МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКАРИНЫ
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра ПМиИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по теме:

Разработка системы нейросетевого управления

Руководитель
доц., канд. техн. наук _____ О.И. Федяев
(подпись) (дата)

Разработал
студент гр. ПО-99 а _____ А.Е. Тихонов
(подпись) (дата)

2003

РЕФЕРАТ

Пояснительная записка работы 34 с., 17 рис., 2 табл., 6 источников.

Объектом исследования – нейросетевое управление.

Цель работы – создание схемы нейросетевого управления на примере автомобиля.

Метод исследования – ознакомление с существующими структурами нейросетевого управления, написание алгоритмов реализующих нейросеть с алгоритмом обучения, экспериментальное исследование свойств объекта с получением обучающего множества для нейросети.

Результаты работы – программный продукт, реализующий нейросетевое управление для тестового объекта; приложение реализующее модель объекта.

НЕЙРОСЕТЬ, УПРАВЛЕНИЕ, СИНТЕЗ, АНАЛИЗ, ОПТИМАЛЬНОСТЬ, АЛГОРИТМ, ПРОГРАММА

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время существует много методов синтеза управляющих элементов для управления динамическими моделями, основой построения для которых служит теория автоматического управления. Однако пока не существует общего подхода для управления динамическими трудно формализуемыми моделями. Одним из таких подходов может выступать управление основанное на искусственных нейронных сетях. В работе рассмотрены основные способы нейросетевого управления, дан сравнительный анализ с существующими схемами управления. Произведено построение нейросетевого регулятора для управления тестовым объектом – автомобиль. Построение нейросетевых регуляторов достаточно новая проблема для современной науки, большинство разработок не имеют соответствующего теоретического обоснования, хотя и показывают достаточно перспективные результаты. В данной области еще необходимо провести много исследований, выработать соответствующие общие алгоритмы для построения нейрорегуляторов, для того, чтобы широко применять такие регуляторы на практике.

1 АВТОМАТИЧЕСКОЕ УПРАВЛЕНИЕ ОБЪЕКТАМИ

1.1 Основные определения

Управление каким-либо объектом – это процесс воздействия на него с целью обеспечения требуемого течения процессов в объекте или требуемого изменения состояния. Основой управления является получения и обработка информации о состоянии объекта и внешних условиях его работы для определения воздействий, которые необходимо приложить к объекту, чтобы обеспечить достижение цели управления.

Управление, осуществляемое без участия человека, называется автоматическим управлением. Устройство, с помощью которого осуществляется управление, называется управляющим устройством. Совокупность объекта управления и управляющего устройства образует систему автоматического управления (САУ).

Общий вид системы автоматического управления представлен на рис. 1.1., где О – объект управления, УУ – управляющее устройство.

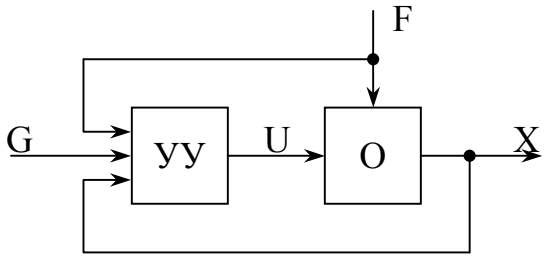

Рисунок 1.1 Блок-схема системы автоматического управления

Состояние объекта характеризуется выходной величиной X. От управляющего устройства на вход объекта поступает управляющее воздействие U. Помимо управляющего воздействия, к объекту приложено возмущающее воздействие (возмущение, помеха) F, которое изменяет состояние объекта, т.е. X, препятствуя управлению. На вход управляющего

устройства подается задающее воздействие (задание) G, содержащее информацию о требуемом значении X, т.е. о цели управления. В самом общем случае на вход объекта поступает также информация о текущем состоянии объекта в виде выходной величины X и о действующем на объект возмущении F. Переменные U, G, F и X в общем случае являются векторами.

1.2 Классификация систем автоматического управления

В зависимости от видов используемой в управляющем устройством информации различают два основных типа САУ – разомкнутые и замкнутые системы. В разомкнутых системах на вход управляющего устройства не подается величина X, т.е. нет обратной связи. Такие системы различают на системы управления по задающему воздействию (на вход УУ подается G) и по возмущению (на вход УУ подается F). В замкнутых САУ на вход УУ подается X.

Широко распространенным видом САУ являются системы автоматического регулирования (САР), задача которой заключается в поддержании выходной величины объекта X на заданном уровне G, т.е. в поддержании равенства X=G.

В зависимости от характера задающего воздействия САР делятся на три вида: системы стабилизации, системы программного регулирования (управления) и следящие системы. В системах стабилизации задающее воздействие постоянно, в системах программного регулирования оно изменяется по заранее заданному закону, в следящих системах оно тоже изменяется, но закон изменения заранее не известен. Управляющее устройство в САР – регулятор, а выходная величина – регулируемая величина.

В зависимости от количества выходных координат объекта управления, образующих вектор выходной величины X, САУ делятся на одномерные и многомерные(двумерные и т.д.).

Многомерные САУ делятся на системы связного и несвязного управления. Система несвязного управления имеет несколько управляющих устройств, каждое из которых осуществляет управления своей выходной координатой объекта. При этом устройства не имеют взаимных связей. В системе связного управления отдельные управляющие устройства связаны друг с другом.

САУ делятся на линейные и нелинейные системы. Линейные системы описываются линейными дифференциальными уравнениями. Для таких систем справедлив принцип суперпозиции – реакция на любую комбинацию внешних воздействий равна сумме реакций на каждое из этих воздействий, поданных на систему порознь. Благодаря принципу суперпозиции разработана теория систем автоматического уравнения. К нелинейным системам принцип суперпозиции не применим. Нет и общей теории нелинейных дифференциальных уравнений.

САУ делятся на стационарные – параметры не изменяются во времени и нестационарными – параметры являются функцией от времени. В соответствие с данным определением для нестационарной системы реакция на одно и тоже воздействие зависит от момента приложения этого воздействия.

В зависимости от характера действия составляющих САУ делятся на системы непрерывного действия и дискретного действия. Различие состоит в том, что при подаче входного сигнала, который плавно изменяется, у дискретных систем выходная величина изменяется дискретно (скачками).

Выделяют адаптивные системы (самоприспосабливающиеся). Они обладают способностью приспосабливаться к изменению условий работы и улучшать свою работу по мере накопления опыта. Неадаптивные (обыкновенные) САУ такой способностью не обладают. Они имеют

постоянную настройку. Если вследствие какого-либо изменения условий работы обыкновенной системы её настройку необходимо изменить, то это должен выполнить человек. Область применения адаптивных САУ – управление объектами, свойства или условия которых недостаточно изучены.

1.3 Характеристика процессов в системах автоматического управления

Как и у всякой динамической системы, процессы в САУ делят на установившиеся и переходные.

При рассмотрении САУ имеет значение следующие понятия: устойчивость системы, качество процесса управления и точность управления.

Устойчивость – это свойство системы возвращаться в установившееся состояние после того, как она была выведена из этого состояния каким-либо возмущением.

