Санкт-Петербургский государственный университет

Прикладная математика и информатика

Динамические системы, эволюционные уравнения, экстремальные задачи и математическая кибернетика

Аветян Манук Гегамович

# Моделирование естественных нейронных сетей

Выпуская квалификационная работа

Научный руководитель:
доктор технических наук, профессор
Фрадков Александр Львович

Рецензент:
кандидат физико-математических наук
Усик Егор Владимирович

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Applied Mathematics and Informatics

Dynamical systems, evolution equations, extremal problems and mathematical cybernetics

Avetian Manuk Geghamovich

# Modelling natural neural networks

Graduation Project

Scientific supervisor:

Doctor of Engineering, Professor

A.L. Fradkov

Reviewer:

Ph.D.

E.V. Usik

Saint-Petersburg

2017

# Аннотация

Целью данной работы являлось создание программных средств для автоматизации анализа в одной из задач исследования математических моделей естественных (биологических) нейронных сетей: выявления типа химеры по данным вычислительного эксперимента для двухуровневой сети осцилляторов ФитцХью – Нагумо. После рассмотрения типов химер были выделены признаки, которые применили на экспериментальных данных для вывода алгоритма. Альтернативно задача классификации была решена однослойным персептроном с помощью тех же данных. Оба решения были применены на большом количестве данных для построения карт.

# Оглавление

# 1. Постановка задачи

## 1.1 Предпосылки

В последнее время исследование связанных систем привело к совместным исследованиям между различными областями, такими как нелинейная динамика, сетевая наука и статистическая физика, с множеством приложений, например, в физике, биологии и технологии. Поскольку числовые ресурсы развивались быстрыми темпами, анализ и моделирование больших сетей с использованием все более сложных схем сопряжения стали доступными, что привело к появлению множества новых динамических сценариев.

Так в 2002 году Курамото и Баттогтох сообщили, что массивы нелокально связанных осцилляторов могут спонтанно разделиться на синхронизированные и десинхронизированные субпопуляции [1]. Это было удивительным аспектом, так как раннее считалось, что связанные идентичные осцилляторы либо синхронизируются, либо будут работать несвязно, хаотично. Так как сеть обладала гибридной природой, объединяющей как когерентные, так и некогерентные части, Стивом Строгацем было предложено называть такие состояния химерными. из-за их похожести на мифологических греческих зверей, будто собранных из несопоставимых частей [2].

Недавние работы показали, что состояния химер не ограничиваются фазовыми осцилляторами, а могут быть найдены в большом разнообразии различных систем. Они включают в себя временные дискретные и хаотические модели с непрерывным временем [3,4] и не ограничены одним пространственным измерением. Также двухмерные конфигурации учитывают состояния химер [5,6]. Более того, аналогичные сценарии существуют для временной связи [7][9], и их динамические свойства и симметрии также подвергались теоретическим исследованиям [4,8,9] Только в самом недавнем прошлом химерные состояния были реализованы в экспериментах на химических осцилляторах [10] и электрооптических решетках связанных карт [11]. Нелокальность соединения - ключевая характеристика для состояний химер - также предполагает интересную связь с материаловедением. Например, магнитные частицы Януса, которые подвергаются индуцированному синхронизацией структурному переходу во вращающемся магнитном поле [12,13]. Нелокальность имеет большое значение не только для состояний химер, но и для других динамических явлений, таких как турбулентная перемежаемость [14]. Гибридные состояния были также сообщены в контексте нейронауки под понятием Bump State [15]. Позднее они были подтверждены для нелокально связанных моделей Ходжкина-Хаксли [16] и могут объяснять экспериментальное наблюдение частичной синхронности в нейральной активности во время движения глаз [17].

Предпосылками к исследованию системы, описанной в данной статье, послужили такие задачи как:

1. Изучение однополушарного сна.

   Многие существа спят с половиной своего мозга одновременно [18]. Такой однополушарный сон впервые был зарегистрирован у дельфинов и других

морских млекопитающих, и в настоящее время он встречается у птиц и выводится у ящериц [19]. Когда записываются мозговые волны, бодрствующая сторона мозга демонстрирует десинхронизированную электрическую активность, соответствующую миллионам нейронов, осциллирующих по фазе, тогда как спящая сторона синхронизирована. Изучение простейших систем из двух популяций осцилляторов, слабо аналогичным двум полушариям, таких, что одна синхронизирует, а другая нет, позволит лучше понять данный феномен.

2. Изучение эпилепсии.

   Двухуровневая сеть, имитирует две различные области мозга, связанные между собой. Эпилепсия представляет собой передачу состоянии синхронизации из одной области мозга в другую, т.е. нейроны в мозгу человека начинают патологически синхронизироваться. Теоретическое исследование такой сети позволило бы лучше понять механику работы эпилепсии, а значит и способы борьбы с ней.

## 1.2 Химерные состояния

Общим определением химерных состояний является такое состояние сети из одинаковы связанных осцилляторов, в котором одновременно существуют синхронные и асинхронное поведения.

После открытия сделанным Куромото появилось множество вопросов, на некоторые из которых за последние несколько лет удалось ответить.

На данный момент известно, что:

1) Для конечных сетей осцилляторов, численные эксперименты показывают, что состояния химер на кольце являются на самом деле долгоживущими переходными процессами [20]. При бесконечном числе осцилляторов состояния химеры оказываются стабильными [21].
2) Химеры устойчивы ко многим различным типам возмущений [22].
3) Химеры встречаются в пространственных сетях. Например, для осцилляторов, распределенные по бесконечной плоскости [23,24,25], тор [26,27] и сфера [28,29].
4) Химеры встречаются в произвольных сетях [30].

Но несмотря на большое количество работ по химерам, следует отметить, что общие механизмы формирования химерных состояний в различных системах практически остаются невыясненными. Более того, до сих пор отсутствует какая-либо классификация ансамблей взаимодействующих осцилляторов, позволяющая прогнозировать реализацию химер в определенных классах систем.

Более точный обзор химерных состояний можно увидеть в статье [31].

## 1.3 Осцилляторы ФитцХью Нагумо

В нейробиологии всегда стояла задача как можно точнее описать работу биологического нейрона. В 1952 году была разработана так называемая модель Ходжкина – Хаксли, за которую её создатели получили Нобелевскую премию в области физиологии и медицины за 1963 год. Эта модель представляет собой систему дифференциальных уравнений четвертого порядка.

Данная модель является сложной и поэтому стали появляться различные виды ее упрощения. Одной из них является модель ФитцХью-Нагумо. Она названа в честь Р. ФитцХью, который предложил систему в 1961 году, [32] и Дж. Нагумо, который вместе с С. Аримото и С. Йошизава сделал тоже самое в 1962 году [33]. Модель ФитцХью-Нагумо представляет собой упрощенную версию модели Хожкина-Хаксли и описывается системой дифференциальных уравнений второго порядка:

$$\begin{cases} \varepsilon \dot{u} = u - \dfrac{u^3}{3} - v \\ \dot{v} = u + a \end{cases} \tag{1.1}$$

где переменная u и $v$ – активатор и ингибитор соответственно. Малый параметр $\varepsilon > 0$ характеризует разницу масштабов времени для активатора и ингибитора. В данной работе (а также в других упоминаемых работах по ФитцХью – Нагумо) выбрано $\varepsilon = 0.05$, но в общем случае его выбирают из отрезка [0.01; 0.05]. В зависимости от пороговой переменной $a$ система находится либо в возбуждаемом состоянии при $|a| > 1$ (реагириет только на подачу внешнего сигнала), либо в осцилляторном состоянии (постоянно самовозбуждаясь и двигаясь по предельному циклу). В данной работе $a = 0.5$.

Осцилляторы ФитцХью – Нагумо актуальны не только в неврологии, но и в таких схемах, как химические [24] и оптоэлектронные[34] осцилляторы и нелинейные электронные схемы [35].

## 1.4 Химерные состояния в сетях ФитцХью – Нагумо

Существование химерных состояний в сети из нейронов ФитцХью – Нагумо было показано Ириной Омельченко в 2013 году в статье [36]. В ней рассматривалась сеть осцилляторов, представимая в виде кольца из нейронов с нелокальной связью, которая определяется как связь нейрона $i$ со всеми соседями $j$, расстояние до которых $|j-i| \le R$:

$$\varepsilon \frac{du_k}{dt} = u_k - \frac{u_k^3}{3} - v_k + \frac{\sigma}{2R}\sum_{j=k-R}^{k+R}\left[b_{uu}(u_j - u_k) + b_{uv}(v_j - v_k)\right], \tag{1.2а}$$

$$\frac{dv_k}{dt} = u_k + a_k + \frac{\sigma}{2R}\sum_{j=k-R}^{k+R}\left[b_{vu}(u_j - u_k) + b_{vv}(v_j - v_k)\right]. \tag{1.2б}$$

Обычно поведение системы зависит не столько от значения $R$, сколько от отношения $R/N = r$, называемого радиусом связи. Кроме того, связь каждой пары нейронов была выполнена при помощи матрицы поворота на угол $\varphi$:

$$B = \begin{pmatrix} b_{uu} & b_{uv} \\ b_{vu} & b_{vv} \end{pmatrix} = \begin{pmatrix} cos\varphi & sin\varphi \\ -sin\varphi & cos\varphi \end{pmatrix}. \tag{1.3}$$

Итоговые уравнения сети можно записать таким образом:

$$\begin{pmatrix} \varepsilon\dot{u}_i \\ \dot{v}_i \end{pmatrix} = \begin{pmatrix} u_i - \frac{u_i^3}{3} - v_i \\ u_i + a \end{pmatrix} + \frac{\sigma}{2rN}\sum_{|j-i|\le N/r}\begin{pmatrix} cos\varphi & sin\varphi \\ -sin\varphi & cos\varphi \end{pmatrix}\begin{pmatrix} u_j - u_i \\ v_j - v_i \end{pmatrix} \tag{1.4}$$

Используя работу [3] было показано, что состояния химеры возникают только при углах $\varphi \approx \frac{\pi}{2}$. Поэтому для моделирования было выбрано $\varphi = \frac{\pi}{2} - 0.1$.

Также, на возникновение химер очень влияет сила связи $\sigma$. В частности, было показано, что при маленьких значениях силы связи (порядка 0.1), химеры возникают при радиусах от 0.25 до 0.43.

Пространственную когерентность и некогерентность состояния химеры можно характеризовать параметром локального порядка [9, 37]

$$Z_k = \left|\frac{1}{2\delta}\sum_{|j-i|\le\delta} e^{i\,\Theta_j}\right|, k = 1, \dots, N, \tag{1.5}$$

где $\Theta_j = arctan\ \frac{v_j}{u_j}$ обозначает геометрическую фазу $j$-ого нейрона.

Параметр локального порядка $Z_k = 1$ указывает, что $k$-я единица принадлежит когерентной части состояния химеры (синхронизация нейрона с соседями на расстоянии $\delta$), а $Z_k < 1$ для некогерентных частей (хаоса). На рисунке 2.1(d) показан параметр локального порядка в интервале времени $t \in [1000; 5000]$, где желтый цвет обозначает когерентные области.

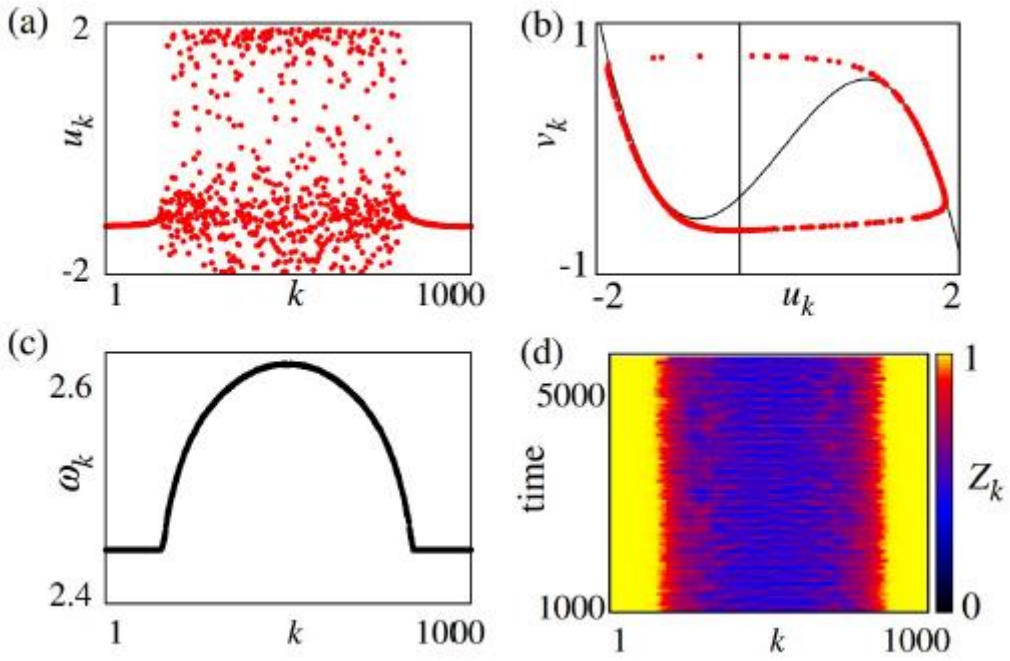


Рис. 1.1.

(a) Значения $u_k$ в момент времени t = 5000. (b) Положения на предельном цикле в момент времени $t = 5000$. (c) Средние частоты каждого нейрон $\omega_k$. (d) значения локального порядка $Z_k$.

Параметры: $\sigma = 0.1, r = 0.35, N = 1000, a = 0.5, \varphi = \frac{\pi}{2} - 0.1$.

## 1.5 Двухуровневая сеть. Начальные условия

Дальнейшим развитием задачи являлось исследование двухуровневой сети, имитирующей две различные области мозга, связанные между собой [38]. Предположим, что в одной области начинается патологическая синхронизация. При каких параметрах связей эта синхронизация перейдет на вторую область? Именно так развивается эпилепсия нейроны в мозгу человека начинают патологически синхронизироваться. Теоретическое исследование такой сети позволило бы лучше понять механику работы эпилепсии, а значит и способы борьбы с ней.

Для создания асимметрии в каждом уровне была выбрана своя топология сети.

В первом уровне нелокальная связь с радиусом $r = 0.35$ и силой связи $\sigma_1 = 0.1$. Во втором уровне фрактальная топология. Для создания фрактальной топологии используется определенный начальный паттерн из нулей и единиц, и затем каждая единица заменяется на начальный паттерн, а ноль на набор нулей такой же длины, как и начальный паттерн. Данную процедуру повторяют несколько раз, а потом к полученной строке с начала приписывают еще один ноль, и затем записывают результат как первую строку матрицы смежности. Остальные строки матрицы смежности получаются просто циклическим сдвигом первой строки. К примеру, паттерн 101 создает что-то наподобие канторовской лестницы, а для одной итерации размер сети $N = 3^2 + 1 = 10$, а матрица смежности выглядит так:

$$G = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ & \vdots & & \vdots & & & & \vdots & & \vdots \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \tag{1.6}$$

Сила связи $\sigma_2$ была также выбрана равной 0.1. Для создания фрактальной топологии здесь использовался паттерн 101, проинтегрированный четыре раза. Итоговый размер второго уровня составляет $N = 3^5 + 1 = 244$ нейрона. Размер первого уровня был выбран таким же.

Потом два уровня соединяются «один над другим», связываются нейроны, имеющие одинаковые номера в каждом уровне. Параметры силы данной связи, а также времени задержки, являются основными переменными, зависимость поведения системы от которых исследовалась. При этом данная связь осуществляется не через матрицу поворота, а только по $u-u$ переменным.

Схематично полученную сеть можно изобразить так:

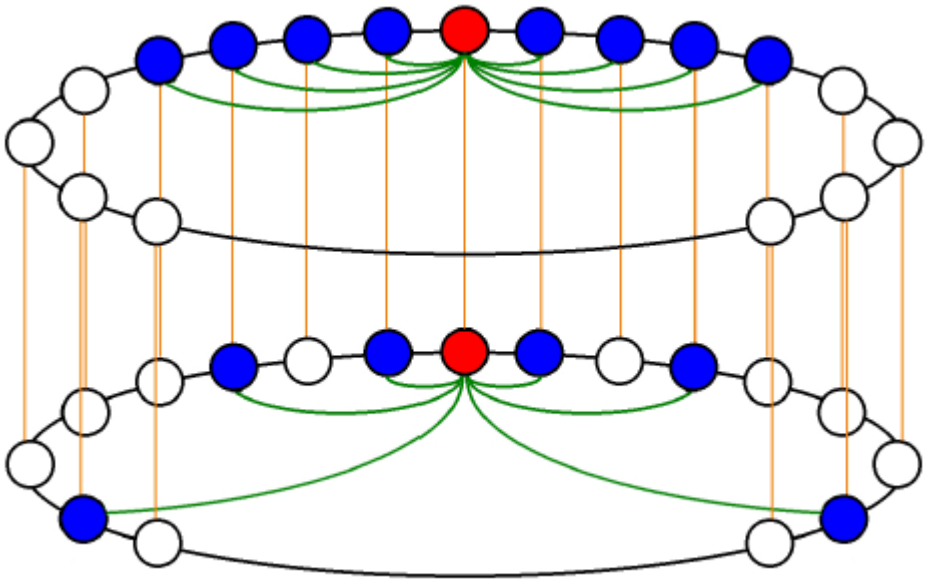

Рис. 1.2.

Уравнения, описывающие систему:

$$\begin{pmatrix} \varepsilon\dot{u}_{1i} \\ \dot{v}_{1i} \end{pmatrix} = \begin{pmatrix} u_{1i} - \frac{u_{1i}^3}{3} - v_{1i} \\ u_{1i} + a \end{pmatrix} + \frac{\sigma_1}{2rN} \sum_{|j-i| \le N/r} R_\varphi \begin{pmatrix} u_{1j} - u_{1i} \\ v_{1j} - v_{1i} \end{pmatrix} + \sigma_3 \begin{pmatrix} u_{2i}(-\tau) - u_{1i} \\ 0 \end{pmatrix}, \quad (1.7а)$$

$$\begin{pmatrix} \varepsilon\dot{u}_{2i} \\ \dot{v}_{2i} \end{pmatrix} = \begin{pmatrix} u_{2i} - \frac{u_{2i}^3}{3} - v_{2i} \\ u_{2i} + a \end{pmatrix} + \frac{\sigma_2}{|G_i|} \sum_{G_{ij}=1} R_\varphi \begin{pmatrix} u_{2j} - u_{2i} \\ v_{2j} - v_{2i} \end{pmatrix} + \sigma_3 \begin{pmatrix} u_{1i}(-\tau) - u_{2i} \\ 0 \end{pmatrix}, \quad (1.7б)$$

где $R_\varphi = \begin{pmatrix} cos\varphi & sin\varphi \\ -sin\varphi & cos\varphi \end{pmatrix}$ – матрица поворота, $\tau$ – время задержки.

