set yrange[%d:%d] \n",minZ-30,maxZ+30);

fprintf(gr," plot \"result\" using 1:3 title \'Z\' with linespoints,\"result\" using 1:x title \'Z\*\' with linespoints \n");

fprintf(gr,"####################################Фазовый портрет численности хищников и жертв\n");

fprintf(gr," set autoscale\n");

fprintf(gr," set size square\n");

fprintf(gr," unset log\n");

fprintf(gr," unset label\n");

fprintf(gr," set xtic auto \n");

fprintf(gr," set ytic auto\n");

fprintf(gr," set title \"Фазовый портрет состояния численности популяции хищников и жертв\" font \"arial,35\"\n");

fprintf(gr," set xlabel \"Хищники\" font \"arial,25\"\n");

fprintf(gr," set ylabel \"Жертвы\" font \"arial,25\"\n");

fprintf(gr," set xrange[%d:%d]\n",minY-50,maxY+50);

fprintf(gr," set yrange[%d:%d] \n",minZ-50,maxZ+50);

fprintf(gr," plot \"result\" using 2:3 title \'YZ\' with linespoints,\"result\" using y:x title \'YZ\*\' with linespoints \n");

fprintf(gr,"y=y+2\n");

fprintf(gr,"x=x+2\n");

fprintf(gr,"####################################Вектор параметров модели\n");

fprintf(gr,"set bar 5.000000 front\n");

fprintf(gr,"set style circle radius graph 0.04, first 0.00000, 0.00000\n");

fprintf(gr,"set style ellipse size graph 0.07, 0.05, first 0.00000 angle 0 units xy\n");

fprintf(gr,"set style textbox transparent margins 3.0, 3.0 border\n");

fprintf(gr,"unset logscale\n");

fprintf(gr,"unset paxis 2 tics\n");

fprintf(gr,"unset paxis 3 tics\n");

fprintf(gr,"unset paxis 4 tics\n");

fprintf(gr,"unset paxis 5 tics\n");

fprintf(gr,"unset paxis 6 tics\n");

fprintf(gr,"unset paxis 7 tics\n");

fprintf(gr," set autoscale\n");

fprintf(gr," set size square\n");

fprintf(gr," unset log\n");

fprintf(gr," unset label\n");

fprintf(gr," set title \"Вектор параметров модели\" font \"arial,35\"\n");

fprintf(gr," set xlabel \"Номер коэффициента\" font \"arial,25\"\n");

fprintf(gr," set ylabel \"Значение\" font \"arial,25\"\n");

fprintf(gr," set xrange[%f:%f]\n",0.0,6.0);

fprintf(gr," set yrange[%lf:%lf] \n",minV-0.1,maxV+0.1);

fprintf(gr," unset paxis 2 tics\n");

fprintf(gr," set pointsize 1\n");

fprintf(gr," set bar 10\n");

fprintf(gr," set grid noytics mxtics mytics\n");

fprintf(gr,"set paxis 1 range [ \* : \* ] noreverse nowriteback\n");

fprintf(gr,"set paxis 2 range [ \* : \* ] noreverse nowriteback\n");

fprintf(gr,"set paxis 3 range [ \* : \* ] noreverse nowriteback\n");

fprintf(gr,"set paxis 4 range [ \* : \* ] noreverse nowriteback\n");

fprintf(gr,"set paxis 5 range [ \* : \* ] noreverse nowriteback\n");

fprintf(gr,"set paxis 6 range [ \* : \* ] noreverse nowriteback\n");

fprintf(gr,"set paxis 7 range [ \* : \* ] noreverse nowriteback\n");

fprintf(gr,"set colorbox vertical origin screen 1, 0.7, 1 size screen 0.5, 0.8, 1 front noinvert bdefault\n");

fprintf(gr,"p=i+2\n");

char p= '%';

fprintf(gr,"plot \"Vecmodel\" using 2:p pt 2 title '','' using 2:p:(sprintf(\"[%%s=%%s]\",stringcolumn(1),stringcolumn(p))) with labels title sprintf(\"№ вектора %%d\",i) font \"arial,20\" tc lt 2 \n");

fprintf(gr,"}\n unset multiplot");

fclose(gr);

end\_time = omp\_get\_wtime();

printf("Время на замер %lf\n", end\_time-start\_time);

printf("maxY=%d minY=%d maxZ=%d minZ=%d \n", maxY, minY,maxZ, minZ);

}