Качество процесса управления характеризуется тем, насколько процесс управления близок к желаемому. Количественно они выражаются критериями качества:

1. Время переходного процесса – интервал времени от начала переходного процесса до момента, когда отклонение выходной величины от её нового установившегося значения становится меньше определенной величины – обычно 5%.
2. Максимальное отклонение в переходной период (перерегулирование) – отклонение определяется от нового установившегося значения и выражается в процентах.
3. Колебательность переходного процесса – определяется числом колебаний, равных числу минимумов кривой переходного процесса за время переходного процесса. Часто колебательность выражают в процентах как отношение соседних максимумов кривой переходного процесса.

4. Точность управления характеризуется погрешностью системы в установившихся режимах (расхождение между желаемым сигналом и действительным) – статизм.

2 АНАЛИЗ СВОЙСТВ ОБЪЕКТА

Для определения вида передаточной функции объекта решаем систему уравнений(А.1) методом Рунге-Кутта. Приведем систему к виду:

$$\frac{d^2v}{dt^2} = f(t, v, \frac{dv}{dt}) \tag{2.1}$$

Получим:

$$f = \frac{K_1 \cdot K_2 \cdot x - v - (T_1 + T_2)\frac{dv}{dt}}{T_1 \cdot T_2} \tag{2.2}$$

Имея следующие начальные условия:

$$\begin{aligned} &v(t_0) = v_0 \\ &\frac{dv}{dt}\Big| t_0 = v'_0 \end{aligned} \tag{2.3}$$

Вводим следующие обозначения:

$$\begin{aligned} &t_0 + k\Delta t = t_k \\ &v(t_k) = v_k \\ &\frac{dv}{dt}\Big| t_k = v'_k \\ &f(t_k, v_k, v'_k) = f_k \\ &k = 0,1,2\ldots \end{aligned} \tag{2.4}$$

Применяем следующие формулы:

$$\begin{aligned} &v_{k+1} = v_k + v'_k \cdot \Delta t + \frac{r_1 + r_2 + r_3}{6}\Delta t \\ &v'_{k+1} = v'_k + \frac{r_1 + 2 \cdot r_2 + 2 \cdot r_3 + r_4}{6} \\ &r_1 = f(t_k, v_k, v'_k) \cdot \Delta t \\ &r_2 = f(t_k + \frac{\Delta t}{2}, v_k + v'_k \frac{\Delta t}{2}, v'_h + \frac{r1}{2}) \cdot \Delta t \\ &r_3 = f(t_k + \frac{\Delta t}{2}, v_k + v'_k \frac{\Delta t}{2} + \frac{r_1}{4}\Delta t, v'_k + \frac{r_2}{2}) \cdot \Delta t \\ &r_4 = f(t_k + \Delta t, v_k + v'_k \cdot \Delta t + \frac{r_2}{2}\Delta t, v'_k + r_3) \cdot \Delta t \end{aligned} \tag{2.5}$$

В итоге получаем динамическое отображение

$$\big(x(t), v(t), v'(t), v''(t)\big) \to \big(v(t + \Delta t), v'(t + \Delta t), v''(t + \Delta t)\big) \tag{2.6}$$

Это отображение реализовано программно. В дальнейшем для анализа свойств объекта используется эта программная модель. Необходимо подчеркнуть, что в общем случае отображение такого вида может бить получено не из аналитически заданного объекта, а путем считывания соответствующих датчиков объекта не поддающегося формализации.

Вид переходных процессов изображен на рис.2.1, рис.2.2 при нулевых начальных условиях.

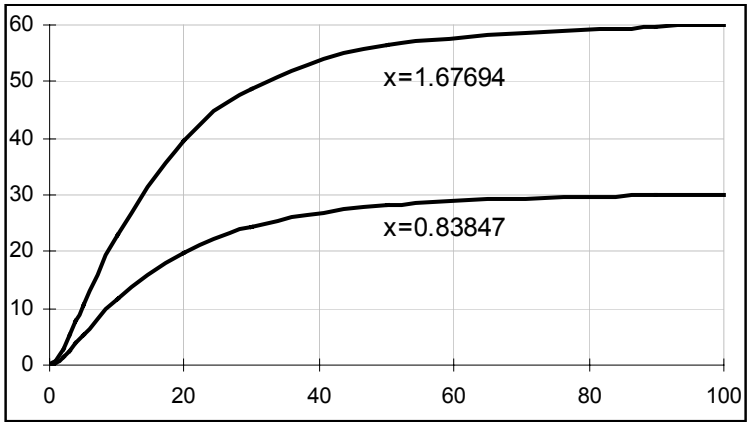

Рисунок 2.1 Переходной процесс для v

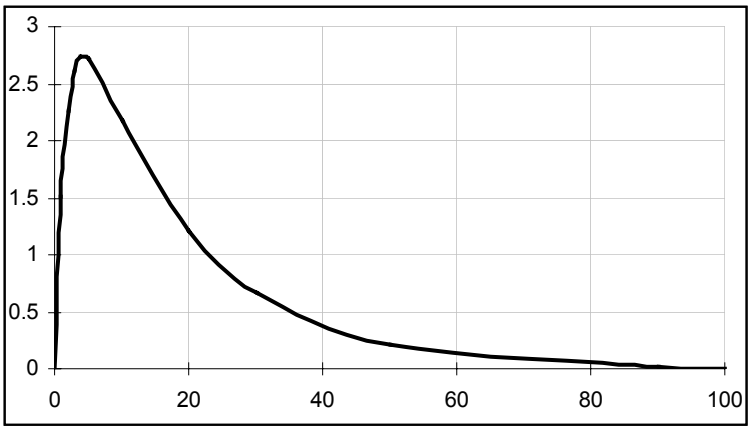

Рисунок 2.2 Переходной процесс для $\frac{dv}{dt}$

Анализ графика для v, показывает что система устойчива, имеет длительность переходного процесса порядка 60 секунд, переходной процесс не колебательный.

3 ОСНОВНЫЕ СХЕМЫ НЕЙРОСЕТЕВОГО УПРАВЛЕНИЯ

В общем виде объект управления описывается своей функциональной моделью $P\left(\mathbf{u}(t), \mathbf{y}(t)\right)$ связывающей вектор входных воздействий $\mathbf{u}(t)$ с вектором выходных сигналов $\mathbf{y}(t)$. Такое описание берет свое начало от идеи «черного ящика» и не является портретом динамического поведения объекта, а отражает только его функциональные связи.