Начальные условия для моделирования были выбраны таким образом: первый уровень начинал в химерном состоянии, тогда как второй – в синхронизированном. На рис. 1.3. можно увидеть график значений активаторов во времени, а также график средних частот, в случае, когда уровни между собой не связаны (сила связи $\sigma_3 = 0$).

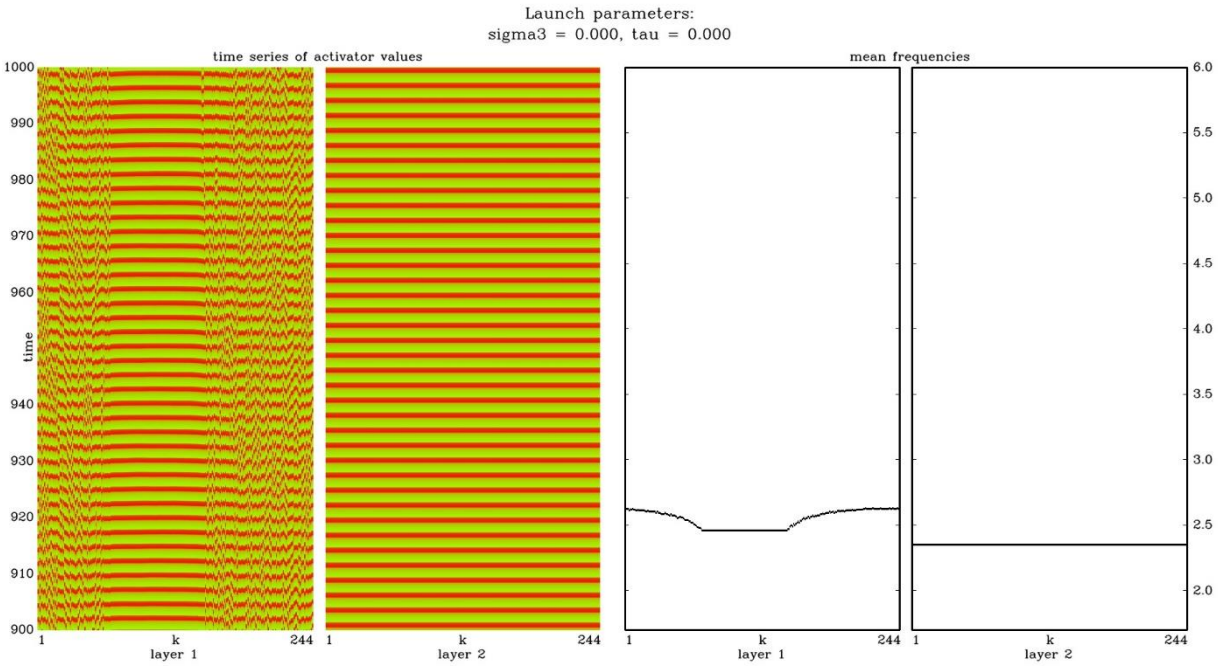


Рис. 1.3. Химерное состояние при $\sigma_3 = 0$ и $\tau = 0$.

## 1.6 Задача классификации

После моделирования 100 равномерно распределенных точек на области с $\sigma_3 \in [0; 0.5]$ и $\tau \in [0; 3]$ при времени моделировании $t = 100000$ и их рассмотрении удалось выделить три типа колебаний:

1. Классическая химера

   Данная химера (рис.1.4., рис. 1.5) как следует из названия похожа на классический типа химер описанный в главе 1. Далее будем говорить о ней как, о типе 1.

2. Хаос и синхронизация

   Хаос (рис.1.6., рис. 1.7) представляет собой хаотические колебания.

   Синхронизацией (рис.1.8., рис. 1.9) были названы колебания, которые по состоянию напоминает состояние синхронизации, но на самом деле она достигается при $t \to \infty$. А в общем случае данный тип напоминает хаос с большей частотой колебания и меньшей амплитудой, поэтому решено было объединить их. Кроме того, оба данных колебания не совсем подходят под описание химер. Далее будем ссылаться на них как к типу 2.

3. Двухкластерная химера

   Данная химера (рис.1.10., рис. 1.11) сильнее всего выделяется среди остальных. Представляет собой разделение точек на два слоя, в общем случае, каждый из которых напоминает предыдущие типы. Соответственно обозначим ее как тип 3.

Возможно на области существуют и другие типы химер, но для дальнейшего их исследования требуется иметь возможность классифицировать их на огромных областях. Поэтому возникла следующая задача: создать алгоритм классификации данных химер и написать программное обеспечение для его использования. После применение данной программы на область можно будет по созданной карте продолжать дальнейшее исследование.

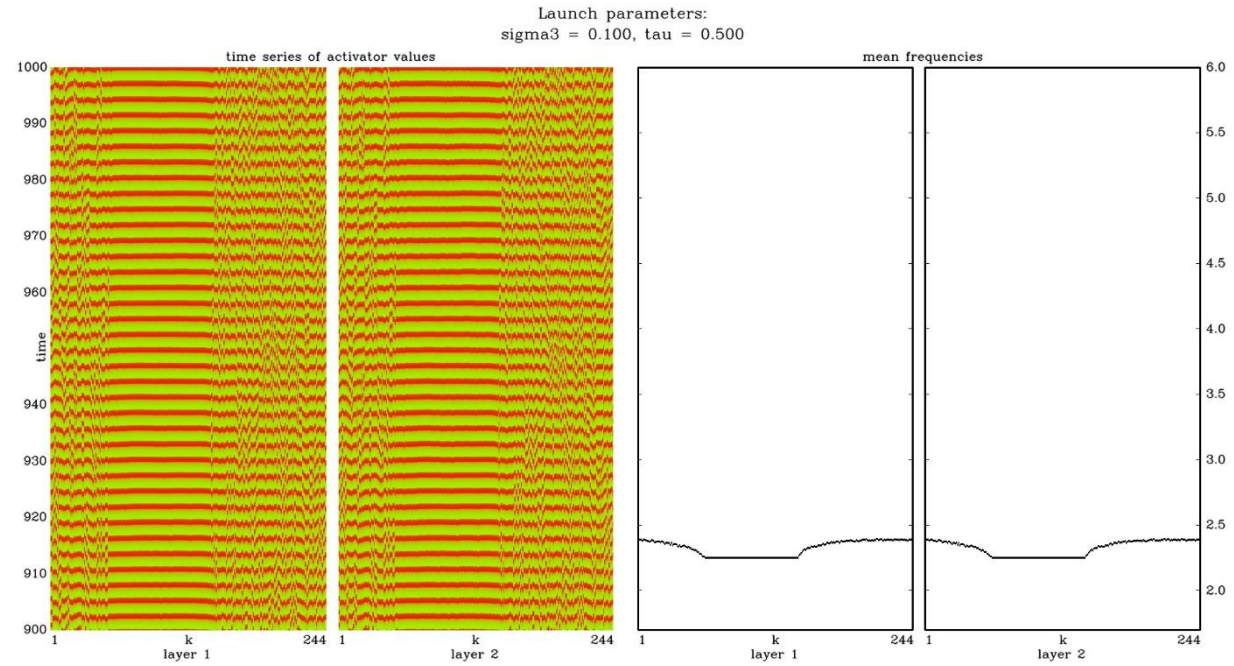


Рис. 1.4. Классическая химера при $\sigma_3 = 0.1$, $\tau = 0.5$ и $t = 1000$

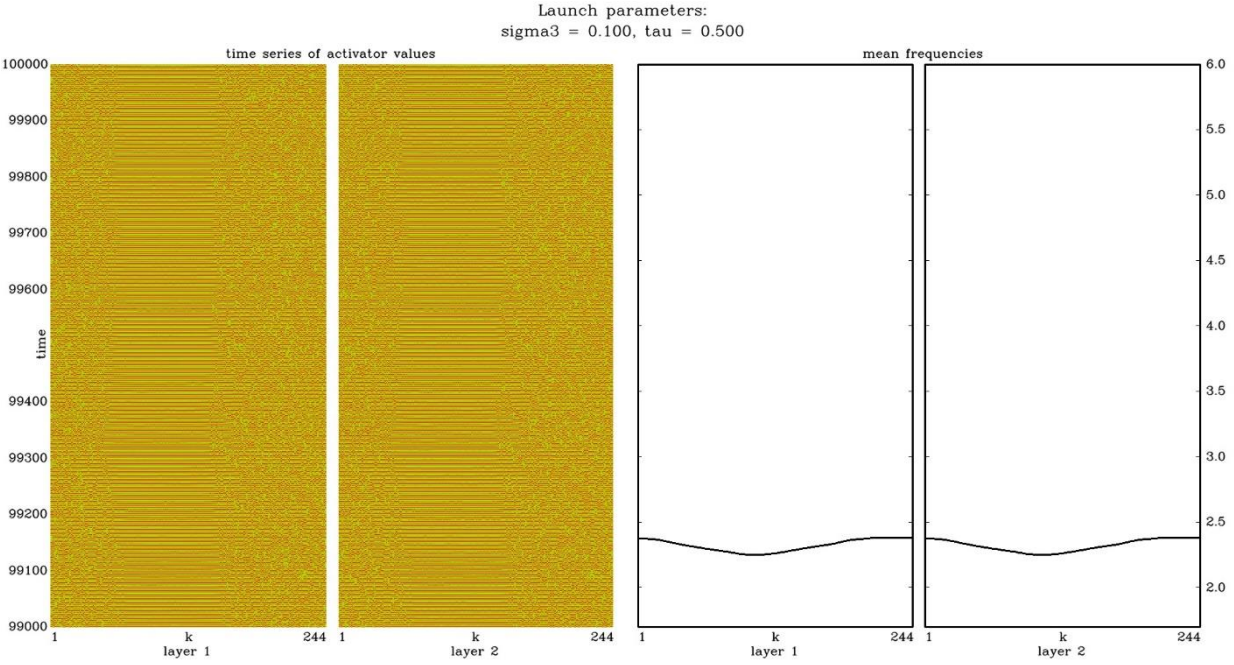


Рис. 1.5. Классическая химера при $\sigma_3 = 0.1$, $\tau = 0.5$ и $t = 100000$

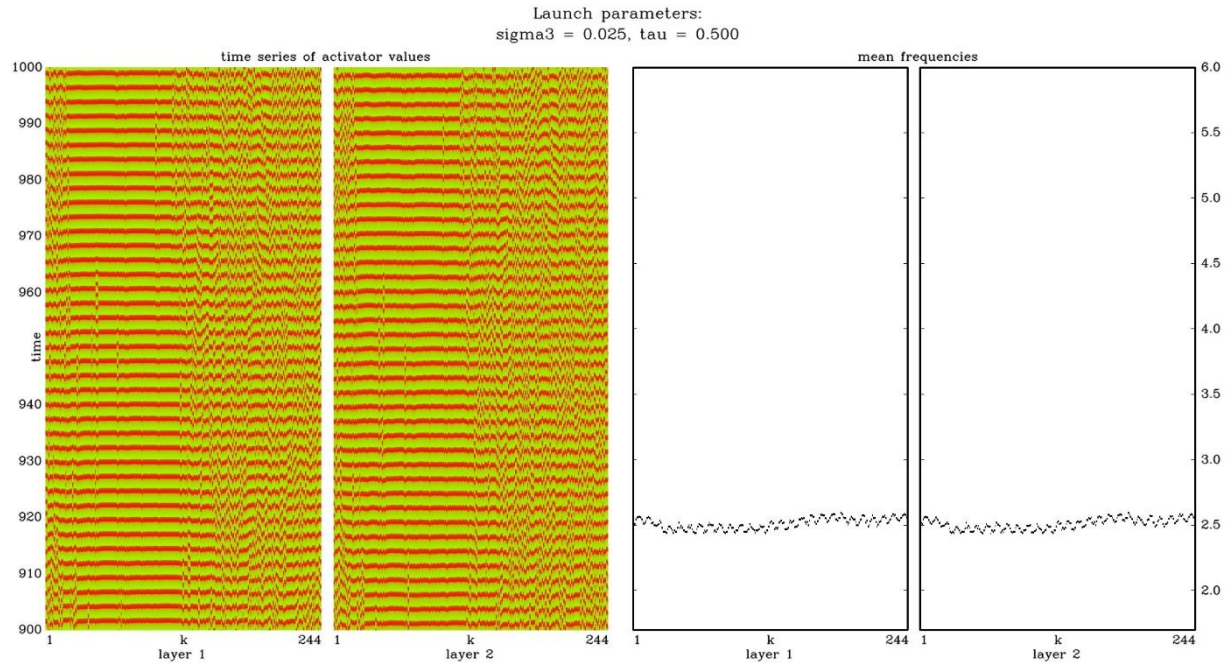


Рис. 1.6. Хаос при $\sigma_3 = 0.1$, $\tau = 0.5$ и $t = 1000$

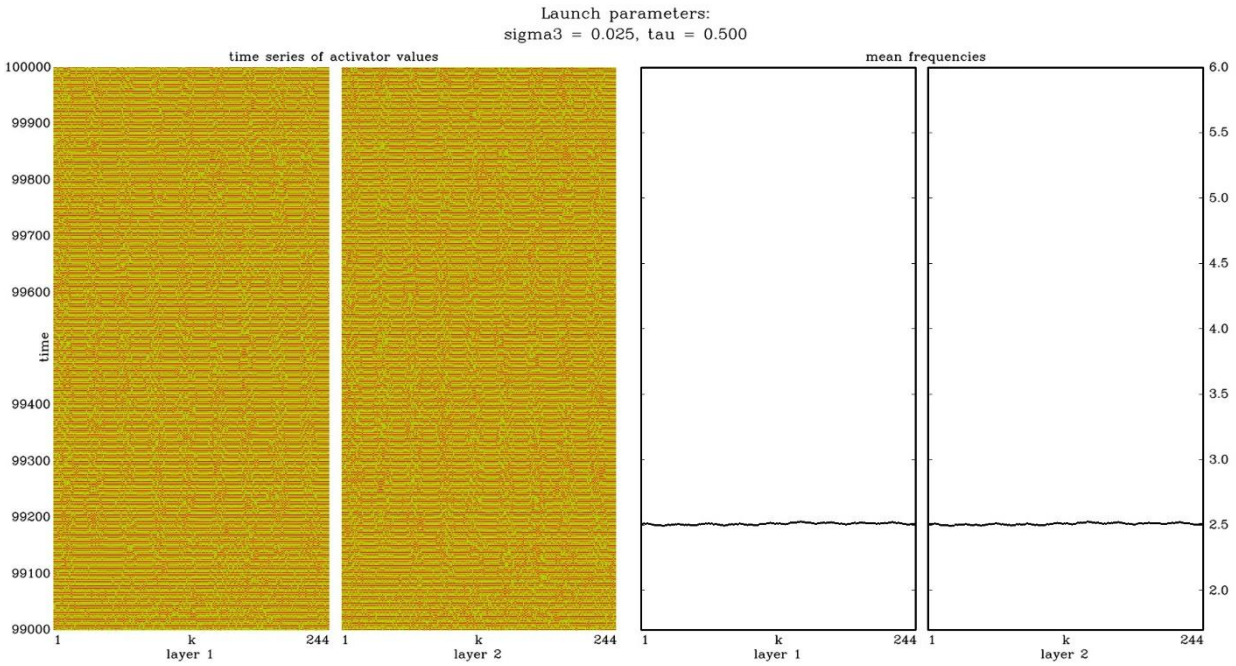


Рис. 1.7. Хаос при $\sigma_3 = 0.1$, $\tau = 0.5$ и $t = 100000$

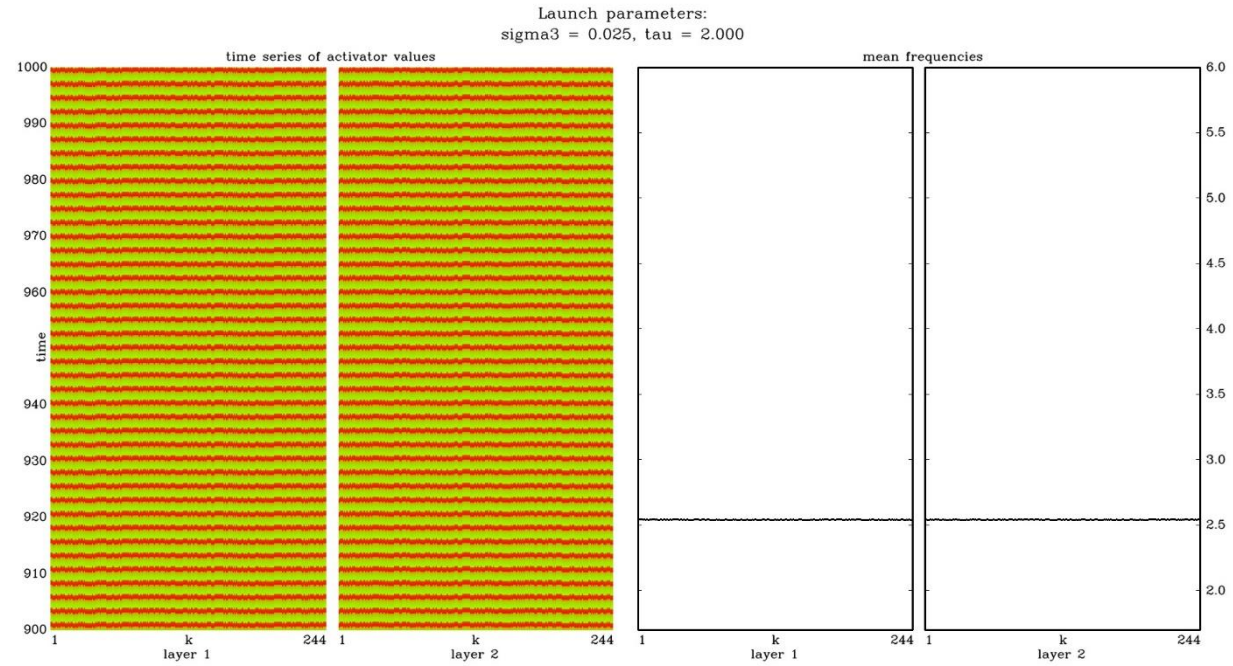


Рис. 1.8. Синхронизация при $\sigma_3 = 0.1$, $\tau = 0.5$ и $t = 1000$

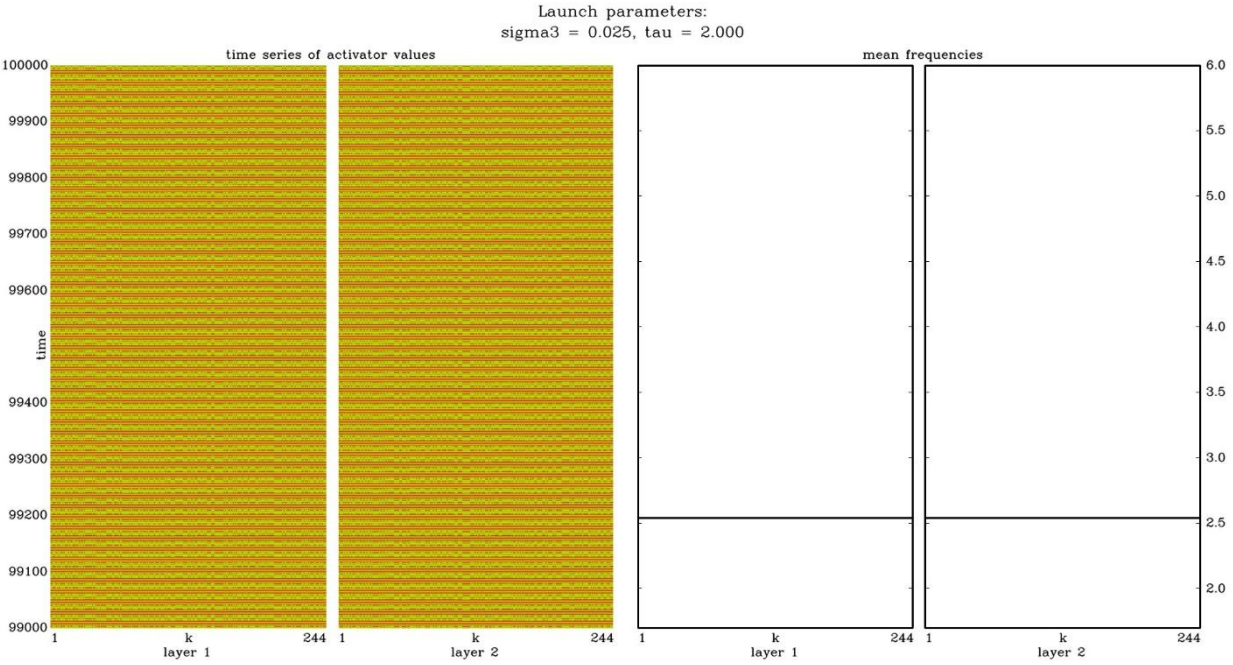


Рис. 1.9. Синхронизация при $\sigma_3 = 0.1$, $\tau = 0.5$ и $t = 100000$

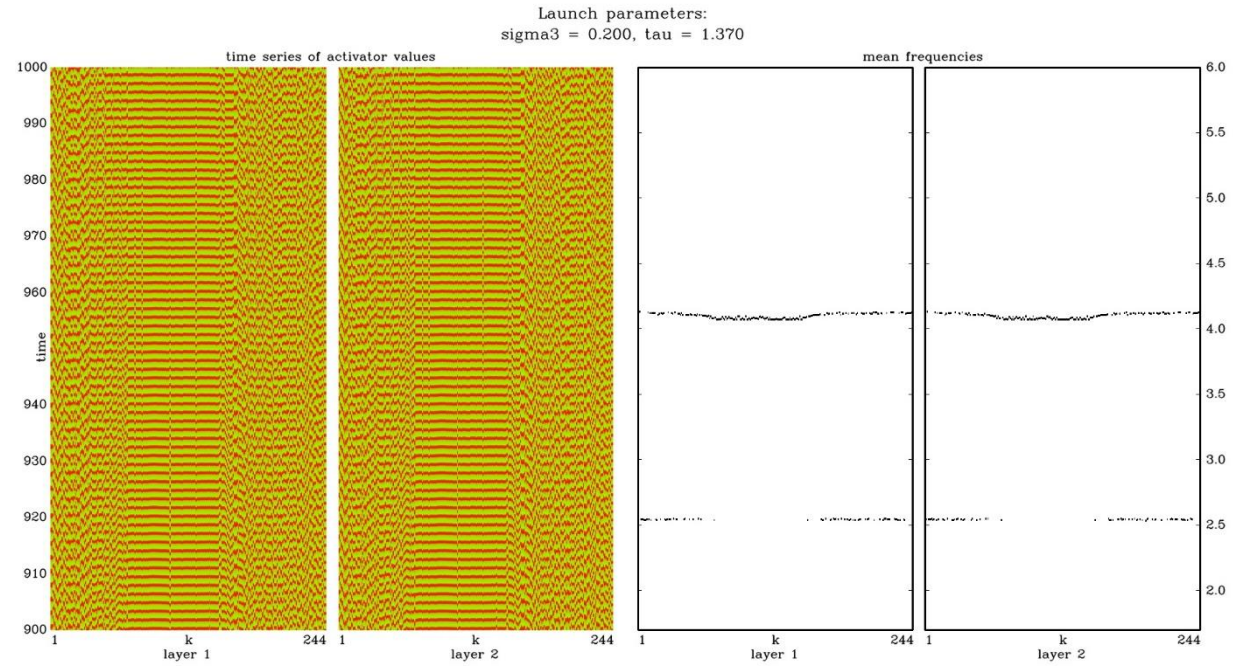


Рис. 1.10. Двухкластерная химера при $\sigma_3 = 0.1$, $\tau = 0.5$ и $t = 1000$

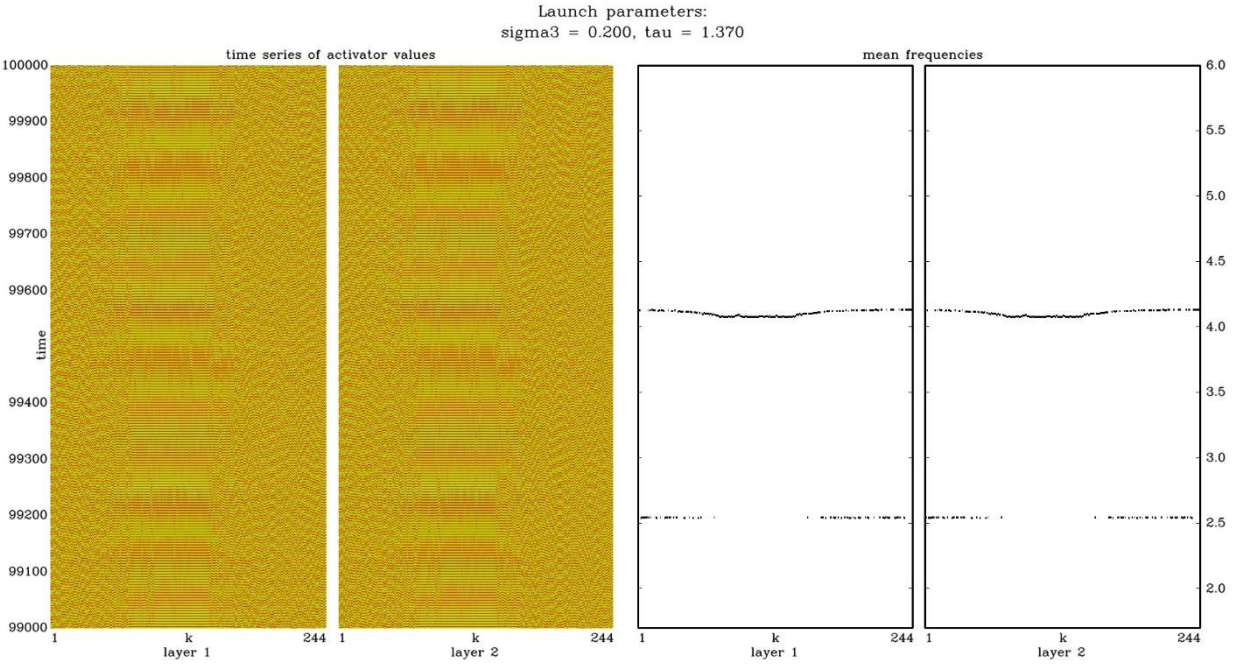


Рис. 1.11. Двухкластерная химера при $\sigma_3 = 0.1$, $\tau = 0.5$ и $t = 100000$

# 2. Классификация химер

## 2.1 Введение

После рассмотрения колебаний из 1.6 видно насколько различные могут быть типы химер. Из-за чего эффективности стандартных алгоритмов ставится под сомнение. Поэтому решено было выделить некоторые признаки из данных типов химер и создать алгоритме на их основе с использование некоторых правил.

## 2.2 Признак 1

Первый признак используется для того, чтобы явно выделить тип 3 от остальных. Для этого нейроны разбиваются на множества, в которых найдется такая пара $i, j$, что разность $(\omega_i - \omega_j)$ не будет превышать по модулю некоторое $\varepsilon$. После этого выбираются два наибольших по мощности множества и соответственно отношение количество нейронов в них к $N$ обозначаются как $K_1$ и $K_2$.

## 2.3 Признак 2

Второй признак используется для выявления частоты колебании в химере, это позволит выделить тип хаос/синхронизация. Этим признаком будет количество экстремумов в химере.

Так как при моделировании есть погрешности и получаем график химеры не является гладким, то следует сгладить его.

Сглаживание выглядит следующим образом:

1. также, как и в 1 признаке, разбиваем нейроны на множества, и, учитывая, что 1 признак отсекает тип 3, считаем, что работаем с другими типами, а, значит, химера находится там, где количество нейронов больше. Нейроны, не вошедшие в это множество, будет считать, что имеют погрешность в расчетах
2. заменяем значение $\omega$ по правилу:

$$\bar{\omega}_0 = \omega_0, \qquad \bar{\omega}_N = \omega_N, \qquad \bar{\omega}_i = \frac{\bar{\omega}_l + \omega_r}{2}, \qquad i \neq 0, N, \tag{2.1}$$

где $l$ – индекс ближайшего слева нейрона для $i$, находящегося в том же множестве, а $r$ – ближайшего справа.

Далее находим экстремумы, проверяя, является ли каждая $i$ точка экстремумом для ближайших $r_{extr}$ соседей в своем множестве, и обозначаем как $K_{extr}$.

## 2.3 Признак 2

Последний признак был добавлен из-за того, что существуют типы синхронизаций, имеющие малое количество экстремумов (см. таблицу).

При синхронизации нейроны имеют идентичные показатели ω, поэтому при состоянии близком к этому следует рассчитать:

$$\Delta\omega = \omega_{max} - \omega_{min}, \tag{2.2}$$

где $\omega_{max}$ и $\omega_{min}$ соответственно максимальное и минимальное значение ω на множестве имеющем наибольшую мощность (см. признак 1).

## 2.4 Правила классификации

### 2.4.1 Описание

Теперь, выделив признаки химер, можно создать правила (алгоритм) для определения ее типа:

1. Если выполняются соотношения $K_1 \leq P_1$ и $K_2 \geq P_2$, то данная химера относится к типу 3. Иначе переходим к шагу 2.
2. Если выполняются соотношения $K_{extr} \leq P_{extr}$ и $\Delta\omega \geq \varepsilon_\omega$, то это химера типа 1. Иначе химера тип 2.

$P_1, P_2, P_{extr}, \varepsilon_\omega$ – некоторые вещественные значения, которые следует подбирать исходя из экспериментальных данных.

### 2.4.2 Схема

Приведем схему для наглядности на рис. 2.1.

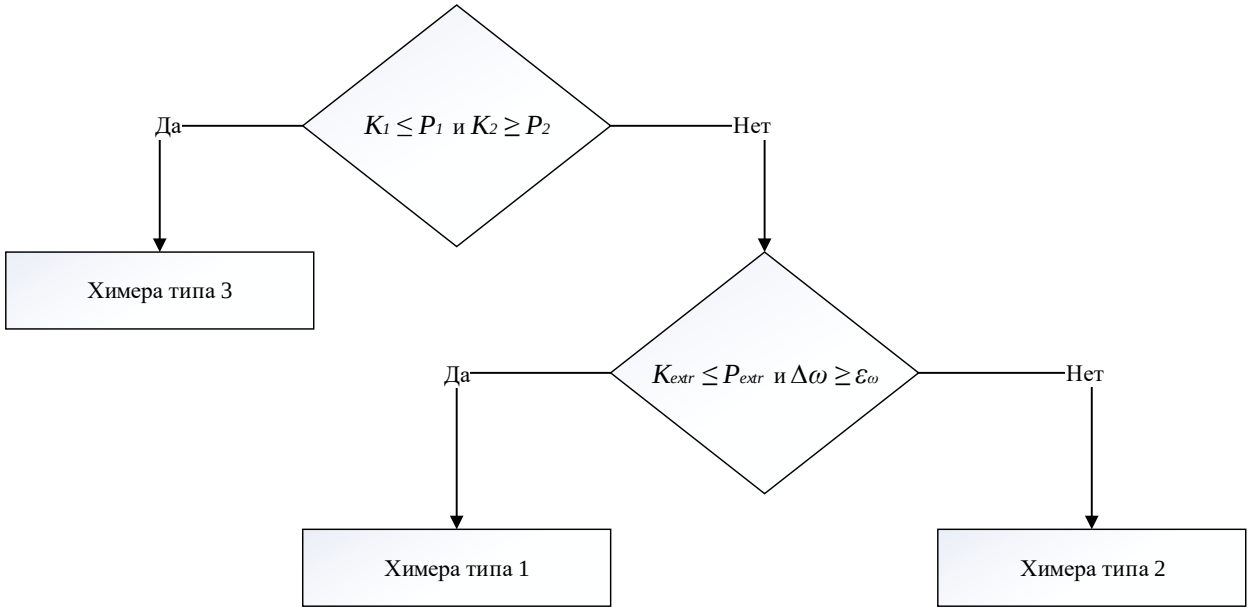


Рис. 2.1. Схема выведенного алгоритма

## 2.5 Экспериментальные расчеты признаков

После выделения признаков применяем их на 100 точках, равномерно распределенных на области с $\sigma_3 \in [0; 0.5]$ и $\tau \in [0; 3]$. И из полученных результатов, подберем параметры для алгоритма.

Так как точек много, то мы не будем приводить всю таблицу значений (см. приложение А), а только обобщение из нее.

| | | | Тип 1 | Тип 2 | Тип 3 |
|---|---|---|---|---|---|
| $K_1$ | | Max | 1 | 1 | 0,913934 |
| | | Min | 0,967213 | 0,922131 | 0,504098 |
| | | Среднее арифметическое | 0,995133 | 0,992008 | 0,822541 |
| $K_2$ | | Max | 0,028689 | 0,077869 | 0,495902 |
| | | Min | 0 | 0 | 0,086066 |
| | | Среднее арифметическое | 0,004184 | 0,007889 | 0,177459 |
| $K_{extr}$ | при $r_{extr} = 3$ | Max | 46 | 64 | 67 |
| | | Min | 4 | 10 | 42 |
| | | Среднее арифметическое | 16,35417 | 29,55 | 52,1 |
| | при $r_{extr} = 6$ | Max | 19 | 27 | 31 |
| | | Min | 3 | 5 | 23 |
| | | Среднее арифметическое | 6,6875 | 13,575 | 26,1 |
| | при $r_{extr} = 10$ | Max | 9 | 21 | 16 |
| | | Min | 2 | 2 | 10 |
| | | Среднее арифметическое | 4,666667 | 8,85 | 13,4 |
| $\Delta\omega$ | | Max | 0,15199 | 0,03337 | 0,18787 |
| | | Min | 0,02753 | 0,00619 | 0,01797 |
| | | Среднее арифметическое | 0,073197 | 0,017742 | 0,067766 |

Табл. 2.1. Обобщение результатов применения признаков к 100 точкам, равномерно распределенных на области с $\sigma_3 \in [0; 0.5]$ и $\tau \in [0; 3]$

Учитывая данные из таблицы 1, было решено установить следующие параметры:

1) $r_{extr} = 6$
2) $P_1$ = 0,914. Максимальное значения $K_1$ у типа 3 с округлением вверх до 3 знака после запятой.
3) $P_2 = 0{,}086$. Минимальное значения $K_2$ у типа 3 с округлением вниз до 3 знака после запятой.
4) $P_{extr} = 19$. Максимальное значения $K_{extr}$ у типа 1.
5) $\varepsilon_\omega = 0{,}028$. Минимальное значение $\Delta\omega$ у типа 1 с округлением вверх до 3 знака после запятой.

Использование данных правил на 100 точках таблицы (приложение А) дает точный результат с типом, который мы для них определили.

## 2.6 Применение алгоритма

После подбора параметров приступаем к классификации большой области данных. Для этого выделим равномерно 1600 точек на области с $\sigma_3 \in [0; 0.5]$ и $\tau \in [0; 3]$ и используем алгоритм описанный выше.

На рис 2.1. видны основные области скопления различных типов химер. Но данная карта не отображает процесс перехода одного типа химеры к другой. Что затрудняет дальнейшее исследование химер.