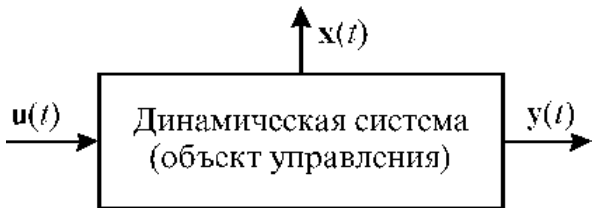

Рисунок 3.1 Динамическая система

Введем разбиение по времени $t_0,t_1,t_2,....$, где $t_{i+1}=t_i+\mathrm{D}\ t$, и обозначим $\mathbf{x}(t_k)$, $\mathbf{y}(t_k)$ и $\mathbf{u}(t_k)$ как $\mathbf{x}(k)$, $\mathbf{y}(k)$ и $\mathbf{u}(k)$, соответственно. Тогда динамику системы можно описать следующими разностными уравнениями:

$$\begin{aligned} \mathbf{x}(k+1) &= \Phi\left(\mathbf{x}(k), \mathbf{u}(k)\right); \\ \mathbf{y}(k) &= F\left(\mathbf{x}(k)\right). \end{aligned} \tag{3.1}$$

Предмет теории управления составляют анализ и синтез динамических систем, в которых изменение одной или нескольких переменных ограничивается в определенных пределах. Задачей управления является синтез контроллера, который формирует желаемый вход $\mathbf{u}(k)$, основываясь на информации, доступной в момент времени t_k.

Использование ИНС позволяет решать задачу управления нелинейным объектом путем создания адаптивной СУ с обучаемым нейроконтроллером. Здесь под обучением подразумевается процесс выработки в СУ желаемой реакции на внешние сигналы путем многократных воздействий на систему и внешней корректировки. Внешняя корректировка осуществляется «учителем», которому известна желаемая реакция СУ на определенные

воздействия. Таким образом, при обучении “учитель” сообщает системе дополнительную информацию о том, верна или неверна ее реакция (рис.3.2).

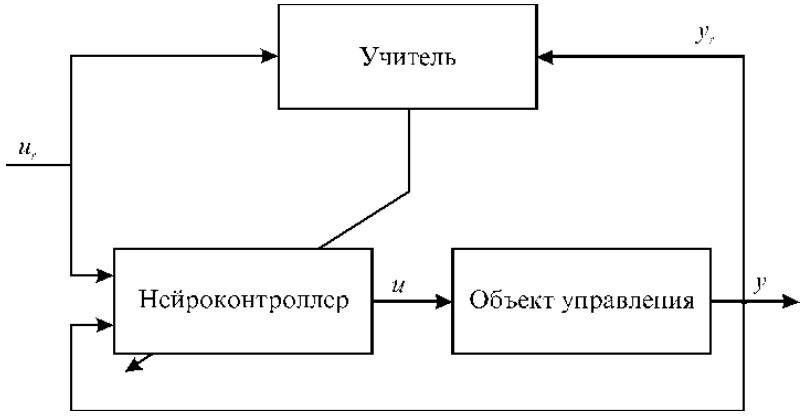

Рисунок 3.2 Общая схема СУ с обучаемым нейроконтроллером

Существует множество подходов к применению ИНС в качестве НК. Можно использовать ИНС, как нелинейные усилители при интегральной, дифференциальной и пропорциональной частях ПИД-контроллера (рис. 3.3).

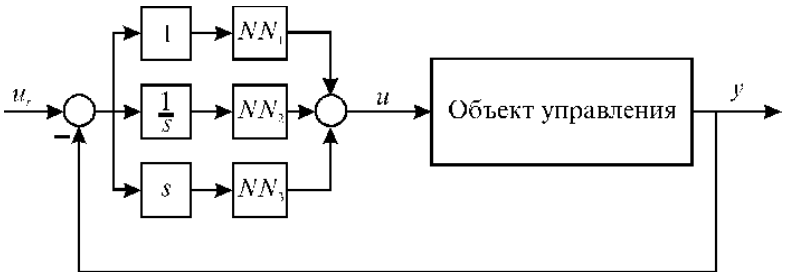

Рисунок 3.3 СУ с комбинированным ПИД-нейроконтроллером

Можно испольовать ИНС получающую на вход сигнал управления системой $\mathbf{u}_r(k)$ и задержанные несколько раз сигналы с выхода объекта управления. В общем случае на ИНС можно подавать и задержанные сигналы со своего выхода (рис.3.4). Таким образом, он формирует управляющее воздействие по следующему закону:

$$\mathbf{u}(k) = NN\left(\mathbf{y}(k), \mathbf{y}(k-1), \ldots, \mathbf{y}(k-l_1), \mathbf{u}(k-1), \mathbf{u}(k-2), \ldots, \mathbf{u}(k-l_2), \mathbf{u}_r(k)\right) \quad (3.2)$$

где l_1 и l_2 — глубины задержек обратных связей по выходу и входу объекта управления, соответственно.

Рисунок 3.4 Система управления с нейроконтроллером

ИНС можно обучать по схеме копирования эталонной модели(рис.3.5).

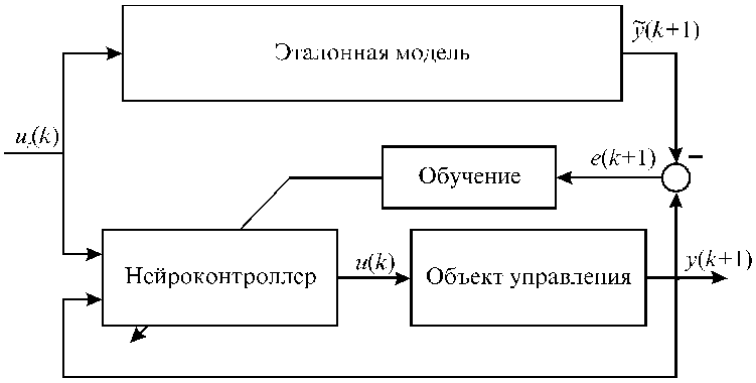

Рисунок 3.5 Нейросетевая система управления с эталонной моделью

ИНС можно обучать и без эталонной модели (рис.3.6).

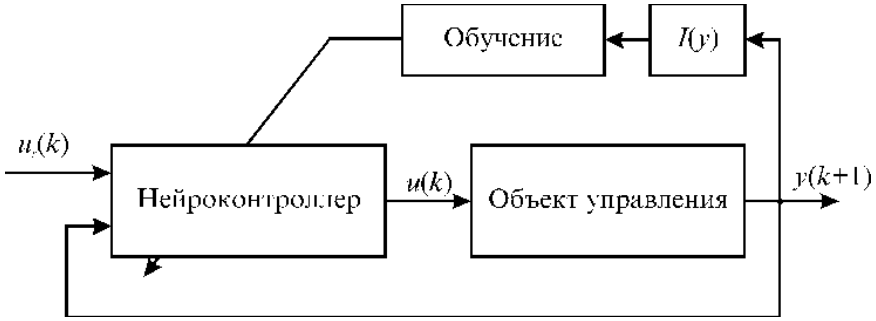

Рисунок 3.6 Система управления с экстремальным законом управления

ИНС можно обучать обратной динамике объекта (рис.3.7).