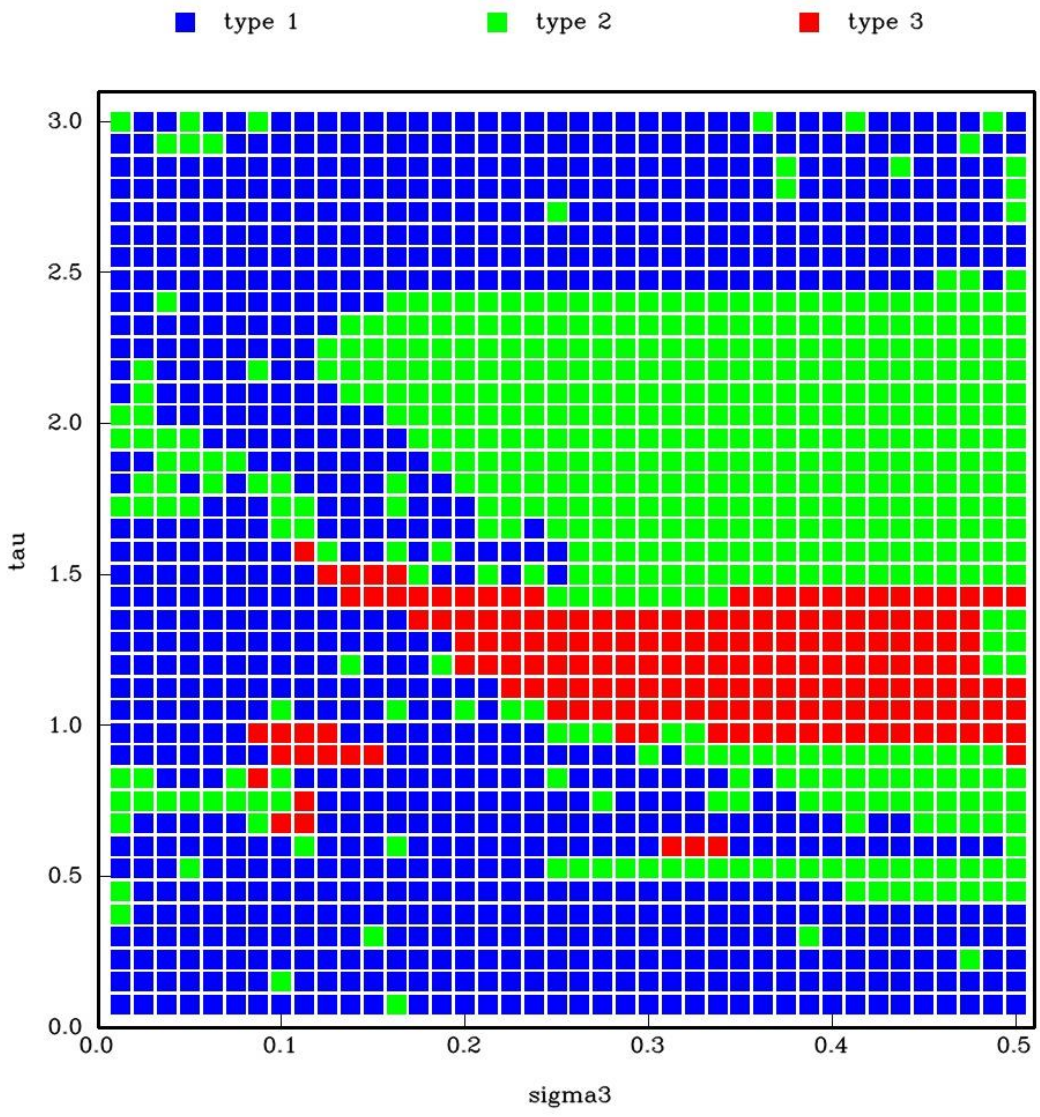


Рис 2.2. Применение алгоритма классификации на 1600 точек области с $\sigma_3 \in [0; 0.5]$ и $\tau \in [0; 3]$

# 3. Классификация однослойным персептроном

## 3.1 Введение

В предыдущей главе были выделены признаки и применены на область путем подбора параметром по таблице экспериментальных данных.

Во время заполнения таблицы возникали моменты со сложностью определения типа: если даже выделялись признаки различных типов, приходилось с точностью говорить, к какому типу данную химеру причисляем. Кроме того, параметры подбирались на наше усмотрение. И как уже говорилось в 2.6 переходный процесс одного типа к другому также становится недоступным.

Данные проблемы можно относительно решить искусственной нейронной сетью, которую часто применяют в задачах классификации, таких как распознавание образов, текста и речи [39,40,41].

## 3.2 Однослойный персептрон

В 1957 году Френком Розенбаттом была предложена одна из первых моделей искусственных нейронных сетей – перцептрон [42]. Существует множество различных видов данной модели, но наиболее простой и популярной является однослойный персептрон, который является двухуровневой нерекуррентной сетью (рис. 3.1).

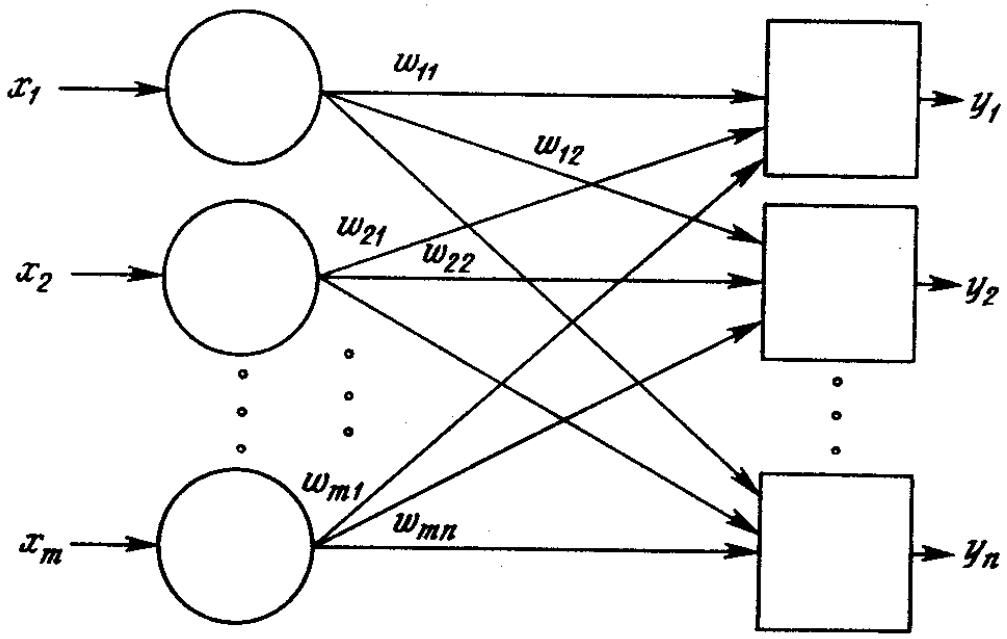


Рис. 3.1. Общий вид однослойного персептрона

Кратко опишем данную модель:

1. Первый уровень представляет собой набор входных данных, ассоциирующийся с синапсами, компоненты, которого соединены с компонентами второго уровня - нейронами.
2. Каждая связь характеризуется весовым коэффициентом $w_{ij}$.

3. Состояние нейрона есть взвешенная сумма его входов

$$s_j = \sum_{i=1}^{m} w_{ij} x_i. \tag{3.1}$$

4. Выход нейрона определяется функцией состояния:

$$y_j = f(s_j), \tag{3.2}$$

где $f$ – функция активации.

## 3.3 Обучение персептрона

Обучение происходит на основе метода с учителем, т.е. каждый образец обучающей выборки содержит вектор входных с вектором желаемых выходных данных, по которой система настраивает весовые коэффициенты $w_{ij}$. Компоненты входного вектора имеют непрерывным диапазоном значений; в то время как компоненты выходных данных могут быть представлены либо 0, либо 1. Это связанно с тем, что в Розенбатт в своей работе использовал пороговую функцию активации (рис 3.2).

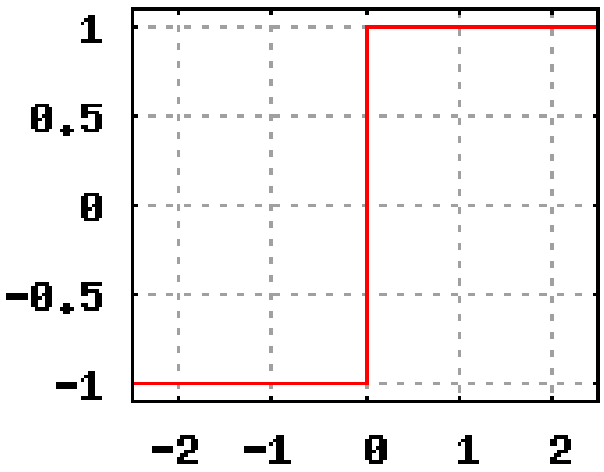


Рис 3.2. Пороговая функция

Уидроу и Хофф [43] изменили модель Розенблатта, а именно вместо пороговой функции использовали линейную (рис. 3.3).

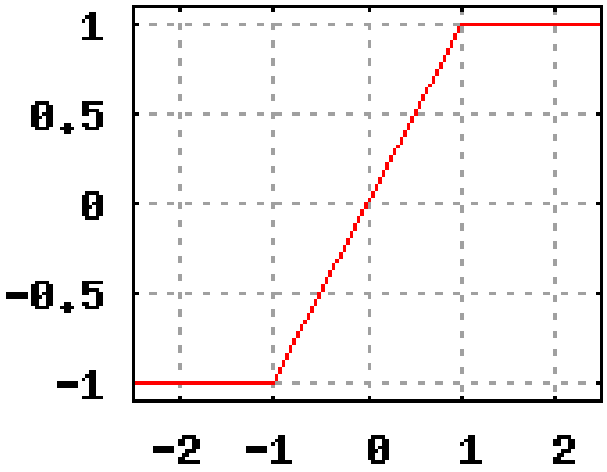


Рис 3.2. Линейная функция

Метод обучения Уидроу-Хоффа [43] также известен как дельта правило. Его целью является минимизация функцию ошибки:

$$E = \max_k(E^k), \tag{3.3}$$

$$E^k = \frac{1}{2}\sum_{j=1}^{n}\left(d_j^k - y_j^k\right)^2, \tag{3.4}$$

где $E$ – максимальная ошибка сети, $E^k$ – ошибка сети на $k$-ом образце, $d_j^k$ – желаемое значения $j$-ого компонента вектора выхода на $k$-ом образце.

Для минимизации $E$ проведем минимизацию $E^k$ методом градиентного спуска.

$$w_{ij}(t+1) = w_{ij}(t) - \alpha\frac{\partial E^k}{\partial w_{ij}(t)} \tag{3.5}$$

где $\alpha$ – скорость обучения, а

$$\frac{\partial E^k}{\partial w_{ij}(t)} = \frac{\partial E^k}{\partial y_j^k}\cdot\frac{\partial y_j^k}{\partial w_{ij}(t)} = \left(y_j^k - d_j^k\right)\cdot x_i^k. \tag{3.6}$$

В итоге получаем следующую формулу для подсчета весовых коэффициентов:

$$w_{ij}(t+1) = w_{ij}(t) - \alpha\left(y_j^k - d_j^k\right)\cdot x_i^k. \tag{3.7}$$

## 3.4 Применение персептрона

Для нашей задачи будем использовать схему персептрона на рис. 3.3.

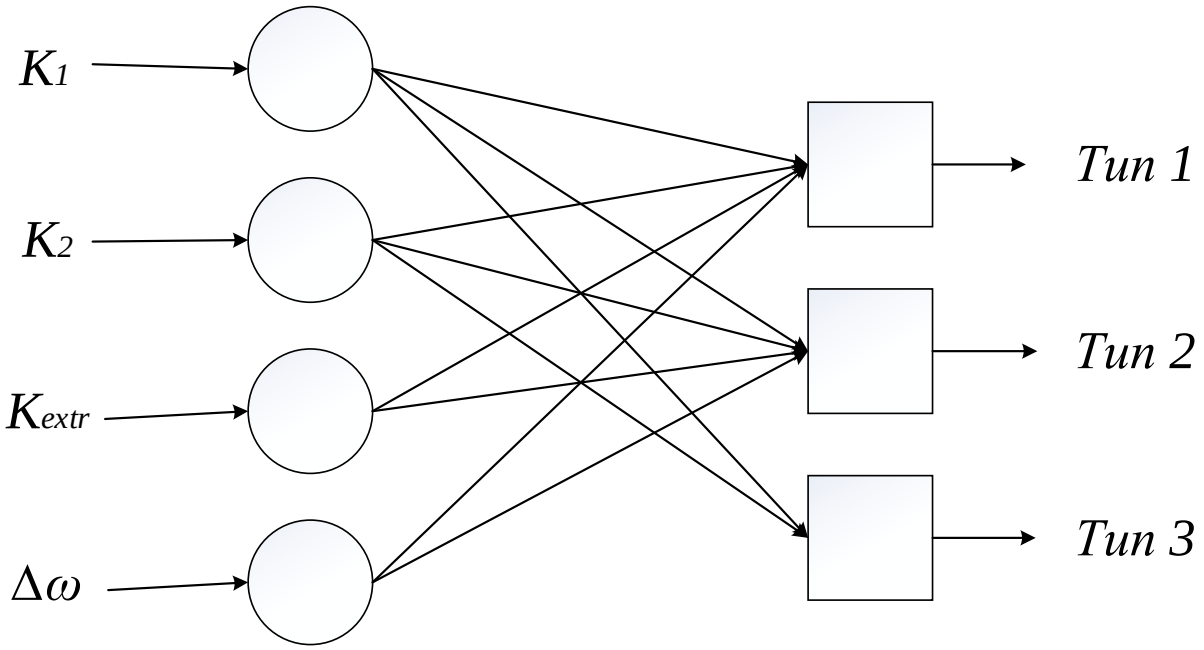


Рис 3.3. Схема персептрона

Зададим следующие значения параметров:

$$w_{ij} = 0.001, \alpha = 0.1. \tag{3.8}$$

Обучающей выборкой послужили значения из таблиц в приложении А. Так как выведенный алгоритм дает значения в этих точках, совпадающим с выбранным нами типом, то значение функции ошибки послужит критерием сравнения этих методов. Кроме того, учитывая, что в пограничных точках между областями различных типов химер значение $E$ будет относительно большим, то следует ввести также дополнительный критерий:

$$\bar{E} = \frac{\sum_{k=1}^{P} E^k}{n}, \tag{3.9}$$

где $\bar{E}$ – среднеарифметическая функция ошибки сети, $P$ – мощность обучающей выборки.

Цикл обучения проходил многократно до тех пор, пока количество итерации не превысит значения $K_{it} = 1000000$ или пока значения функции $\bar{E}$ и $E$ не стабилизируется, т.е. разница значении функции будет меньше некоторого $\varepsilon_{it} = 0.0000001$.

Входные данные следует нормализовать иначе у нас будут величины разных порядков, что может сильно повлиять на решение нейронной сети. Для этого зададим параметры $K_1^n = 1, K_2^n = 1, K_{extr}^n = 30, \Delta\omega^n = 0.05$ на которые соответственно будем делить компоненты входных параметров.

Таким образом входные и выходные данных будут принадлежать области значения $[0; 1]$. Получаем, что каждый тип на выходе имеет степень достоверности, варьирующуюся между 1 (принадлежит данному типу) и 0 (не принадлежит данному типу). Для наглядности поставим в соответствие палитру RGB (рис. 3.4), что даст возможность лучше наблюдать изменения состояния химер.

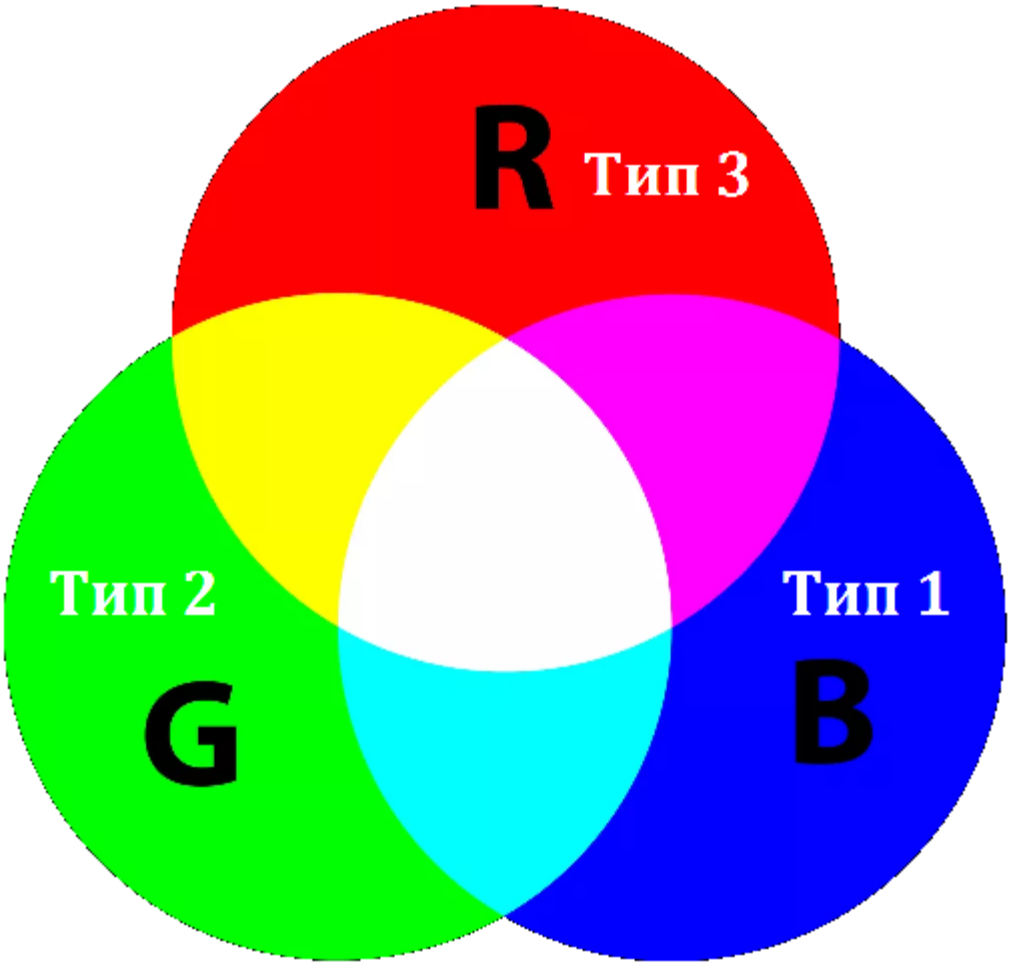


Рис. 3.4. Палитра цветов при различных значениях типов химер и их промежуточных состояниях.

После процесса обучения были получены следующие результаты функций ошибок:

$$\bar{E} = 0.0197, E = 0.5725 \quad (3.9)$$

Как и ожидалось, значение $E$ получилось относительно большим, в то время как $\bar{E}$ в допустимых пределах.

После обучение применим результат на 1600 равномерно распределенных точек области с $\sigma_3 \in [0; 0.5]$ и $\tau \in [0; 3]$ (рис. 3.5).

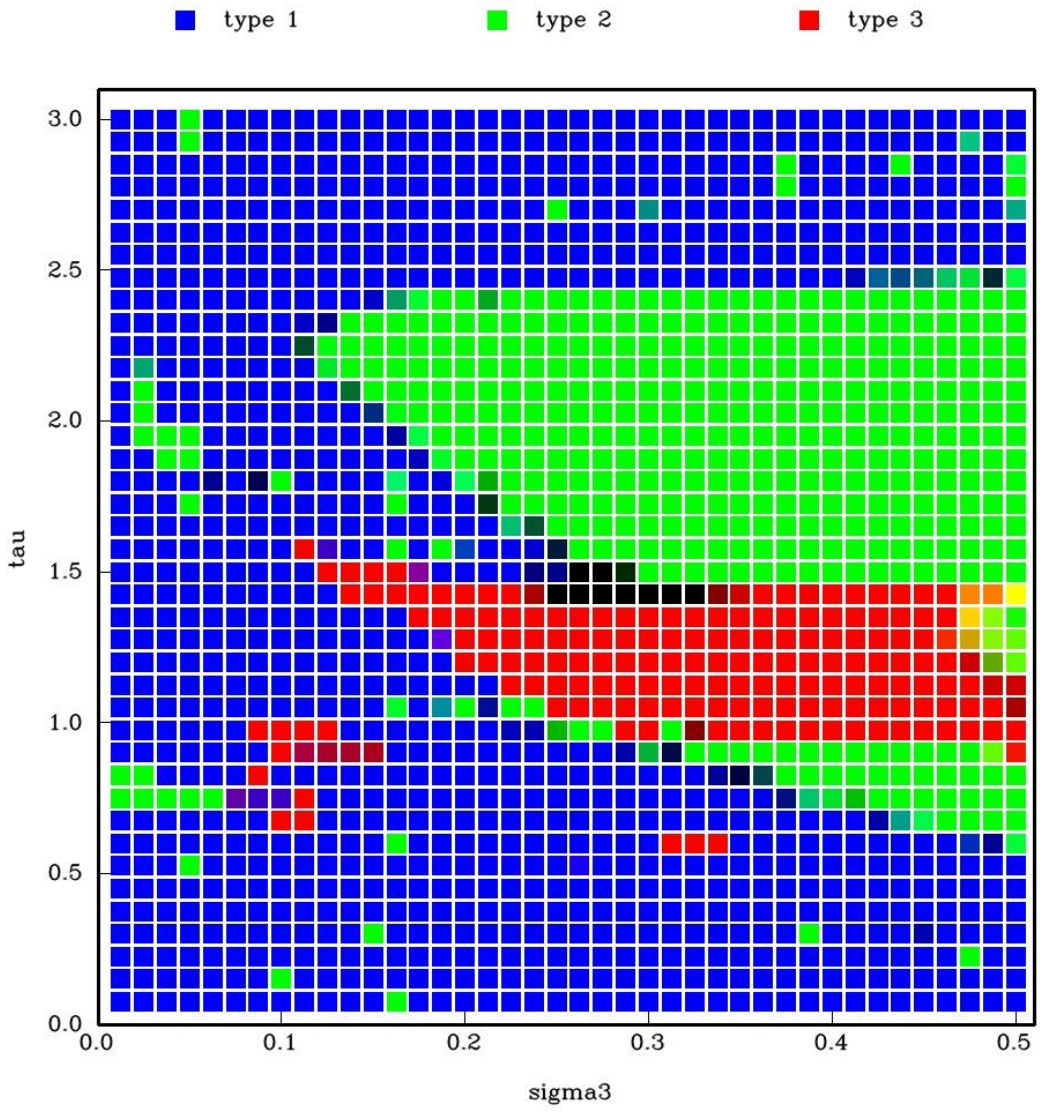


Рис. 3.5. Применение персептрона к 1600 точкам области
с $\sigma_3 \in [0; 0.5]$ и $\tau \in [0; 3]$

# 4. Моделирование и реализация

При классификации приходится работы с большими объемами данных, а, значит, что общее время обработки информации может длится месяцами. Кроме того, методы классификации могут быть разными, поэтому решено было разделить общую задачу на три части:

1. моделирование и расчеты
2. обучение нейронной сети
3. классификация данных и построение карты.

Это подталкивает нас сделать три программы, имеющие соответствующие цели. Далее рассмотрим каждую задачу отдельно.

Программное обеспечение было написано на языке C++ 11 в среде Microsoft Visual Studio 2015 Express с использование графической библиотеки OpenCV для вывода графики и библиотеки OpenMP для распараллеливания расчетов.

## 4.1 Моделирование и расчеты

### 4.1.1 Описание

Для этой задача была создана программа, которая на основе исходных данных и области классификации моделирует постановку задачи из главы 2 и сохраняет результаты моделирования для дальнейшего использования.

Для расчета системы дифференциальных уравнений (требуется ссылка) использовался метод Хойна (Heun’s method), который относится к модифицированным методам Эйлера.

Среднее время моделировании для одних исходных данных при времени моделировании t = 100 000 составляет 40-45 минут.

Среднее время моделировании для шести исходных данных при параллельном вычислении составляет 100 – 110 минут.

Код программы описан в приложении Б.

### 4.1.2 Блок-схема

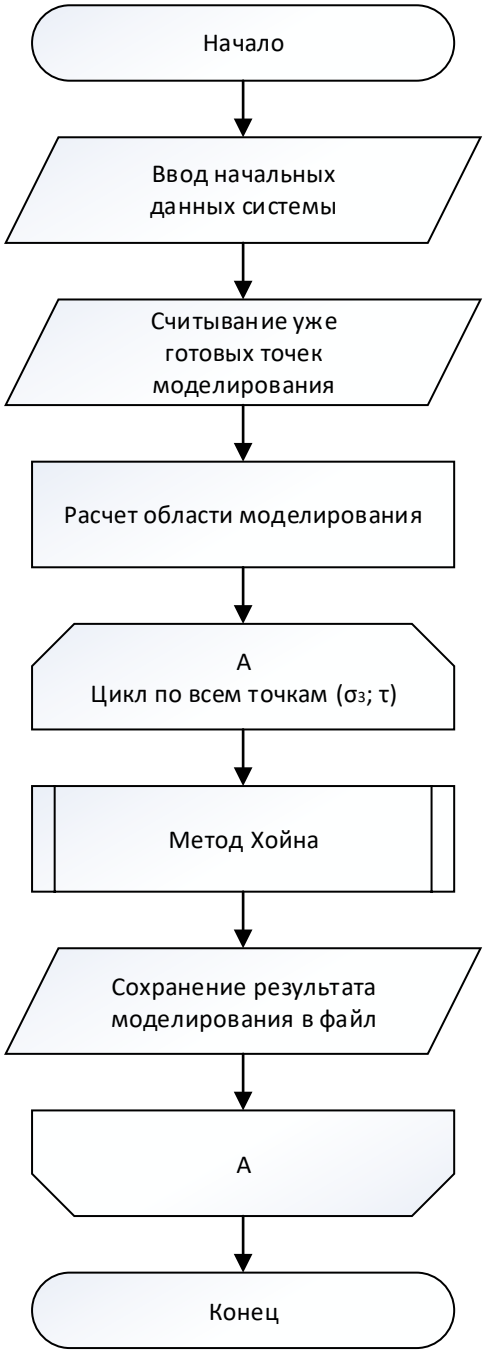


Рис 4.1. Схема программы для моделирования и расчетов

## 4.2 Обучение нейронной сети

### 4.2.1 Описание

Целью этой задачи является обучение нейронной сети для дальнейшего использования. На вход программы поступают данные в виде описания признаков классификации и тип химеры, к которому они принадлежат. Эти данные используются для обучения персептрона методом градиентного спуска. После настройки веса сохраняются в текстового файл для передачи следующей программе.

Код программы описан в приложении В.

### 4.2.2 Блок-схема

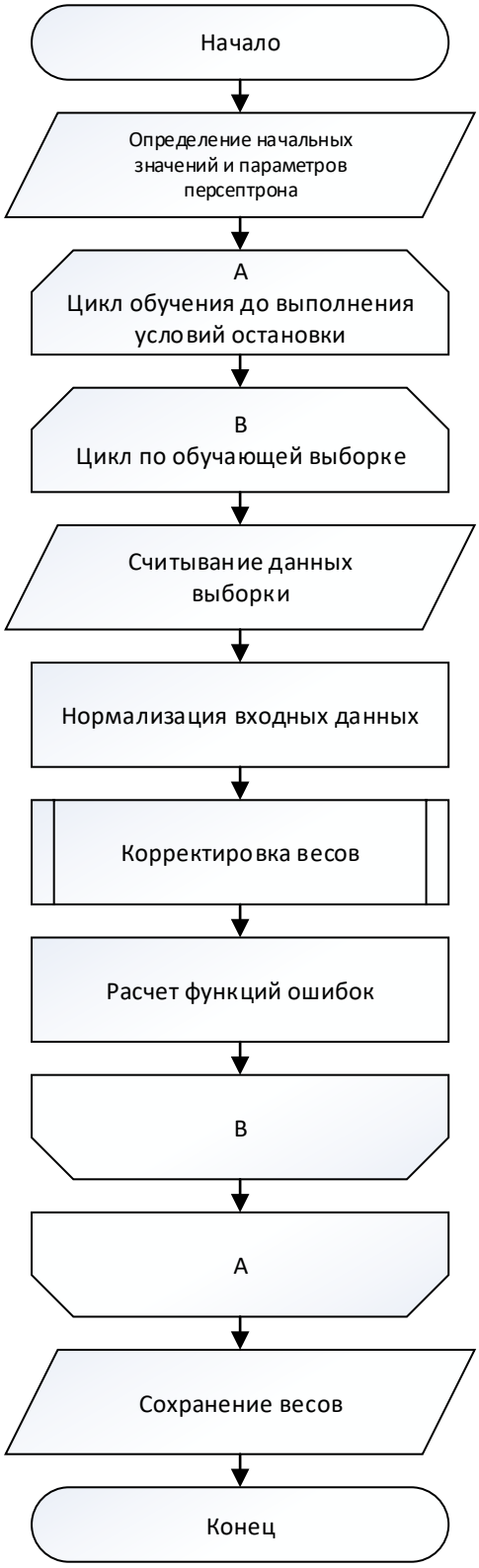


Рис 4.2. Схема программы для обучения нейронной сети

## 4.3 Классификация данных и построение карт

### 4.3.1 Описание

Когда мы получили данные моделирования мы можем приступить к задаче классификации. Здесь мы использовали результаты моделирования предыдущих программ и алгоритм классификации из главы 3. Данная программа позволяет построить две карты, соответствующие двум методам классификации.

Код программы описан в приложении Г.

### 4.3.2 Блок-схема

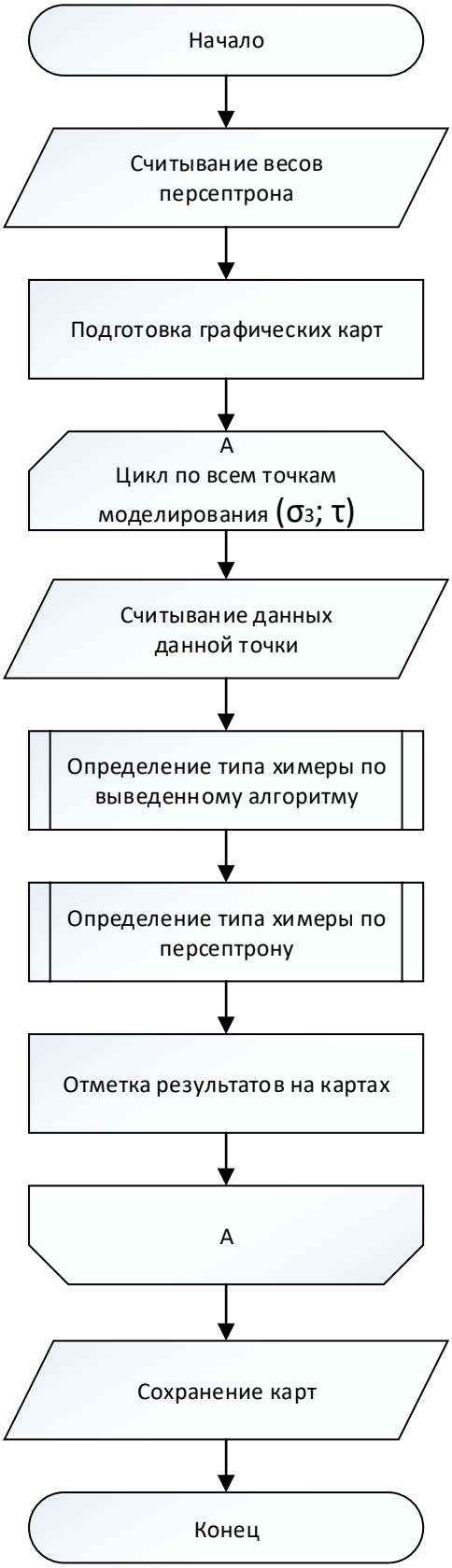


Рис 4.3. Схема программы для классификации данных и построения карт

# Заключение

В рамках этой работы рассмотрены несколько типов химер и на их основе выделены признаки классификаций. После их использованы на экспериментальных данных выведена правила, которые составляют вместе алгоритм классификации. При применении признаков к экспериментальных данных возникали сложности с точным определением типа. Данная проблема влияет на правила алгоритма и также не позволяет определить переходные состояния между типами химер.

Для сравнения эффективности алгоритма данная задача альтернативно была решена с помощью однослойного персептрона с обучением на основе экспериментальных данных методом градиентного спуска. Это позволило выделить переходные состояния, что дает возможность более тщательно исследовать данные области и продвинуться в исследовании химерных состояний и задач, на которые они ориентированы.

Так как результат применения нашего алгоритма на точках, используемых для обучения персептрона, совпадал с заданным нами типом, то это позволило сравнить оба метода. Как и оказалось, в большинстве случаев отклонения были незначительны, в тех случаях, когда признаки того или иного типа были явными, но в местах с преобладанием различных типов, где с сомнениями выбирался тот или иной тип химер, отклонения были относительно большими.

Таким образом использование выведенного алгоритма позволяет подбирать параметры, которые дают совпадение с результатами нашего выбора, но использование персептрона дает возможно лучше распознавать и исследовать переходные состояния между типами химер.

# Литература


1. Y. Kuramoto and D. Battogtokh, Nonlin. Phenom. in Complex Syst. 5, 380 (2002).
2. Daniel M Abrams and Steven H Strogatz. Chimera states for coupled oscillators. Phys. Rev. Lett., 93:174102, 2004.
3. I. Omelchenko, Y. L. Maistrenko, P. Hövel, and E. Schöll, Phys. Rev. Lett. 106, 234102 (2011).
4. I. Omelchenko, B. Riemenschneider, P. Hövel, Y. L. Maistrenko, and E. Schöll, Phys. Rev. E 85, 026212 (2012).
5. E. A. Martens, C. R. Laing, and S. H. Strogatz, Phys. Rev. Lett. 104, 044101 (2010).
6. O. E. Omelchenko, M. Wolfrum, S. Yanchuk, Y. L. Maistrenko, and O. Sudakov, Phys. Rev. E 85, 036210 (2012).
7. G. C. Sethia, A. Sen, and F. M. Atay, Phys. Rev. Lett. 100, 144102 (2008).
8. E. A. Martens, Chaos 20, 043122 (2010).
9. M. Wolfrum, O. E. Omel'chenko, S. Yanchuk, and Y. L. Maistrenko, Chaos 21, 013112 (2011).
10. M. R. Tinsley, S. Nkomo, and K. Showalter, Nat. Phys. 8, 662 (2012).
11. A. Hagerstrom, T. E. Murphy, R. Roy, P. Hövel, I. Omelchenko, and E. Schöll, Nat. Phys. 8, 658 (2012)
12. J. Yan, M. Bloom, S. C. Bae, E. Luijten, and S. Granick, Nature (London) 491, 578 (2012).
13. S. H. L. Klapp, Nature (London) 491, 530 (2012).
14. D. Battogtokh and Y. Kuramoto, Phys. Rev. E 61, 3227 (2000).
15. C. R. Laing and C. C. Chow, NeuralComput.13, 1473 (2001).
16. H. Sakaguchi, Phys. Rev. E 73, 031907 (2006).
17. S. Funahashi, C. J. Bruce, and P. S. Goldman-Rakic, J. Neurophysiol. 65, 1464 (1991).
18. N. C. Rattenborg, C. J. Amlaner, and S. L. Lima, Neurosci. Biobehav. Rev. 24, 817 (2000).
19. C. G. Mathews, J. A. Lesku, S. L. Lima, and C. J. Amlaner, Ethology 112, 286 (2006).
20. Matthias Wolfrum and Oleh E Omel'chenko. Chimera states are chaotic transients. Phys. Rev. E, 84:015201, 2011.
21. O E Omel'chenko. Coherence–incoherence patterns in a ring of non-locally coupled phase oscillators. Nonlinearity, 26(9):2469, 2013.
22. Edward Ott and Thomas M Antonsen. Long time evolution of phase oscillator systems. Chaos, 19(2):023117, 2009.
23. Yoshiki Kuramoto and Shin-ichiro Shima. Rotating spirals without phase singularity in reaction-diffusion systems. Prog. Theor. Phys. Supp., 150:115–125, 2003.
24. Shin-ichiro Shima and Yoshiki Kuramoto. Rotating spiral waves with phaserandomized core in nonlocally coupled oscillators. Phys. Rev. E, 69:036213, 2004.
25. Erik A Martens, Carlo R Laing, and Steven H Strogatz. Solvable model of spiral wave chimeras. Phys. Rev. Lett., 104:044101, 2010.
26. Oleh E OmelChenko, Matthias Wolfrum, Serhiy Yanchuk, Yuri L Maistrenko, and