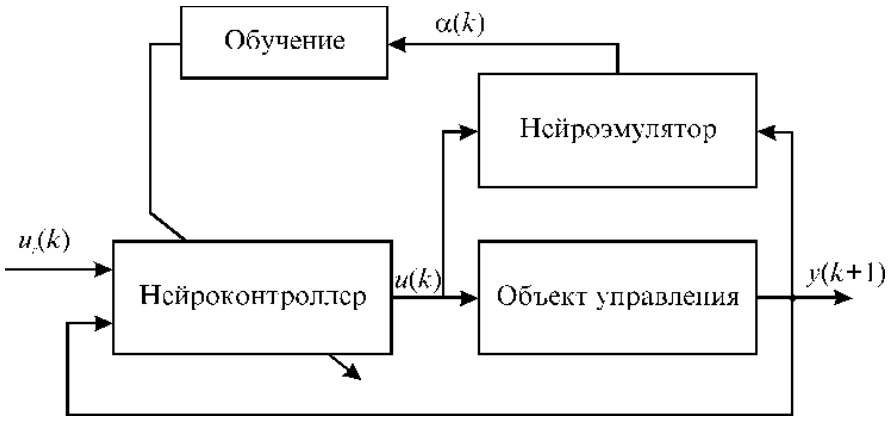

Рисунок 3.7 Система управления с нейроэмулятором

4 ПРИМЕРЫ РЕАЛИЗАЦИИ НЕЙРОРЕГУЛЯТОРОВ

4.1 Схема с эталонной моделью без учета помех

Применена схема изображенная на рис.3.5. Эталонная модель синтезировалась вручную путем подбора соответствующего входного воздействия для получения желаемого выхода – поддержание скорости автомобиля на уровне 60 км/ч, с расчетным временем импульса 3с. Входами ИНС служили текущее состояние объекта, однозначно определяемого по выходному сигналу и его первой производной. Обучающее множество приведено в табл.4.1.

Таблица 4.1 Обучающее множество для схемы без учета помех

V	V`	U
0	0	5
15.635	7.805	5
39.859	7.917	4.325
60	5.819	-0.697
60	-3.113	2.947
30	0	5
43.013	6.496	3.985
60	5.005	-0.364
60	-2.678	2.769
50	0	4.595
60	4.992	-0.359
60	-2.671	2.766
60	0	1.6769

Результаты моделирования представлены на рис.3.8, рис.3.9.

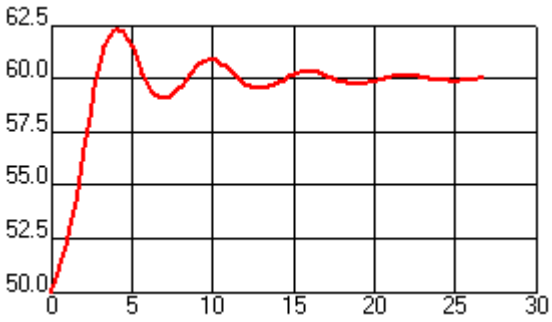

Рисунок 3.8 Разгон с 50 до 60 км/ч, длина импульса 3с

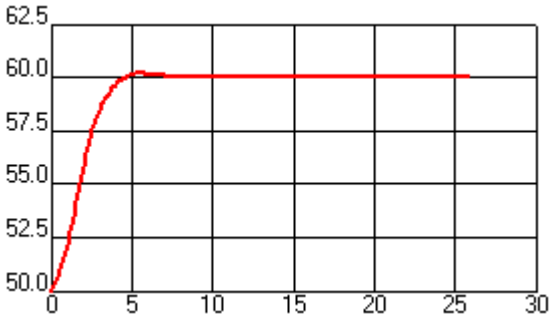

Рисунок 3.9 Рисунок 3.8 Разгон с 50 до 60 км/ч, длина импульса 2с

Отметим что уменьшение импульса (повышение частоты работы нейрорегулятора) приводит к более гладкой кривой.

4.2 Схема с эталонной моделью с учетом помех

Проведено моделирование аналогичное схеме без учета помех, с добавление входа нейрорегулятора – предыдущее значение управляющего импульса. Обучающее множество приведено в табл.4.2. Результаты моделирования (сглаженные – импульс 0.1с) приведены на рис.3.10, рис.3.11.

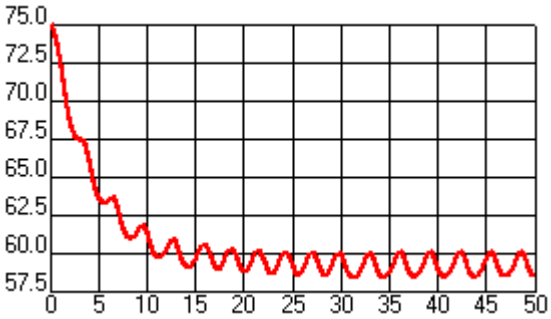

Рисунок 3.10 Помеха +70% действует с 15с по 30с

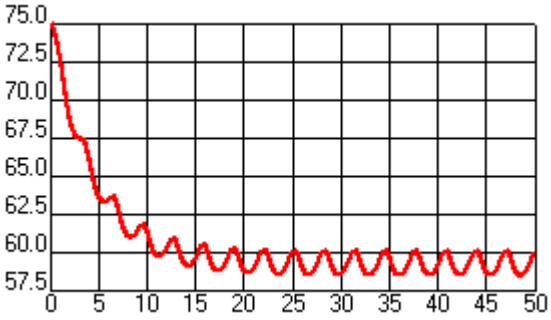

Рисунок 3.11 Помеха отсутствует

Таблица 4.2 Обучающее множество для схемы с учетом помех

V	V′	U(-1)	U
15.634892	7.805375	1.000000	1.000000
15.634892	7.805375	0.500000	0.500000
39.859189	7.917386	1.000000	0.932546
39.859189	7.917386	0.467454	0.400000
60.000001	5.818868	0.932546	0.430338
60.000001	5.818868	0.702208	0.200000
110.000000	0.000000	0.430338	0.000000
110.000000	0.000000	0.830337	0.400000
60.000002	-5.612952	0.459465	0.896646
60.000002	-5.612952	-0.137181	0.300000
60.000001	3.003061	0.896646	0.545195
60.000001	3.003061	0.751451	0.400000
20.000000	0.000000	0.545195	1.000000
20.000000	0.000000	0.345196	0.800000
55.403158	7.032267	1.000000	0.515001
55.403158	7.032267	0.584999	0.100000
60.000002	-1.467556	0.515001	0.727554
60.000002	-1.467556	0.687447	0.900000
90.000000	0.000000	0.727554	0.000000
90.000000	0.000000	0.927553	0.200000
60.000001	3.032150	0.956557	0.544009
60.000001	3.032150	0.712548	0.300000
60.000000	-1.622273	0.544009	0.733865
60.000000	-1.622273	0.310144	0.500000
40.000000	0.000000	0.733865	1.000000
40.000000	0.000000	0.133865	0.400000
59.999999	0.682036	0.649914	0.639871
59.999999	0.682036	0.910043	0.900000
59.999999	-0.364905	0.639871	0.682576
59.999999	-0.364905	0.157295	0.200000
70.000000	0.000000	0.682576	0.375843
70.000000	0.000000	0.806734	0.500000
60.000001	2.670988	0.871329	0.558741
60.000001	2.670988	0.712588	0.400000
60.000001	-1.429042	0.558741	0.725983
60.000001	-1.429042	0.732758	0.900000
60.000000	0.000000	0.725983	0.667691
60.000000	0.000000	0.558291	0.500000
60.000001	0.000001	0.667692	0.667691
60.000001	0.000001	0.750000	0.750000
60.000001	0.000000	0.667691	0.667691
60.000001	0.000000	0.900000	0.900000

Явно выраженная колебательность переходного процесса является следствием плохой обученности сети. Однако общий эффект виден – нейрегулятор практически не реагирует на помеху.

4.3 Схема обучения обратной динамике объекта

Схема обучения приведена на рис.3.7. Входами нейрорегулятора являются – опорный сигнал, выход объекта, первая производная выхода объекта. При обучении на опорный сигнал подавался текущий выход, на второй и третий вход нейрорегулятора подавались задержанные скорость с первой производной. Разница между сигналом который подавался на объект и выходом нейрорегулятора рассматривалась как ошибка обучения. Так как обучающее множество не формировалась, а объект и нейрорегулятор работали одновременно, то ошибка обучения представляет собой и ошибку обобщения. Выходы объекта снимались после 3с задержки после подачи управляющего воздействия. Результаты моделирования (сглаженные) приведены на рис.3.12, 3.13, 3.14.

Рисунок 3.12 Разгон с 30 км/ч до 40 км/ч

Рисунок 3.13 Торможение с 80 км/ч до 75 км/ч

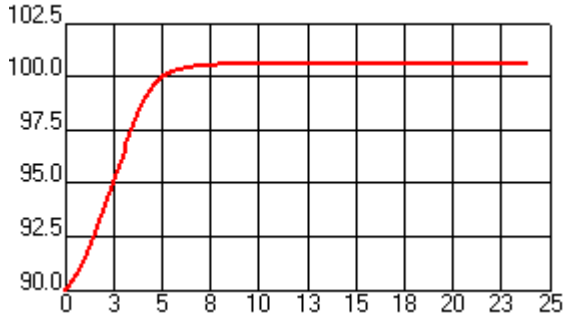

Рисунок 3.13 Разгон с 90 км/ч до 100 км/ч

4.4 Анализ результатов

Для всех нейрорегуляторов использовалась ИНС с одним скрытым слоем (число нейронов 50). Приведенные нейрорегуляторы демонстрируют возможность применения ИНС в задачах управления. Наилучшие результаты показала система с эталонной моделью без учета помех, наихудшие результаты у системы с эталонной моделью с учетом помех. Это связано не с конкретной схемой управления, а с тем, что с увеличением сложности моделируемой функции или числом входов ИНС время обучения возрастает очень сильно. Это является главным недостатком алгоритма обратного распространения ошибки, применяемого для обучения. Если для первой системы понадобилось порядка ста тысяч предъявлений обучающего множества, то для второй и третьей счет идет уже на миллионы.

Используемые схемы управления также налагают ограничения на применимость нейрорегуляторов. Схема с эталонной моделью хороша только если эта эталонная модель есть (например гонщик за рулем автомобиля), причем эталонная модель копируется полностью, ничего нового ИНС не открывает. В схеме с обучением обратной динамике объекта напротив, ИНС узнает объект по внешним воздействиям и сама формирует управляющую кривую. Но требует дополнительного входа у ИНС по сравнению со схемой с эталонной моделью для задач стабилизации для управляющего воздействия.

ВЫВОДЫ

Нейросетевое управление свободно от ограничений на линейность системы, эффективно в условиях шумов и после окончания обучения обеспечивает управление в реальном масштабе времени. Нейросетевые СУ более гибко настраиваются на реальные условия, образуя модели полностью адекватные поставленной задаче, не содержащие ограничений, связанных с построением формальных систем. Кроме того, нейросетевые СУ не только реализуют стандартные адаптивные методы управления, но и предлагают свои алгоритмические подходы к ряду задач, решение которых вызывает затруднение вследствие неформализованности. Так, появляется возможность обработки в рамках одной модели данных одной природы – для НС важна лишь их коррелированность.

Таким образом, будущее интеллектуального управления лежит в сочетании традиционного управления с потенциальными возможностями и перспективами использования систем, основанных на использовании искусственных НС.

ПЕРЕЧЕНЬ ССЫЛОК

1. Юревич Е.А. Теория автоматического управления. – Ленинград: Энергия, 1975. – 416 с.
2. Заенцев И.В. Нейронные сети: основные модели – Воронеж 1999. – 76 с.
3. Миркес Е.М. Учебное пособие по курсу нейроинформатика – Красноярск 2002. – 347 с.
4. Горбань А.Н., Дудин-Барковский В.Л. Нейроинформатика. – Новосибирск: Наука. Сибирское предприятие РАН, 1998. – 296с.
5. Вороновский Г.К., Махотило К.В. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности – Харьков: Основа, 1997. – 107 с.
6. Уоссермен Ф. Нейрокомпьютерная техника: теория и практика – 1992.

ПРИЛОЖЕНИЕ А ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу

ст. гр. ПО-99-а Тихонову Андрею Евгеньевичу

Тема:

Исходные данные: объект управления представляет автомобиль. Параметр, которым необходимо управлять – это его скорость. Скорость автомобиля можно регулировать с помощью другого параметра. Это может быть: сила нажатия на педаль акселератора и т.п. На скорость автомобиля может влиять множество внешних факторов: уклон под которым движется автомобиль, качество сцепления с дорогой, ветер. Информация о скорости автомобиля поступает с датчика скорости.

Динамика объекта управления описывается следующей системой дифференциальных уравнений (А.1).

$$\begin{cases} T_1 \dfrac{dy}{dt} + y = K_1 x \\ T_2 \dfrac{dv}{dt} + v = K_2 y \end{cases} \qquad \text{(А.1)}$$

Параметры T1, T2, K1, K2 определены экспериментально и имеют следующие значения соответственно: K1=5, K2=7.156, T1=1.735, T2=16.85.

Задание: требуется построить такой регулятор в классе нейросетевых структур, который обеспечивал бы управление объектом при соблюдении следующих требований синтезируемой системе автоматического управления:

1. Физическая реализуемость регулятора.
2. Устойчивость работы.
3. Минимальная сложность.

Дата вадачи задания

Руководитель курсового проекта

ПРИЛОЖЕНИЕ Б ТЕКСТ ПРОГРАММЫ

Model.h

```
#pragma once

#define DEF_K1 5.0
#define DEF_K2 7.156
#define DEF_T1 1.735
#define DEF_T2 16.85

class CModel
{
public:
	CModel(void);
	void SetParam(double K1, double K2, double T1, double T2);
	void GetParam(double * K1, double * K2, double * T1, double * T2) const;
	void SetState(double V, double dV);
	void GetState(double * V, double * dV = NULL) const;
	double Next(double t, double X);
	~CModel(void);
private:
	double f(double X, double V, double dV) const;
	double m_K1;
	double m_K2;
	double m_T1;
	double m_T2;
	double m_V;
	double m_dV;
};
```