Oleksandr Sudakov. Stationary patterns of coherence and incoherence in two-dimensional arrays of non-locally-coupled phase oscillators. Phys. Rev. E, 85:036210, 2012.
27. Mark J Panaggio and Daniel M Abrams. Chimera states on a flat torus. Phys. Rev. Lett., 110:094102, 2013.
28. Mark J Panaggio and Daniel M Abrams. Chimera states on a sphere. Phys. Rev. E, 91:022909, 2015.
29. Mark J Panaggio. Spot and Spiral Chimera States: Dynamical Patterns in Networks of Coupled Oscillators. PhD thesis, Northwestern University, 2014.
30. Murray Shanahan. Metastable chimera states in community-structured oscillator networks. Chaos, 20(1):013108, 2010.
31. M. J. Panaggio and D. M. Abrams. Chimera states: coexistence of coherence and incoherence in networks of coupled oscillators. Nonlinearity 28 R67. 2015.
32. FitzHugh R. (1961) Impulses and physiological states in theoretical models of nerve membrane. Biophysical J. 1:445–466
33. Nagumo J., Arimoto S., and Yoshizawa S. (1962) An active pulse transmission line simulating nerve axon. Proc. IRE. 50:2061–2070
34. D. P. Rosin, K. E. Callan, D. J. Gauthier, and E. Schöll, Europhys. Lett. 96, 34001 (2011).
35. M. Heinrich, T. Dahms, V. Flunkert, S. W. Teitsworth, and E. Schöll, New J. Phys. 12, 113030 (2010).
36. I. Omelchenko et al., When Nonlocal Coupling between Oscillators Becomes Stronger:Patched Synchrony or Multichimera States. Phys. Rev. Lett. 110 224101. 2013.
37. O. E. Omel'chenko, M. Wolfrum, and Y. L. Maistrenko, Phys. Rev. E 81, 065201(R) (2010).
38. D.Nikitin. Cluster synchronization and control of neural oscillatory networks.
39. В.А.Якубович. Некоторые общие теоретические принципы построения обучаемых опознающих систем. I // Вычислительная техника и вопросы программирования. Л.: Изд-во ЛГУ, 1965. С.3--71.
40. В.А.Якубович. О некоторых общих принципах построения обучающихся опознающих систем // Самообучающиеся автоматические системы, изд-во "Наука", М., 1966.
41. Фомин Я. А. Распознавание образов: теория и применения. — 2-е изд. — М.: ФАЗИС, 2012. — 429 с. — ISBN 978-5-7036-0130-4.
42. Розенблатт, Ф. Принципы нейродинамики: Перцептроны и теория механизмов мозга. — М.: Мир, 1965. — 480 с.
43. B.Widrow, M.Hoff Adaptive switching circuits. // IRE WESCON Convention Record, part 4, pp. 96-104. New York: Institute of Radio Engineers, 1960.

# Приложение А

## Значения экспериментальных данных для типа 1

| № | $\sigma_3$ | $\tau$ | Тип | $K_1$ | $K_2$ | $K_{extr}$ при значениях $r_{extr}$ | | | Δω |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $r_{extr}=3$ | $r_{extr}=6$ | $r_{extr}=10$ | |
| 1 | 0,05 | 0,3 | 1 | 1 | 0 | 5 | 3 | 3 | 0,13014 |
| 2 | 0,05 | 0,6 | 1 | 1 | 0 | 6 | 3 | 3 | 0,13933 |
| 3 | 0,05 | 0,9 | 1 | 1 | 0 | 14 | 4 | 4 | 0,06044 |
| 4 | 0,05 | 1,2 | 1 | 1 | 0 | 10 | 6 | 4 | 0,08116 |
| 5 | 0,05 | 1,5 | 1 | 1 | 0 | 8 | 4 | 3 | 0,10164 |
| 6 | 0,05 | 1,8 | 1 | 1 | 0 | 18 | 3 | 3 | 0,11347 |
| 7 | 0,05 | 2,1 | 1 | 1 | 0 | 11 | 4 | 4 | 0,0447 |
| 8 | 0,05 | 2,4 | 1 | 1 | 0 | 5 | 4 | 3 | 0,06198 |
| 9 | 0,05 | 2,7 | 1 | 1 | 0 | 5 | 4 | 4 | 0,11425 |
| 10 | 0,1 | 0,3 | 1 | 1 | 0 | 9 | 5 | 4 | 0,15199 |
| 11 | 0,1 | 0,6 | 1 | 1 | 0 | 11 | 3 | 3 | 0,05729 |
| 12 | 0,1 | 1,2 | 1 | 1 | 0 | 10 | 5 | 5 | 0,0865 |
| 13 | 0,1 | 1,5 | 1 | 1 | 0 | 8 | 3 | 3 | 0,12827 |
| 14 | 0,1 | 2,1 | 1 | 1 | 0 | 16 | 10 | 5 | 0,04051 |
| 15 | 0,1 | 2,4 | 1 | 1 | 0 | 17 | 8 | 6 | 0,03757 |
| 16 | 0,1 | 2,7 | 1 | 1 | 0 | 7 | 4 | 4 | 0,09533 |
| 17 | 0,1 | 3 | 1 | 1 | 0 | 17 | 5 | 3 | 0,08644 |
| 18 | 0,15 | 1,2 | 1 | 1 | 0 | 10 | 4 | 4 | 0,07041 |
| 19 | 0,15 | 1,8 | 1 | 1 | 0 | 12 | 5 | 4 | 0,03708 |
| 20 | 0,15 | 2,7 | 1 | 1 | 0 | 15 | 4 | 3 | 0,06398 |
| 21 | 0,15 | 3 | 1 | 0,967213 | 0,020492 | 21 | 12 | 9 | 0,1002 |
| 22 | 0,2 | 0,3 | 1 | 1 | 0 | 6 | 5 | 4 | 0,08569 |
| 23 | 0,2 | 0,6 | 1 | 1 | 0 | 7 | 4 | 4 | 0,08534 |
| 24 | 0,2 | 0,9 | 1 | 1 | 0 | 20 | 8 | 6 | 0,04416 |
| 25 | 0,2 | 1,5 | 1 | 0,971311 | 0,028689 | 42 | 12 | 9 | 0,04461 |
| 26 | 0,2 | 2,7 | 1 | 1 | 0 | 4 | 3 | 3 | 0,06913 |
| 27 | 0,2 | 3 | 1 | 0,971311 | 0,028689 | 27 | 14 | 6 | 0,08844 |
| 28 | 0,25 | 0,3 | 1 | 1 | 0 | 5 | 3 | 3 | 0,13526 |
| 29 | 0,25 | 0,6 | 1 | 1 | 0 | 7 | 4 | 2 | 0,0665 |
| 30 | 0,25 | 0,9 | 1 | 1 | 0 | 21 | 9 | 7 | 0,03484 |
| 31 | 0,25 | 1,5 | 1 | 0,987705 | 0,012295 | 31 | 10 | 7 | 0,03168 |
| 32 | 0,25 | 3 | 1 | 0,97541 | 0,02459 | 24 | 9 | 5 | 0,07676 |
| 33 | 0,3 | 0,3 | 1 | 1 | 0 | 5 | 3 | 3 | 0,12233 |
| 34 | 0,3 | 0,6 | 1 | 1 | 0 | 9 | 5 | 3 | 0,05524 |
| 35 | 0,3 | 0,9 | 1 | 1 | 0 | 24 | 12 | 9 | 0,02753 |
| 36 | 0,3 | 2,7 | 1 | 1 | 0 | 15 | 6 | 4 | 0,02818 |
| 37 | 0,3 | 3 | 1 | 0,979508 | 0,020492 | 24 | 6 | 5 | 0,06624 |
| 38 | 0,35 | 0,3 | 1 | 1 | 0 | 8 | 3 | 3 | 0,12905 |
| 39 | 0,35 | 0,6 | 1 | 1 | 0 | 14 | 7 | 5 | 0,04539 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 40 | 0,35 | 2,7 | 1 | 1 | 0 | 10 | 3 | 3 | 0,04123 |
| 41 | 0,35 | 3 | 1 | 0,979508 | 0,016393 | 36 | 8 | 4 | 0,05801 |
| 42 | 0,4 | 0,3 | 1 | 1 | 0 | 21 | 8 | 6 | 0,12869 |
| 43 | 0,4 | 0,6 | 1 | 1 | 0 | 32 | 15 | 7 | 0,03884 |
| 44 | 0,4 | 2,7 | 1 | 1 | 0 | 44 | 12 | 6 | 0,04496 |
| 45 | 0,4 | 3 | 1 | 0,967213 | 0,02459 | 46 | 19 | 9 | 0,05378 |
| 46 | 0,45 | 0,6 | 1 | 1 | 0 | 13 | 9 | 4 | 0,03118 |
| 47 | 0,45 | 2,7 | 1 | 1 | 0 | 11 | 5 | 4 | 0,03071 |
| 48 | 0,45 | 3 | 1 | 0,967213 | 0,02459 | 44 | 16 | 9 | 0,04702 |
| 49 | 0,5 | 0,3 | 1 | 1 | 0 | 6 | 3 | 3 | 0,10043 |
| 50 | 0,5 | 3 | 1 | 0,967213 | 0,02459 | 44 | 14 | 8 | 0,04237 |
| Max | | | | 1 | 0,028689 | 46 | 19 | 9 | 0,15199 |
| Min | | | | 0,967213 | 0 | 4 | 3 | 2 | 0,02753 |
| Среднее арифметическое | | | | 0,995133 | 0,004184 | 16,35417 | 6,6875 | 4,666667 | 0,073197 |

Значения экспериментальных данных для типа 2

| № | $\sigma_3$ | $\tau$ | Тип | $K_1$ | $K_2$ | $K_{extr}$ при значениях $r_{extr}$ | | | Δω |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $r_{extr}=3$ | $r_{extr}=6$ | $r_{extr}=10$ | |
| 1 | 0,05 | 3 | 2 | 1 | 0 | 64 | 27 | 15 | 0,00819 |
| 2 | 0,1 | 1,8 | 2 | 1 | 0 | 53 | 27 | 11 | 0,01007 |
| 3 | 0,15 | 0,3 | 2 | 1 | 0 | 27 | 12 | 7 | 0,02511 |
| 4 | 0,15 | 0,6 | 2 | 1 | 0 | 39 | 13 | 8 | 0,03337 |
| 5 | 0,15 | 2,1 | 2 | 1 | 0 | 18 | 7 | 6 | 0,02337 |
| 6 | 0,15 | 2,4 | 2 | 1 | 0 | 28 | 10 | 7 | 0,02983 |
| 7 | 0,2 | 1,8 | 2 | 1 | 0 | 20 | 6 | 3 | 0,02725 |
| 8 | 0,2 | 2,1 | 2 | 1 | 0 | 15 | 8 | 8 | 0,01552 |
| 9 | 0,2 | 2,4 | 2 | 1 | 0 | 26 | 13 | 8 | 0,02511 |
| 10 | 0,25 | 1,8 | 2 | 1 | 0 | 23 | 9 | 6 | 0,0208 |
| 11 | 0,25 | 2,1 | 2 | 1 | 0 | 20 | 9 | 8 | 0,0111 |
| 12 | 0,25 | 2,4 | 2 | 1 | 0 | 29 | 21 | 21 | 0,02309 |
| 13 | 0,25 | 2,7 | 2 | 1 | 0 | 23 | 7 | 4 | 0,02192 |
| 14 | 0,3 | 1,5 | 2 | 0,983607 | 0,016393 | 34 | 15 | 11 | 0,02403 |
| 15 | 0,3 | 1,8 | 2 | 1 | 0 | 22 | 11 | 9 | 0,01598 |
| 16 | 0,3 | 2,1 | 2 | 1 | 0 | 19 | 10 | 8 | 0,00969 |
| 17 | 0,3 | 2,4 | 2 | 1 | 0 | 21 | 7 | 3 | 0,02086 |
| 18 | 0,35 | 0,9 | 2 | 1 | 0 | 24 | 13 | 8 | 0,0225 |
| 19 | 0,35 | 1,5 | 2 | 0,983607 | 0,016393 | 33 | 15 | 10 | 0,01921 |
| 20 | 0,35 | 1,8 | 2 | 1 | 0 | 24 | 9 | 7 | 0,01309 |
| 21 | 0,35 | 2,1 | 2 | 1 | 0 | 28 | 12 | 9 | 0,00723 |
| 22 | 0,35 | 2,4 | 2 | 1 | 0 | 14 | 8 | 4 | 0,01993 |
| 23 | 0,4 | 0,9 | 2 | 0,979508 | 0,020492 | 43 | 17 | 9 | 0,02133 |
| 24 | 0,4 | 1,5 | 2 | 0,963115 | 0,036885 | 46 | 20 | 11 | 0,01639 |
| 25 | 0,4 | 1,8 | 2 | 1 | 0 | 48 | 27 | 13 | 0,01142 |
| 26 | 0,4 | 2,1 | 2 | 1 | 0 | 36 | 14 | 9 | 0,01007 |
| 27 | 0,4 | 2,4 | 2 | 1 | 0 | 32 | 16 | 11 | 0,02182 |
| 28 | 0,45 | 0,3 | 2 | 1 | 0 | 34 | 15 | 9 | 0,02004 |
| 29 | 0,45 | 0,9 | 2 | 0,946721 | 0,04918 | 46 | 20 | 14 | 0,01766 |
| 30 | 0,45 | 1,5 | 2 | 0,963115 | 0,036885 | 35 | 16 | 10 | 0,0134 |
| 31 | 0,45 | 1,8 | 2 | 1 | 0 | 23 | 12 | 9 | 0,00886 |
| 32 | 0,45 | 2,1 | 2 | 1 | 0 | 19 | 11 | 8 | 0,00648 |
| 33 | 0,45 | 2,4 | 2 | 1 | 0 | 24 | 9 | 7 | 0,01899 |
| 34 | 0,5 | 0,6 | 2 | 1 | 0 | 19 | 8 | 6 | 0,02704 |
| 35 | 0,5 | 1,2 | 2 | 0,922131 | 0,077869 | 45 | 23 | 16 | 0,01721 |
| 36 | 0,5 | 1,5 | 2 | 0,938525 | 0,061475 | 52 | 26 | 15 | 0,01203 |
| 37 | 0,5 | 1,8 | 2 | 1 | 0 | 26 | 11 | 9 | 0,00737 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 38 | 0,5 | 2,1 | 2 | 1 | 0 | 19 | 10 | 8 | 0,00619 |
| 39 | 0,5 | 2,4 | 2 | 1 | 0 | 21 | 14 | 7 | 0,01812 |
| 40 | 0,5 | 2,7 | 2 | 1 | 0 | 10 | 5 | 2 | 0,02799 |
| Max | | | | 1 | 0,077869 | 64 | 27 | 21 | 0,03337 |
| Min | | | | 0,922131 | 0 | 10 | 5 | 2 | 0,00619 |
| Среднее арифметическое | | | | 0,992008 | 0,007889 | 29,55 | 13,575 | 8,85 | 0,017742 |

Значения экспериментальных данных для типа 3

| № | $\sigma_3$ | $\tau$ | Тип | $K_1$ | $K_2$ | $K_{extr}$ при значениях $r_{extr}$ | | | Δω |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $r_{extr} = 3$ | $r_{extr} = 6$ | $r_{extr} = 10$ | |
| 1 | 0,1 | 0,9 | 3 | 0,901639 | 0,098361 | 44 | 23 | 12 | 0,18787 |
| 2 | 0,15 | 0,9 | 3 | 0,913934 | 0,086066 | 46 | 31 | 10 | 0,18003 |
| 3 | 0,15 | 1,5 | 3 | 0,844262 | 0,155738 | 54 | 26 | 14 | 0,06817 |
| 4 | 0,2 | 1,2 | 3 | 0,905738 | 0,094262 | 53 | 23 | 14 | 0,08707 |
| 5 | 0,25 | 1,2 | 3 | 0,504098 | 0,495902 | 51 | 27 | 13 | 0,02827 |
| 6 | 0,3 | 1,2 | 3 | 0,733607 | 0,266393 | 67 | 25 | 16 | 0,03001 |
| 7 | 0,35 | 1,2 | 3 | 0,790984 | 0,209016 | 59 | 30 | 16 | 0,03156 |
| 8 | 0,4 | 1,2 | 3 | 0,848361 | 0,151639 | 56 | 30 | 15 | 0,02582 |
| 9 | 0,45 | 1,2 | 3 | 0,877049 | 0,122951 | 49 | 23 | 12 | 0,02089 |
| 10 | 0,5 | 0,9 | 3 | 0,905738 | 0,094262 | 42 | 23 | 12 | 0,01797 |
| Max | | | | 0,913934 | 0,495902 | 67 | 31 | 16 | 0,18787 |
| Min | | | | 0,504098 | 0,086066 | 42 | 23 | 10 | 0,01797 |
| Среднее арифметическое | | | | 0,822541 | 0,177459 | 52,1 | 26,1 | 13,4 | 0,067766 |

# Приложение Б

## Constants.h

```
#pragma once

const double M_PI=3.1415926;
const int DELAY_COUNTS = 3000;

const int N = 244;                                          // количество нейронов
const int  Fractal_level = 5;

const double r = 0.35;                                      // радиус связи
const double a = 0.5;                                       // система в осцилляторном
режиме (самовозбуждение)
const double eps = 0.05;                                // определяет разницу масштабов
времени для активатора и ингибитора
const double phi = -M_PI/2 + 0.1;                   // угол поворота
const double dt = 0.001;                                //
const double sigma12 = 0.1;                             // сила связи



const double T = 1000;
const double T_paint = 900;                             // T - 100
const double T_omega = 400;                             // на глаз хз что


const int pixel_width = 1920;                           //
const int pixel_height = 1080;                          //

////////////TEMP/////////////
const int m = int(T / dt);                              //
const int rN = int(r * N);                              // ограничение на расстояние связи
(P)

const int G1_size = 2 * rN;                             // 2*R
```

# Main.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <ctime>
#include <vector>
#include <map>
#include <string>
#include <thread>
#include <iomanip>