Model.cpp

```
#include "StdAfx.h"
#include "model.h"

CModel::CModel(void)
{
	m_K1 = DEF_K1;
	m_K2 = DEF_K2;
	m_T1 = DEF_T1;
	m_T2 = DEF_T2;
	m_V = 0;
	m_dV = 0;
}

CModel::~CModel(void)
{
}

void CModel::SetParam(double K1, double K2, double T1, double T2)
{
	m_K1 = K1;
	m_K2 = K2;
	m_T1 = T1;
	m_T2 = T2;
}

void CModel::GetParam(double * K1, double * K2, double * T1, double * T2) const
```

```
{
	*K1 = m_K1;
	*K2 = m_K2;
	*T1 = m_T1;
	*T2 = m_T2;
}

void CModel::SetState(double V, double dV)
{
	m_V = V;
	m_dV = dV;
}

void CModel::GetState(double * V, double * dV) const
{
	*V = m_V;
	if(dV != NULL)
		*dV = m_dV;
}

double CModel::Next(double t, double X)
{
	double dt = 0.001;
	for(int i = 0; i < t / dt; i++)
	{
		double r1 = f(X, m_V, m_dV) * dt;
		double r2 = f(X, m_V + m_dV * dt / 2, m_dV + r1 / 2) * dt;
		double r3 = f(X, m_V + m_dV * dt / 2 + r1 * dt / 4, m_dV + r2 / 2) * dt;
		double r4 = f(X, m_V + m_dV * dt + r2 * dt / 2, m_dV + r3) * dt;
		m_V += m_dV * dt + (r1 + r2 + r3) * dt / 6;
		m_dV += (r1 + 2 * r2 + 2 * r3 + r4) / 6;
	}
	return m_V;
}

double CModel::f(double X, double V, double dV) const
{
	return (m_K1 * m_K2 * X - V - (m_T1 + m_T2) * dV) / m_T1 / m_T2;
}
```

NeuroNet.h

```
#pragma once

#define NN_OK				0				//операция прошла успешно
#define NN_NO_MEMORY		1				//не хватает памяти для выделения
#define NN_EXIST			2				//объект уже создан
#define NN_NOT_EXIST		3				//объект не создан
#define NN_ERR_FILE_OPEN	4				//ошибка открытия файла
#define NN_ERR_FILE_READ	5				//ошибка чтения файла
#define NN_ERR_FILE_WRITE	6				//ошибка записи файла
#define NN_NO_LEARN			7				//не зашружено обучающее множество

//описание класса CNeuroNet - нейросеть
class CNeuroNet
{
public:
	//конструктор - все параметры в ноль
	CNeuroNet();
	//выделение памяти  с инициализацией структуры сети
	//ВНИМАНИЕ оператор new в случае ошики возвращает NULL (Visual Studio 7)
	int Create(int nLayer, const int * pLayer);
	//загрузка нейросети из файла
```

```
        int Load(LPCTSTR filename);
        //сохранение нейросети в файл
        int Save(LPCTSTR filename) const;
        //заполнени матрицы m_pWeight случайными значениями равномерно
        //в интервале (left[i], right[i]) слоя i
        int SetRandomWeight(const double * pLeft = NULL, const double * pRight = NULL);
        //вычисление выхода(pOut) по входу нейросети(pInput)
        int Forward(const double * pInput, double * pOut = NULL);
        //метод обратного распространения ошибки,
        //pOut - целевой выход, NU - константа скорости сходимости
        int BackPropagation(const double * pOut, double NU);
        //загрузить обучающее множество из файла
        int LoadLearn(LPCTSTR filename);
        //обучить нейросеть, по количеству эпох - epoch
        //eps - максимальное распождение по обучающему множуству
        int Learn(int epoch, double NU, double * eps);
        //деструктор - освобождение памяти
        ~CNeuroNet();
        //логистическая функция активации и ее производная
        static double Activation(double x);
        static double Derivative(double x);
private:
        //признак выделения памяти под нейросеть
        bool m_bAllocated;
        //количество слоев(с фиктивным входным слоем)
        //для сети с одним скрытым слоем m_nLayer=3
        int m_nLayer;
        //m_pLayer[i] - кол-во нейронов в слое i=0..m_nLayer-1
        int * m_pLayer;
        //m_pWeight[i][j][k] - вес связи от нейрона j(слой i) к нейрону k(слой i+1)
        //i=0..m_nLayer-2, j=0..m_pLayer[i]-1, k=0..m_pLayer[i+1]-1;
        double *** m_pWeight;
        //m_pSignal[i][j] - выход нейрона j(слой i), i=0..m_nLayer, j=0..m_pLayer[i]-1
        double ** m_pSignal;
        //m_pSigma[i][j] - сигма(в обр. распр. ошибки) нейрона j(слой i+1, i=0..m_nLayer-2)
        double ** m_pSigma;
        //длина обучающего множества
        int m_nLearn;
        //m_pLearn[i][j]-обучающие пары,i=0..m_nLearn,j=0..m_pLayer[0]+m_pLayer[m_nLayer-1]-1
        double ** m_pLearn;
};
```

NeuroNet.cpp

```
#include "StdAfx.h"
#include "NeuroNet.h"
#include <math.h>

//--------------------------------------------------------------------------------
CNeuroNet::CNeuroNet()
{
        //все в ноль
        m_bAllocated = false;
        m_nLayer = 0;
        m_pLayer = NULL;
        m_pWeight = NULL;
        m_pSignal = NULL;
        m_pSigma = NULL;
        m_nLearn = 0;
        m_pLearn = NULL;
}
//--------------------------------------------------------------------------------
int CNeuroNet::Create(int nLayer, const int * pLayer)
```

```
{
	//если память уже выделена то выход с ошибкой
	if(m_bAllocated == true)
		//объект уже создан
		return NN_EXIST;
	int i, j;
	//выделение памяти
	try
	{
		bool flag;
		m_nLayer = nLayer;
		//выделение памяти m_pLayer
		m_pLayer = new int [m_nLayer];
		if(pLayer == NULL)
			throw 0;
		//копирование исходных данных
		for(i = 0; i < m_nLayer; i++)
			m_pLayer[i] = pLayer[i];
		//выделение памяти m_pWeight
		m_pWeight = new double ** [m_nLayer-1];
		if(m_pWeight == NULL)
			throw 1;
		//выделение памяти m_pWeight[i]
		flag = true;
		for(i = 0; i < m_nLayer - 1; i++)
		{
			m_pWeight[i] = NULL;
			m_pWeight[i] = new double * [m_pLayer[i] + 1];
			if(m_pWeight[i] == NULL)
				flag = false;
		}
		if(flag == false)
			throw 2;
		//выделение памяти m_pWeight[i][j]
		flag = true;
		for(i = 0; i < m_nLayer - 1; i++)
		for(j = 0; j < m_pLayer[i] + 1; j++)
		{
			m_pWeight[i][j] = NULL;
			m_pWeight[i][j] = new double [m_pLayer[i+1]];
			if(m_pWeight[i][j] == NULL)
				flag = false;
		}
		if(flag == false)
			throw 3;
		//выделение памяти m_pSignal;
		m_pSignal = new double * [m_nLayer];
		if(m_pSignal == NULL)
			throw 4;
		//выделение памяти m_pSignal[i]
		flag = true;
		for(i = 0; i < m_nLayer; i++)
		{
			m_pSignal[i] = NULL;
			m_pSignal[i] = new double [m_pLayer[i]];
			if(m_pSignal[i] == NULL)
				flag = false;
		}
		if(flag == false)
			throw 5;
		//выделение памяти m_pSigma;
		m_pSigma = new double * [m_nLayer - 1];
		if(m_pSigma == NULL)
			throw 6;
```

```
		//выделение памяти m_pSigma[i]
		flag = true;
		for(i = 0; i < m_nLayer - 1; i++)
		{
			m_pSigma[i] = NULL;
			m_pSigma[i] = new double [m_pLayer[i+1]];
			if(m_pSigma[i] == NULL)
				flag = false;
		}
		if(flag == false)
			throw 7;
	}
	//если произошла ошибка выделения памяти
	catch(int error)
	{
		//обработка ошибок, break не используется, освобождается память до шага error
		switch(error)
		{
		case 7:
			for(i = 0; i < m_nLayer - 1; i++)
				delete [] m_pSigma[i];
			delete [] m_pSigma;
			m_pSigma = NULL;
		case 6:
		case 5:
			for(i = 0; i < m_nLayer; i++)
				delete [] m_pSignal[i];
			delete [] m_pSignal;
			m_pSignal = NULL;
		case 4:
		case 3:
			for(i = 0; i < m_nLayer - 1; i++)
			for(j = 0; j < m_pLayer[i] + 1; j++)
				delete [] m_pWeight[i][j];
		case 2:
			for(i = 0; i < m_nLayer - 1; i++)
				delete [] m_pWeight[i];
			delete [] m_pWeight;
			m_pWeight = NULL;
		case 1:
			delete [] pLayer;
			pLayer = NULL;
		case 0:
			//память не выделена, выход с ошибкой
			m_bAllocated = false;
			//не хватает памяти для выделения
			return NN_NO_MEMORY;
		}
	}
	//устанавливаем признак успешного выделения памяти и выходим
	m_bAllocated = true;
	//операция прошла успешно
	return NN_OK;
}
//---------------------------------------------------------------------------
int CNeuroNet::Load(LPCTSTR filename)
{
	//если память уже выделена то выход с ошибкой
	if(m_bAllocated == true)
		//объект уже создан
		return NN_EXIST;
	try
	{
		CString line;
```

```
		int i, j, k;
		//открываем файл
		CStdioFile fin(filename, CFile::modeRead | CFile::typeText);
		//чтение заголовка
		fin.ReadString(line);
		if(line != "нейросеть")
			throw(1);
		//чтение количества слоев
		fin.ReadString(line);
		int nLayer = atoi(line);
		//создание массива количества нейронов в слое
		int * pLayer = new int [nLayer];
		if(pLayer == NULL)
			throw(2);
		//чтение количества нейронов в слое
		for(i = 0; (i < nLayer); i++)
		{
			fin.ReadString(line);
			pLayer[i] = atoi(line);
		}
		if(i != nLayer)
			throw(3);
		//создание нейросети
		if(Create(nLayer, pLayer) != NN_OK)
			throw(4);
		delete [] pLayer;
		//чтение m_pWeight[]
		for(i = 0; i < m_nLayer - 1; i++)
		{
			//веса из %d слоя в %d слой
			fin.ReadString(line);
			for(j = 0; j < m_pLayer[i] + 1; j++)
			{
				fin.ReadString(line);
		  for(k = 0; k < m_pLayer[i+1]; k++)
				{
					m_pWeight[i][j][k] = atof(line);
					line.Delete(0, line.Find('\t') + 1);
				}
			}
		}
	}
	catch(CFileException * error)
	{
		if(error->m_cause == CFileException::endOfFile)
			//ошибка чтения файла
			return NN_ERR_FILE_READ;
		else
			//ошибка открытия файла
			return NN_ERR_FILE_OPEN;
	}
	catch(int error)
	{
		if(error == 1)
			//ошибка чтения файла - неверный формат
			return NN_ERR_FILE_READ;
		else
			//ошибки выделения памяти
			return NN_NO_MEMORY;
	}
	//операция прошла успешно
	return NN_OK;
}
//---------------------------------------------------------------------------
```

```
int CNeuroNet::Save(LPCTSTR filename) const
{
	//если память не выделена то выход
	if(m_bAllocated == false)
		//объект не создан
		return NN_NOT_EXIST;
	try
	{
		CString line, word;
		int i, j, k;
		//создаем файл
		CStdioFile fout(filename, CFile::modeCreate | CFile::modeWrite |
CFile::typeText);
		//заголовок
		fout.WriteString("нейросеть\n");
		//запись m_nLayer;
		line.Format("%d\tколичество слоев\n", m_nLayer);
		fout.WriteString(line);
		//запись m_pLayer[]
		for(i = 0; i < m_nLayer; i++)
		{
			line.Format("%d\tколичество нейронов в слое %d\n", m_pLayer[i], i);
			fout.WriteString(line);
		}
		//запись m_pWeight[]
		for(i = 0; i < m_nLayer - 1; i++)
		{
			line.Format("веса из %d слоя в %d слой\n", i, i+1);
			fout.WriteString(line);
			for(j = 0; j < m_pLayer[i] + 1; j++)
			{
				line = "";
				for(k = 0; k < m_pLayer[i+1]; k++)
				{
					word.Format("%20.17f\t", m_pWeight[i][j][k]);
					line += word;
				}
				line += "\n";
				fout.WriteString(line);
			}
		}
	}
	catch(CFileException * error)
	{
		if(error->m_cause == CFileException::diskFull)
			//ошибка записи файла
			return NN_ERR_FILE_WRITE;
		else
			//ошибка открытия файла
			return NN_ERR_FILE_OPEN;
	}
	//операция прошла успешно
	return NN_OK;
}
//-----------------------------------------------------------------------------
int CNeuroNet::SetRandomWeight(const double * pLeft, const double * pRight)
{
	//если память не выделена то выход
	if(m_bAllocated == false)
		//объект не создан
		return NN_NOT_EXIST;
	double left, right;
	//заполнение случайными значениями
	srand((unsigned)time(NULL));
```

```
	for(int i = 0; i < m_nLayer - 1; i++)
	{
		//условие для выбора границ по умолчанию
		if((pLeft == NULL) || (pRight == NULL) || pLeft[i] >= pRight[i])
		{
			left = 0;
			right = 1.0 / m_pLayer[i];
		}
		for(int j = 0; j < m_pLayer[i] + 1; j++)