#include <omp.h>

#include "Constants.h"
using namespace std;



vector<int> parent(2 * N);
vector<int> rang(2 * N);

class lp
{
public:
	static int launches;
	double sigma3;
	double tau;
	lp(double sigma3 = 0, double tau = 0)
		: sigma3(sigma3), tau(tau) {}
}params[10000];


int delayed_counter(int pointer, int delay_value)
{
	return (pointer - delay_value) >= 0 ? pointer - delay_value : pointer -
delay_value + DELAY_COUNTS;
}

void produce_derivatives(const vector<double> & u, const vector<double> & v, const
vector<double> & u_del, vector<double> & du, vector<double> & dv, bool is_delayed, double
sigma3)
{
	double R[2][2] = { { cos(phi),-sin(phi) },{ sin(phi),cos(phi) } };

	///SUM_ON_SEGMENTS

	vector<double> sums_u(N, 0), sums_v(N, 0);
	sums_u[0] = u[0];
	sums_v[0] = v[0];
	for (int i = 1; i < N; i++) {
		sums_u[i] = sums_u[i - 1] + u[i];
		sums_v[i] = sums_v[i - 1] + v[i];
	}

	///FIRST_LAYER

	for (int i = 0; i < N; i++) {
		double delta_u;
		double delta_v;
		if (i + rN >= N) {
			delta_u = sums_u[N - 1] + sums_u[i + rN - N] - sums_u[i - rN - 1];
			delta_v = sums_v[N - 1] + sums_v[i + rN - N] - sums_v[i - rN - 1];
```

```
		}
		else if (i - rN <= 0) {
			delta_u = sums_u[N - 1] + sums_u[i + rN] - sums_u[i - rN - 1 + N];
			delta_v = sums_v[N - 1] + sums_v[i + rN] - sums_v[i - rN - 1 + N];
		}
		else {
			delta_u = sums_u[i + rN] - sums_u[i - rN - 1];
			delta_v = sums_v[i + rN] - sums_v[i - rN - 1];
		}
		delta_u = (delta_u - u[i]) / G1_size - u[i];                        //
почему делим на 2P
		delta_v = (delta_v - v[i]) / G1_size - v[i];

		du[i] = (R[0][0] * delta_u + R[0][1] * delta_v) * sigma12 / eps;
		dv[i] = (R[1][0] * delta_u + R[1][1] * delta_v) * sigma12;
	}


	///SUM_ON_TRIPLETS

	vector<vector<double> > sum_tri_u(Fractal_level + 1, vector<double>(N, 0));
	vector<vector<double> > sum_tri_v(Fractal_level + 1, vector<double>(N, 0));

	for (int i = 0; i < N; i++) {
		sum_tri_u[0][i] = u[N + i];
		sum_tri_v[0][i] = v[N + i];
	}
	int pow3 = 1;
	int G2_size = 1;
	for (int j = 1; j <= Fractal_level; j++) {
		for (int i = 0; i < N; i++) {
			int i_minus = (i - pow3) < 0 ? i - pow3 + N : i - pow3;
			int i_plus = (i + pow3) >= N ? i + pow3 - N : i + pow3;
			sum_tri_u[j][i] = sum_tri_u[j - 1][i_minus] + sum_tri_u[j -
1][i_plus]; // почему так
			sum_tri_v[j][i] = sum_tri_v[j - 1][i_minus] + sum_tri_v[j -
1][i_plus];
		}
		pow3 *= 3;
		G2_size *= 2;
				// почему на 2
	}

	///SECOND_LAYER

	for (int i = 0; i < N; i++) {
		int i2 = (i < N / 2) ? i - N / 2 + N : i - N / 2;
		double delta_u = sum_tri_u[Fractal_level][i2] / G2_size - u[N + i];
		double delta_v = sum_tri_v[Fractal_level][i2] / G2_size - v[N + i];

		du[N + i] = (R[0][0] * delta_u + R[0][1] * delta_v) * sigma12 / eps;

		double temp = du[N + i];

		dv[N + i] = (R[1][0] * delta_u + R[1][1] * delta_v) * sigma12;
	}

	///OTHER_SUMMANDS

	if (is_delayed) {
		for (int i = 0; i < N; i++) {
			du[i] += (u[i] - u[i] * u[i] * u[i] / 3 - v[i] + sigma3 * (u_del[i +
N] - u[i])) / eps;
			dv[i] += u[i] + a;
```

```
		}

		for (int i = N; i < 2 * N; i++) {
			du[i] += (u[i] - u[i] * u[i] * u[i] / 3 - v[i] + sigma3 * (u_del[i -
N] - u[i])) / eps;
			dv[i] += u[i] + a;
		}
	}
	else {
		for (int i = 0; i < N; i++) {
			du[i] += (u[i] - u[i] * u[i] * u[i] / 3 - v[i] + sigma3 * (u[i + N] -
u[i])) / eps;
			dv[i] += u[i] + a;
		}

		for (int i = N; i < 2 * N; i++) {
			du[i] += (u[i] - u[i] * u[i] * u[i] / 3 - v[i] + sigma3 * (u[i - N] -
u[i])) / eps;
			dv[i] += u[i] + a;
		}
	}

}

void add_omega(double time, vector<double> & omega, int & omega_counts, const
vector<double> & u, const vector<double> & v, const vector<double> & du, const
vector<double> & dv)
{
	if (time < T_omega) return;

	vector<double> z(2 * N, 0);
	for (int i = 0; i < 2 * N; i++) {
		z[i] = (dv[i] * u[i] - du[i] * v[i]) / (u[i] * u[i] + v[i] * v[i]);
	}

	for (int i = 0; i < omega.size(); ++i)
	{
		omega[i] *= omega_counts;
	}

	for (int i = 0; i < omega.size(); ++i)
	{
		omega[i] += z[i];
	}

	omega_counts++;

	for (int i = 0; i < omega.size(); ++i)
	{
		omega[i] *= 1.0 / omega_counts;
	}

}

void Heun_method(vector<double> & new_u, vector<double> & new_v, double dt,
	const vector<double> & u, const vector<double> & v,
	const vector<double> & u_d1, const vector<double> & u_d2, bool is_delayed, double
sigma3, vector<double> & new_du, vector<double> & new_dv)
{
	vector<double> temp_u(2 * N, 0);
	vector<double> temp_v(2 * N, 0);
	vector<double> du1(2 * N, 0), dv1(2 * N, 0);
	vector<double> du2(2 * N, 0), dv2(2 * N, 0);
```

```
	produce_derivatives(u, v, u_d1, du1, dv1, is_delayed, sigma3);	//0*dt
	for (int i = 0; i < 2 * N; i++) {
		temp_u[i] = u[i] + dt * du1[i];
		temp_v[i] = v[i] + dt * dv1[i];
	}
	produce_derivatives(temp_u, temp_v, u_d2, du2, dv2, is_delayed, sigma3);	//1*dt
	for (int i = 0; i < 2 * N; i++) {
		new_du[i] = (du1[i] + du2[i]) / 2;
		new_dv[i] = (dv1[i] + dv2[i]) / 2;
		new_u[i] = u[i] + dt * new_du[i];
		new_v[i] = v[i] + dt * new_dv[i];
	}
}



void solver(double sigma3, double tau, const vector<double> & init_u, const
vector<double> & init_v, int ii)
{
	vector<vector<double> > hist_u(DELAY_COUNTS, vector<double>(2 * N, 0));
	vector<double> hist_v(2 * N, 0);
	vector<double> du(2 * N, 0);
	vector<double> dv(2 * N, 0);
	vector<double> omega(2 * N, 0);

	for (int i = 0; i < omega.size(); ++i)
		omega[i] = 0;

	int omega_counts = 0;

	int del_val = int(tau / dt);
	if (del_val == 0) del_val = 1;

	int pointer = 0;
	int filled = 0;
	hist_u[pointer] = init_u;
	hist_v = init_v;
	pointer++;
	filled++;

	long long prc = 0;
	double T_current = 0;
	int pixel_current = 0;
	long long starttime = clock();

	double time_start;

	for (long long i = 0; i < m; i++)
	{

		if (filled == del_val) {
			int p0 = delayed_counter(pointer, 1);
			int p1 = delayed_counter(pointer, del_val);
			int p2 = delayed_counter(pointer, del_val - 1);
			Heun_method(hist_u[pointer], hist_v, dt,
				hist_u[p0], hist_v,
				hist_u[p1], hist_u[p2], true, sigma3, du, dv);
		}
		else {
			int p0 = delayed_counter(pointer, 1);
			Heun_method(hist_u[pointer], hist_v, dt,
				hist_u[p0], hist_v,
				hist_u[0], hist_u[0], false, sigma3, du, dv);
			filled++;
```

```
		}


		if (prc < 100 * i / m) {
			prc++;
			cerr << prc << " " << (clock() - starttime) / double(CLOCKS_PER_SEC)
<< endl;
		}

		double temp_u = hist_u[pointer][0];

		add_omega(T * i / m, omega, omega_counts, hist_u[pointer], hist_v, du, dv);
		pointer++;
		if (pointer == DELAY_COUNTS) pointer = 0;
	}

	char text_string[60];
	sprintf_s(text_string, 60, "Data\\%.3f %.3f.txt", params[ii].sigma3,
params[ii].tau);

	ofstream Log(text_string, ios_base::out);

	for (int i = 0; i < 2*N; ++i)
	{
		Log << omega[i] << ' ';
	}

	Log.close();

}



int main()
{
	vector<double> init_u(2 * N, 0);
	vector<double> init_v(2 * N, 0);

	double tmp;
	ifstream in("InitialConditions_chimera_N244.dat", fstream::in);
	for (int i = 0; i < N; i++) in >> init_u[i] >> init_v[i] >> tmp;
	in.close();

	for (int i = 0; i < N; i++)
	{
		init_u[N + i] = 1.3321;
		init_v[N + i] = 0.5745;
	}


	fstream result("Set_points.txt", fstream::in);

	double eps_myset = 0.00001;
	double p1, p2;
	int d;

	vector<pair<double, double>> myset;

	while (result >> p1 >> p2 >> d)
	{
		myset.push_back(make_pair(p1, p2));
	}

	result.close();
```

```
	freopen("Set_points.txt", "a", stdout);

	lp::launches = 0;
	//int count_point = 40;
	int count_point = 40;
	double sigma3_length = 0.5, tau_length = 3;
	double sigma3_step = sigma3_length / (count_point), tau_step = tau_length /
(count_point);

	int number = 1;

	//for (double sigma3_cur = sigma3_step; sigma3_cur  <= sigma3_length + eps_myset;
sigma3_cur += sigma3_step)
	//{
	double sigma3_cur = sigma3_step*number;

		for (double tau_cur = tau_step; tau_cur <= tau_length + eps_myset; tau_cur
+= tau_step)
		{
			bool flag_myset = false;
			for (int i = 0; i < myset.size(); ++i)
			{
				if (abs(myset[i].first - sigma3_cur) < eps_myset &&
abs(myset[i].second - tau_cur) < eps_myset)
				{
					flag_myset = true;
					break;
				}
			}
			if (flag_myset) continue;
			params[lp::launches++] = lp(sigma3_cur, tau_cur);
		}

	//}



	omp_set_num_threads(6);

	int count_result = 0;


	cerr << lp::launches << ' ' << number<<'\n';


#pragma omp parallel for  firstprivate(init_u,init_v, params) shared(count_result)
	for (int i = 0; i < lp::launches; i++)
	{


		solver(params[i].sigma3, params[i].tau, init_u, init_v, i);

		cout << params[i].sigma3 << ' ' << params[i].tau << '\n';

		count_result++;
	}


	return 0;
}
```

# Приложение В

## Perceptron.h

```
#pragma once
#include<vector>
#include<cstdio>

using namespace std;

struct Perceptron
{
	vector<vector<pair<double, int> > > w;

	double coeff_study;
	double eps_error;

	vector<double> max_x;
	double max_y;

	double Study(vector<double> x, vector<double> y);
	void Norm_x(vector<double> &x);
	void Norm_y(vector<double> &y);
	double Error(double y, double y_true);
	vector<double> Answer(vector<double>& x);

};
```

# Perceptron.cpp

```
#include "Perceptron.h"

double Perceptron::Study(vector<double> x, vector<double> y)
{
	Norm_x(x);
	Norm_y(y);

	double max_error = -1;

	for (int i = 0; i < w.size(); ++i)
	{
		double my_y = 0;
		for (int j = 0; j < w[i].size(); ++j)
		{
			my_y += w[i][j].first*x[w[i][j].second];
		}

		double curr_error = Error(my_y, y[i]);

		for (int j = 0; j < w[i].size(); ++j)
		{
			double lol = w[i][j].first;

			w[i][j].first += coeff_study*curr_error*x[w[i][j].second];

			lol = w[i][j].first;

			if (abs(curr_error) > max_error)
			{
				max_error = abs(curr_error);
			}
		}

		double new_y = 0;
		for (int j = 0; j < w[i].size(); ++j)
		{
			new_y += w[i][j].first*x[w[i][j].second];
		}
		double error_y = Error(new_y, y[i]);

		double err = abs(curr_error) - abs(error_y);

		if (err < 0)
		{
			double lol = 1;
		}
	}

	return max_error;
}

void Perceptron::Norm_x(vector<double> &x)
{
	for (int i = 0; i < x.size(); ++i)
	{
		x[i] /= max_x[i];
		if (x[i] < 0) x[i] = 0;
		else if (x[i] > 1) x[i] = 1;
	}
}

void Perceptron::Norm_y(vector<double> &y)
{
```

```
	for (int i = 0; i < y.size(); ++i)
	{
		y[i] /= max_y;

		if (y[i] < 0) y[i] = 0;
		else if (y[i] > 1) y[i] = 1;
	}
}

double Perceptron::Error(double y, double y_true)
{
	if (y > 1) y = 1;
	else if (y < 0) y = 0;
	return y_true - y;
}

vector<double> Perceptron::Answer(vector<double>& x)
{
	Norm_x(x);

	vector<double> y(w.size(), 0);

	for (int i = 0; i < w.size(); ++i)
	{

		for (int j = 0; j < w[i].size(); ++j)
		{
			y[i] += w[i][j].first*x[w[i][j].second];
		}

		if (y[i] > 1)
			y[i] = 1;
		else if (y[i] < 0)
			y[i] = 0;
	}

	return y;

}
```

# Main.cpp

```
#include <iostream>
#include <cmath>
#include <iomanip>
#define _CRT_SECURE_NO_WARNINGS
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
#include <algorithm>
#include <map>
#include <set>
#include <string>
#include <time.h>
#include <omp.h>


#include "Perceptron.h"

using namespace std;

Perceptron Perc;


void InitAndStudyPerceptron()
{

	// Init
	Perc.max_x.resize(4);

	Perc.max_x[0] = 1;
	Perc.max_x[1] = 1;
	Perc.max_x[2] = 30;
	Perc.max_x[3] = 0.05;

	Perc.max_y = 1;


	Perc.eps_error = 0.0000001;
	Perc.coeff_study = 0.1;


	Perc.w.resize(3);

	Perc.w[0].resize(4);
	Perc.w[1].resize(4);
	Perc.w[2].resize(2);

	Perc.w[0][0] = make_pair(0.001, 0);
	Perc.w[0][1] = make_pair(0.001, 1);
	Perc.w[0][2] = make_pair(0.001, 2);
	Perc.w[0][3] = make_pair(0.001, 3);

	Perc.w[1][0] = make_pair(0.001, 0);
	Perc.w[1][1] = make_pair(0.001, 1);
	Perc.w[1][2] = make_pair(0.001, 2);
	Perc.w[1][3] = make_pair(0.001, 3);

	Perc.w[2][0] = make_pair(0.001, 0);
	Perc.w[2][1] = make_pair(0.001, 1);


	double last_max_error = 0, max_error = 10, last_common_error=0, common_error=10,
curr_error;
```

```
	double max_sigma3, max_tau;

	int it;

	for (it = 0; (abs(max_error - last_max_error)> Perc.eps_error || abs(common_error
- last_common_error)> Perc.eps_error) && it <= 1000000; ++it)
	{

		last_common_error = common_error;
		last_max_error = max_error;

		common_error = 0;
		max_error = 0;
		ifstream Log_study("Study_point.txt", ios_base::in);

		double K1, K2, K_extr, Delta_omega;
		int read_type_chimera;

		int count_point = 0;

		double read_sigma, read_tau;

		while (Log_study >> read_sigma>> read_tau>>K1 >> K2 >> K_extr >>
Delta_omega >> read_type_chimera)
		{
			vector<double> type(3, 0);
			type[read_type_chimera - 1] = 1;

			curr_error = Perc.Study({ K1, K2, K_extr, Delta_omega }, type);

			if (curr_error >= 0.95)
			{
				double lol = 1;
			}

			if (curr_error >= max_error)
			{
				max_error = curr_error;
			}
			common_error += curr_error;
			count_point++;
		}

		Log_study.close();

		common_error /= count_point;


		cout << it << ' ' << max_error << ' ' << common_error << '\n';



	}

	freopen("Error.txt", "w", stdout);

	cout << it << ' ' << max_error << ' ' << common_error << '\n';
}


int main()
{
	InitAndStudyPerceptron();
```

```
	freopen("Weight.txt", "w", stdout);

	for (int i = 0; i < Perc.w.size(); ++i)
	{
		for (int j = 0; j < Perc.w[i].size(); ++j)
		{
			cout << Perc.w[i][j].first << ' ' << Perc.w[i][j].second << '\n';
		}
	}


	return 0;
}
```

# Приложение В

## Perceptron.h

```
#pragma once
#include<vector>
#include<cstdio>

using namespace std;

struct Perceptron
{
	vector<vector<pair<double, int> > > w;

	double coeff_study;
	double eps_error;

	vector<double> max_x;
	double max_y;

	double Study(vector<double> x, vector<double> y);
	void Norm_x(vector<double> &x);
	void Norm_y(vector<double> &y);
	double Error(double y, double y_true);
	vector<double> Answer(vector<double>& x);