		for(int k = 0; k < m_pLayer[i+1]; k++)
		{
			m_pWeight[i][j][k] = left + (right - left) * rand() / RAND_MAX;
		}
	}
	//операция прошла успешно
	return NN_OK;
}
//---------------------------------------------------------------------------
int CNeuroNet::Forward(const double * pInput, double * pOut)
{
	//если память не выделена то выход
	if(m_bAllocated == false)
		//объект не создан
		return NN_NOT_EXIST;
	int i, j, k;
	//просчет сигналов по слоям
	for(i = 0; i < m_nLayer; i++)
	//по нейронам в слое
	for(j = 0; j < m_pLayer[i]; j++)
	{
		if(i == 0)
		{
			//для нулевого слоя - копия входного сигнала
			m_pSignal[i][j] = pInput[j];
		}
		else
		{
			//просчет сигналов по выходам нейронов предыдущего слоя
			m_pSignal[i][j] = 0;
			//как взвешенная сумма выходов нейрона предыдущего слоя
			for(k = 0; k < m_pLayer[i-1] + 1; k++)
				m_pSignal[i][j] += m_pWeight[i-1][k][j] * (k == m_pLayer[i-1] ? 1
: m_pSignal[i-1][k]);
			//с функцией активации
			m_pSignal[i][j] = Activation(m_pSignal[i][j]);
		}
	}
	//если необходимо выдать выходной сигнал
	if(pOut != NULL)
		//копируем выходной сигнад
		for(i = 0; i < m_pLayer[m_nLayer - 1]; i++)
			pOut[i] = m_pSignal[m_nLayer - 1][i];
	//операция прошла успешно
	return NN_OK;
}
//---------------------------------------------------------------------------
int CNeuroNet::BackPropagation(const double * pOut, double NU)
{
	//если память не выделена то выход
	if(m_bAllocated == false)
		//объект не создан
		return NN_NOT_EXIST;
	int i, j, k;
	//просчет сигмы по слоям
```

```
	for(i = m_nLayer - 1; i > 0; i--)
	//по нейронам в слое
	for(j = 0; j < m_pLayer[i]; j++)
	{
		if(i == m_nLayer - 1)
			//для последнего - дельта умноженая на производную функции активации
			m_pSigma[i-1][j] = Derivative(m_pSignal[i][j]) * (pOut[j] -
m_pSignal[i][j]);
		else
		{
			//просчет сигмы по сигме следующего слоя
			m_pSigma[i-1][j] = 0;
			//как взвешенная сумма сигм нейронов следующего слоя
			for(k = 0; k < m_pLayer[i+1]; k++)
				m_pSigma[i-1][j] += m_pWeight[i][j][k] * m_pSigma[i][k];
			//с производной функции активации
			m_pSigma[i-1][j] *= Derivative(m_pSignal[i][j]);
		}
	}
	//коррекция матрицы весовых коэффициентов
	for(i = 0; i < m_nLayer - 1; i++)
	for(j = 0; j < m_pLayer[i] + 1; j++)
	for(k = 0; k < m_pLayer[i+1]; k++)
		m_pWeight[i][j][k] += NU * m_pSigma[i][k] * (j == m_pLayer[i] ? 1 :
m_pSignal[i][j]);
	//операция прошла успешно
	return NN_OK;
}
//---------------------------------------------------------------------------
int CNeuroNet::LoadLearn(LPCTSTR filename)
{
	//если память не выделена то выход с ошибкой
	if(m_bAllocated == false)
		//объект не создан
		return NN_NOT_EXIST;
	int i, j;
	CString line;
	try
	{
		//открываем файл
		CStdioFile fin(filename, CFile::modeRead | CFile::typeText);
		//чтение заголовка
		fin.ReadString(line);
		if(line != "обучающее множество")
			throw(1);
		//чтение длины обучающего множества
		fin.ReadString(line);
		m_nLearn = atoi(line);
		//создание обучающего множества
		m_pLearn = new double * [m_nLearn];
		if(m_nLearn == NULL)
			throw(2);
		bool flag = true;
		for(i = 0; i < m_nLearn; i++)
		{
			m_pLearn[i] = NULL;
			m_pLearn[i] = new double [m_pLayer[0] + m_pLayer[m_nLayer - 1]];
			if(m_pLearn[i] == NULL)
				flag = false;
		}
		if(flag = false)
			throw(3);
		//чтение m_pLearn[]
		for(i = 0; i < m_nLearn; i++)
```

```
                {
                        fin.ReadString(line);
                for(j = 0; j < m_pLayer[0] + m_pLayer[m_nLayer - 1]; j++)
                        {
                                m_pLearn[i][j] = atof(line);
                                line.Delete(0, line.Find('\t') + 1);
                        }
                }
        }
        catch(CFileException * error)
        {
                if(error->m_cause == CFileException::endOfFile)
                        //ошибка чтения файла
                        return NN_ERR_FILE_READ;
                else
                        //ошибка открытия файла
                        return NN_ERR_FILE_OPEN;
        }
        catch(int error)
        {
                switch(error)
                {
                case 1:
                        //ошибка чтения файла - неверный формат
                        return NN_ERR_FILE_READ;
                case 3:
                        for(i = 0; i < m_nLearn; i++)
                                delete [] m_pLearn[i];
                        delete [] m_pLearn;
                case 2:
                        //ошибки выделения памяти
                        return NN_NO_MEMORY;
                }
        }
        //операция прошла успешно
        return NN_OK;
}
//--------------------------------------------------------------------------------
int CNeuroNet::Learn(int epoch, double NU, double * eps)
{
        //если память не выделена то выход с ошибкой
        if(m_bAllocated == false)
                //объект не создан
                return NN_NOT_EXIST;
        if(m_nLearn == 0)
                //не зашружено обучающее множество
                return NN_NO_LEARN;
        int i, j;
        //обучение
        for(i = 0; i < epoch; i++)
        for(j = 0; j < m_nLearn; j++)
        {
                Forward(m_pLearn[j]);
                BackPropagation(m_pLearn[j] + m_pLayer[0], NU);
        }
        //просчет погрешности
        *eps = 0;
        for(i = 0; i < m_nLearn; i++)
        {
                Forward(m_pLearn[i]);
                for(j = 0; j < m_pLayer[m_nLayer - 1]; j++)
                if(*eps < fabs(m_pLearn[i][j + m_pLayer[0]] - m_pSignal[m_nLayer - 1][j]))
                        *eps = fabs(m_pLearn[i][j + m_pLayer[0]] - m_pSignal[m_nLayer - 1][j]);
        }
```

```
	//операция прошла успешно
	return NN_OK;
}
//---------------------------------------------------------------------------
CNeuroNet::~CNeuroNet()
{
	if(m_bAllocated == true)
	{
		int i, j;
		//удаление памяти m_pWeight[][][]
		for(i = 0; i < m_nLayer - 1; i++)
		{
			for(j = 0; j < m_pLayer[i] + 1; j++)
				delete [] m_pWeight[i][j];
			delete [] m_pWeight[i];
		}
		delete [] m_pWeight;
		//удаление памяти m_pSignal[][]
		for(i = 0; i < m_nLayer; i++)
			delete [] m_pSignal[i];
		delete [] m_pSignal;
		//удаление памяти m_pSigma[][]
		for(i = 0; i < m_nLayer - 1; i++)
			delete [] m_pSigma[i];
		delete [] m_pSigma;
		//удаление памяти m_pLayer[]
		delete [] m_pLayer;
		//удаление обучающего множества
		for(i = 0; i < m_nLearn; i++)
			delete [] m_pLearn[i];
		delete [] m_pLearn;
	}
}
//---------------------------------------------------------------------------
double CNeuroNet::Activation(double x)
{
	return 1 / (1 + exp(-x));
}
//---------------------------------------------------------------------------
double CNeuroNet::Derivative(double x)
{
	return x * (1 - x);
}
//---------------------------------------------------------------------------
```