};
```

# Perceptron.cpp

```
#include "Perceptron.h"

double Perceptron::Study(vector<double> x, vector<double> y)
{
	Norm_x(x);
	Norm_y(y);

	double max_error = -1;

	for (int i = 0; i < w.size(); ++i)
	{
		double my_y = 0;
		for (int j = 0; j < w[i].size(); ++j)
		{
			my_y += w[i][j].first*x[w[i][j].second];
		}

		double curr_error = Error(my_y, y[i]);

		for (int j = 0; j < w[i].size(); ++j)
		{
			double lol = w[i][j].first;

			w[i][j].first += coeff_study*curr_error*x[w[i][j].second];

			lol = w[i][j].first;

			if (abs(curr_error) > max_error)
			{
				max_error = abs(curr_error);
			}
		}

		double new_y = 0;
		for (int j = 0; j < w[i].size(); ++j)
		{
			new_y += w[i][j].first*x[w[i][j].second];
		}
		double error_y = Error(new_y, y[i]);

		double err = abs(curr_error) - abs(error_y);

		if (err < 0)
		{
			double lol = 1;
		}
	}

	return max_error;
}

void Perceptron::Norm_x(vector<double> &x)
{
	for (int i = 0; i < x.size(); ++i)
	{
		x[i] /= max_x[i];
		if (x[i] < 0) x[i] = 0;
		else if (x[i] > 1) x[i] = 1;
	}
}

void Perceptron::Norm_y(vector<double> &y)
{
```

```
	for (int i = 0; i < y.size(); ++i)
	{
		y[i] /= max_y;

		if (y[i] < 0) y[i] = 0;
		else if (y[i] > 1) y[i] = 1;
	}
}

double Perceptron::Error(double y, double y_true)
{
	if (y > 1) y = 1;
	else if (y < 0) y = 0;
	return y_true - y;
}

vector<double> Perceptron::Answer(vector<double>& x)
{
	Norm_x(x);

	vector<double> y(w.size(), 0);

	for (int i = 0; i < w.size(); ++i)
	{

		for (int j = 0; j < w[i].size(); ++j)
		{
			y[i] += w[i][j].first*x[w[i][j].second];
		}

		if (y[i] > 1)
			y[i] = 1;
		else if (y[i] < 0)
			y[i] = 0;
	}

	return y;

}
```

# Constants.h

```
#pragma once

const int N = 244;

const int pixel_width = 1080;
const int pixel_height = 1080;

const double sigma3_start = 0, sigma3_end = 0.5, sigma3_add_g = 0.01;
const double tau_start = 0, tau_end = 3, tau_add_g = 0.1;

const double count_step = 5;

const double sigma3_step = 0.1;
const double tau_step = 0.5;

const double up_ind = 0.15, down_ind = 0.1;
const double left_ind = 0.125, right_ind = 0.125;


const double dash_width = 0.01;
```

# Main.cpp

```
#include <iostream>
#include <cmath>
#include <iomanip>
#define _CRT_SECURE_NO_WARNINGS
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <vector>
#include <algorithm>
#include <map>
#include <set>
#include <string>
#include <time.h>
#include <omp.h>
#include <opencv2\opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iomanip>

#include "Constants.h"
#include "Perceptron.h"

using namespace std;
using namespace cv;

Perceptron Perc;

vector<int> parent(2 * N);
vector<int> rang(2 * N);

void setPixel(IplImage* img, int x, int y, int channel, int val)
{
	*(img->imageData + y * img->widthStep + x * img->nChannels + channel) = val;
}

bool cmp(pair<int, double> a, pair<int, double> b)
{
	return a > b;
}

void make_set(int v) {
	parent[v] = v;
	rang[v] = 0;
}

int find_set(int v) {
	if (v == parent[v])
		return v;
	return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
	a = find_set(a);
	b = find_set(b);
	if (a != b) {
		if (rang[a] < rang[b])
			swap(a, b);
		parent[b] = a;
		if (rang[a] == rang[b])
			++rang[a];
	}
}
```

```
void analysis_chimera_type(vector<double> omega, double& K1, double& K2, double& K_extr,
double& Delta_omega)
{
	CvFont font;
	cvInitFont(&font, CV_FONT_HERSHEY_COMPLEX, 0.4 * pixel_height / 720, 0.4 *
pixel_height / 720, 0, 1, CV_AA);
	char text_string[60];
	CvPoint text_point;


	for (int i = 0; i < N; ++i)
	{
		double lol = omega[i];
	}

	//Type 3

	double eps_dist_y = 0.05;

	for (int i = 0; i < N; i++)
	{
		make_set(i);
	}

	for (int i = 0; i < N; ++i)
	{
		for (int j = 0; j < N; ++j)
		{
			if (i == j) continue;

			if (abs(omega[i] - omega[j]) <= eps_dist_y)
			{
				union_sets(i, j);
			}
		}
	}


	vector<pair<int, double> > count(N, make_pair(0, 0));


	for (int i = 0; i < N; ++i)
	{
		int current_set = find_set(i);

		if (count[current_set].first) continue;

		count[current_set].first++;

		double y_max = -10, y_min = 10;

		for (int j = 0; j < N; ++j)
		{
			if (find_set(j) == current_set && i != j)
			{
				count[current_set].first++;


				if (y_max < omega[j])
					y_max = omega[j];
				if (y_min > omega[j])
					y_min = omega[j];
			}
		}
```

```
			count[current_set].second = y_max - y_min;
		}

		sort(count.begin(), count.end(), cmp);



		K1 = count[0].first*1.0 / N;
		K2 = count[1].first*1.0 / N;
		Delta_omega = count[0].second;


		//Type 1

		for (int count_for = 0; count_for < 1; count_for++)
		{
			for (int i = 1; i < N - 1; ++i)
			{
				int current_set = find_set(i);

				int left_ind, right_ind;

				for (left_ind = i - 1; left_ind >= 0 && find_set(left_ind) !=
current_set; left_ind--);

				if (left_ind < 0) continue;

				for (right_ind = i + 1; right_ind < N && find_set(right_ind) !=
current_set; right_ind++);

				if (left_ind >= N) continue;

				omega[i] = (omega[left_ind] + omega[right_ind]) / 2;

			}
		}

		K_extr = 0;

		int rad_exp = 6;

		double eps_exp = 0.001;

		for (int i = 0; i < N; ++i)
		{
			bool flag_exp = true;
			if (i - rad_exp < 0)
			{
				for (int j = N-(i - rad_exp); j <  N; ++j)
				{
					if (omega[i] < omega[j] && i != j)
					{
						flag_exp = false;
						break;
					}
				}
			}
			if (i + rad_exp >= N)
			{
				for (int j = 0; j <= i + rad_exp - N; ++j)
				{
					if (omega[i] < omega[j] && i != j)
					{
```

```
                                flag_exp = false;
                                break;
                        }
                }
        }

        for (int j = max(i - rad_exp, 0); j < min(i + rad_exp, N); ++j)
        {
                if (omega[i] < omega[j] && i != j)
                {
                        flag_exp = false;
                        break;
                }
        }
        if (flag_exp)
        {
                K_extr++; continue;
        }


        flag_exp = true;
        for (int j = max(i - rad_exp, 0); j < min(i + rad_exp, N); ++j)
        {
                if (omega[i] > omega[j] && i != j)
                {
                        flag_exp = false;
                        break;
                }
        }
        if (flag_exp)
                K_extr++;
    }

}


void InitPerceptron()
{

    // Init
    Perc.max_x.resize(4);

    Perc.max_x[0] = 1;
    Perc.max_x[1] = 1;
    Perc.max_x[2] = 30;
    Perc.max_x[3] = 0.05;

    Perc.max_y = 1;

    Perc.w.resize(3);

    Perc.w[0].resize(4);
    Perc.w[1].resize(4);
    Perc.w[2].resize(2);

    Perc.w[0][0] = make_pair(0.01, 0);
    Perc.w[0][1] = make_pair(0.01, 1);
    Perc.w[0][2] = make_pair(0.01, 2);
    Perc.w[0][3] = make_pair(0.01, 3);

    Perc.w[1][0] = make_pair(0.01, 0);
    Perc.w[1][1] = make_pair(0.01, 1);
    Perc.w[1][2] = make_pair(0.01, 2);
```

```
	Perc.w[1][3] = make_pair(0.01, 3);

	Perc.w[2][0] = make_pair(0.01, 0);
	Perc.w[2][1] = make_pair(0.01, 1);


	ifstream Log_study("Weight.txt", ios_base::in);


	for (int i = 0; i < Perc.w.size(); ++i)
	{
		for (int j = 0; j < Perc.w[i].size(); ++j)
		{
			Log_study >> Perc.w[i][j].first >> Perc.w[i][j].second;
		}
	}


}


void set_map(IplImage* image)
{
	IplImage* image_text1 = cvCreateImage(cvSize(pixel_width, pixel_height),
IPL_DEPTH_8U, 3);
	IplImage* image_text2 = cvCreateImage(cvSize(pixel_width, pixel_height),
IPL_DEPTH_8U, 3);

	CvFont font;
	cvInitFont(&font, CV_FONT_HERSHEY_COMPLEX, 0.4 * pixel_height / 720, 0.4 *
pixel_height / 720, 0, 1, CV_AA);
	char text_string[60];
	CvPoint text_point;

	cvSet(image, CV_RGB(255, 255, 255));


	// Область карты
	cvRectangle(image, cvPoint(left_ind * pixel_width, up_ind * pixel_height),
cvPoint((1 - right_ind) * pixel_width, (1 - down_ind) * pixel_height), CV_RGB(0, 0, 0),
2);


	for (double sigma3_cur = sigma3_start; sigma3_cur <= sigma3_end; sigma3_cur +=
sigma3_step) {

		// Отметки на оси sigma3
		cvLine(image, cvPoint(left_ind * pixel_width + (1 - right_ind - left_ind) *
pixel_height * sigma3_cur / (sigma3_end + sigma3_add_g - sigma3_start), (1 - down_ind) *
pixel_height),
			cvPoint(left_ind * pixel_width + (1 - right_ind - left_ind) *
pixel_height * sigma3_cur / (sigma3_end + sigma3_add_g - sigma3_start), (1 - down_ind -
dash_width) * pixel_height), CV_RGB(0, 0, 0), 1);

		// Надписи на оси sigma3
		sprintf_s(text_string, 60, "%.1f", sigma3_cur);
		text_point = cvPoint((left_ind - 0.015) * pixel_width + (1 - right_ind -
left_ind) * pixel_height * sigma3_cur / (sigma3_end + sigma3_add_g - sigma3_start), (1 -
down_ind + 0.02) * pixel_height); cvPutText(image, text_string, text_point, &font,
CV_RGB(0, 0, 0));
	}


	for (double tau_cur = tau_start; tau_cur <= tau_end; tau_cur += tau_step) {
```

```
            // Отметки на оси tau
            cvLine(image, cvPoint(left_ind * pixel_width, up_ind * pixel_height + (1 -
down_ind - up_ind) * pixel_height * (tau_end + tau_add_g - tau_cur) / (tau_end +
tau_add_g - tau_start)),
                  cvPoint((left_ind + dash_width) * pixel_width, up_ind * pixel_height
+ (1 - down_ind - up_ind) * pixel_height * (tau_end + tau_add_g - tau_cur) / (tau_end +
tau_add_g - tau_start)), CV_RGB(0, 0, 0), 1);

            // Надписи на оси tau
            sprintf_s(text_string, 60, "%.1f", tau_cur);
            text_point = cvPoint((left_ind - 0.04) * pixel_width, (up_ind + 0.005) *
pixel_height + (1 - down_ind - up_ind) * pixel_height * (tau_end + tau_add_g - tau_cur) /
(tau_end + tau_add_g - tau_start)); cvPutText(image, text_string, text_point, &font,
CV_RGB(0, 0, 0));
      }

      // Название оси sigma3
      sprintf_s(text_string, 30, "sigma3");
      text_point = cvPoint(((1 - left_ind - right_ind) / 2 + left_ind - 0.03)*
pixel_width, (1 - down_ind + 0.06) * pixel_height); cvPutText(image, text_string,
text_point, &font, CV_RGB(0, 0, 0));


      // Название оси tau
      sprintf_s(text_string, 30, "tau");
      cvSetZero(image_text1);
      text_point = cvPoint(((1 - up_ind - down_ind) / 2 + up_ind - 0.06) * pixel_height,
(left_ind - 0.06) * pixel_width); cvPutText(image_text1, text_string, text_point, &font,
CV_RGB(255, 255, 255));
      cvTranspose(image_text1, image_text2);
      cvFlip(image_text2);
      cvAbsDiff(image, image_text2, image);


      int cur_coord_x;
      int cur_coord_y;
      int rad_legend = 8;

      // Легенда

      //Type 1
      cur_coord_x = (left_ind + 0.1 - 0.03)* pixel_width;
      cur_coord_y = (up_ind - 0.05 - 0.005) * pixel_height;

      for (int i = max(cur_coord_x - rad_legend, 0); i <= min(cur_coord_x + rad_legend,
pixel_width - 1); ++i)
            for (int j = max(cur_coord_y - rad_legend, 0); j <= min(cur_coord_y +
rad_legend, pixel_height - 1); ++j)
            {
                  setPixel(image, i, j, 0, 255);
                  setPixel(image, i, j, 1, 0);
                  setPixel(image, i, j, 2, 0);
            }

      sprintf_s(text_string, 30, "type 1");
      text_point = cvPoint((left_ind + 0.1)* pixel_width, (up_ind - 0.05) *
pixel_height); cvPutText(image, text_string, text_point, &font, CV_RGB(0, 0, 0));



      //Type 2
      cur_coord_x = (left_ind + 0.35 - 0.03)* pixel_width;
      cur_coord_y = (up_ind - 0.05 - 0.005) * pixel_height;
```

```
	for (int i = max(cur_coord_x - rad_legend, 0); i <= min(cur_coord_x + rad_legend,
pixel_width - 1); ++i)
		for (int j = max(cur_coord_y - rad_legend, 0); j <= min(cur_coord_y +
rad_legend, pixel_height - 1); ++j)
		{
			setPixel(image, i, j, 0, 0);
			setPixel(image, i, j, 1, 255);
			setPixel(image, i, j, 2, 0);
		}

	sprintf_s(text_string, 30, "type 2");
	text_point = cvPoint((left_ind + 0.35)* pixel_width, (up_ind - 0.05) *
pixel_height); cvPutText(image, text_string, text_point, &font, CV_RGB(0, 0, 0));



	//Type 3
	cur_coord_x = (left_ind + 0.6 - 0.03)* pixel_width;
	cur_coord_y = (up_ind - 0.05 - 0.005) * pixel_height;

	for (int i = max(cur_coord_x - rad_legend, 0); i <= min(cur_coord_x + rad_legend,
pixel_width - 1); ++i)
		for (int j = max(cur_coord_y - rad_legend, 0); j <= min(cur_coord_y +
rad_legend, pixel_height - 1); ++j)
		{
			setPixel(image, i, j, 0, 0);
			setPixel(image, i, j, 1, 0);
			setPixel(image, i, j, 2, 255);
		}

	sprintf_s(text_string, 30, "type 3");
	text_point = cvPoint((left_ind + 0.6)* pixel_width, (up_ind - 0.05) *
pixel_height); cvPutText(image, text_string, text_point, &font, CV_RGB(0, 0, 0));

}

int main()
{
	freopen("Set_points.txt", "r", stdin);

	InitPerceptron();

	IplImage* image1 = cvCreateImage(cvSize(pixel_width, pixel_height), IPL_DEPTH_8U,
3);
	IplImage* image2= cvCreateImage(cvSize(pixel_width, pixel_height), IPL_DEPTH_8U,
3);

	set_map(image1);
	set_map(image2);

	double read_sigma3, read_tau;
	int read_type_chimera;

	int rad_point = 8;

	char text_string[60];



	while (cin >> read_sigma3 >> read_tau)
	{
		vector<double> omega(2 * N, 0);
```

```
		char text_string[60];
		sprintf_s(text_string, 60, "Data\\%.6f %.6f.txt", read_sigma3, read_tau);

		ifstream Log(text_string, ios_base::in);


		for (int i = 0; i < 2*N; ++i)
		{
			Log >> omega[i];
		}

		Log.close();


		double K1, K2, K_extr, Delta_omega;


		analysis_chimera_type(omega, K1, K2, K_extr, Delta_omega);

		double P1= 0.914, P2= 0.086, P_extr=19, Eps_omega= 0.028;

		if (K1 <= P1 && K2 >= P2)
		{
			read_type_chimera = 3;
		}
		else if (Delta_omega <= Eps_omega || K_extr>P_extr)
		{
			if (read_sigma3 < 0.1 && read_tau <= 0.5)
				int lol = 1;
			read_type_chimera = 2;
		}
		else
		{
			read_type_chimera = 1;
		}

		vector<double> type;
		vector<double> input_x = { K1, K2, K_extr, Delta_omega };

		type = Perc.Answer(input_x);


		int cur_coord_x = left_ind * pixel_width + (1 - right_ind - left_ind) *
pixel_height * read_sigma3 / (sigma3_end + sigma3_add_g - sigma3_start);
		int cur_coord_y = up_ind * pixel_height + (1 - down_ind - up_ind) *
pixel_height * (tau_end + tau_add_g - read_tau) / (tau_end + tau_add_g - tau_start);


		for (int it = 0; it < 3; ++it)
		{
			double lol = type[it];
			if (type[it] > 1)
				type[it] = 1;
			else if(type[it] < 0)
				type[it] = 0;
		}


		for (int i = max(cur_coord_x - rad_point, 0); i <= min(cur_coord_x +
rad_point, pixel_width - 1); ++i)
			for (int j = max(cur_coord_y - rad_point, 0); j <= min(cur_coord_y +
rad_point, pixel_height - 1); ++j)
			{
				setPixel(image1, i, j, 0, 0);
```

```
                        setPixel(image1, i, j, 1, 0);
                        setPixel(image1, i, j, 2, 0);

                        setPixel(image1, i, j, read_type_chimera - 1, 255);

                        setPixel(image2, i, j, 0, type[0] * 255);
                        setPixel(image2, i, j, 1, type[1] * 255);
                        setPixel(image2, i, j, 2, type[2] * 255);
                }

        }


        // Путь для вывода карты
        sprintf_s(text_string, 60, "Map1.jpg");


        // Сохранение карты по выбранному пути
        cvSaveImage(text_string, image1);


        // Путь для вывода карты
        sprintf_s(text_string, 60, "Map2.jpg");


        // Сохранение карты по выбранному пути
        cvSaveImage(text_string, image2);


        return 0;

}
```