УДК 004.8

Ф. А. Галимянов, Ф. М. Гафаров, Н. А. Емельянова

ИСПОЛЬЗОВАНИЕ СРЕДЫ MICROSOFT VISUAL STUDIO И ЯЗЫКА ПРОГРАММИРОВАНИЯ C++ ДЛЯ МОДЕЛИРОВАНИЯ ДИНАМИКИ ОБРАЗОВАНИЯ НЕЙРОННОЙ СЕТИ

Ключевые слова: нейрон, функция процесс, начало отсчета, глобальные переменные, локальные переменные, бинарный файл, трехмерный массив.

В работе рассматривается программная реализация образования нейронных сетей (процесс роста аксонов) на примере биологической модели. В качестве среды програмирования выбраны Microsoft visual studio 2010 и язык C++. Приводится алгоритм реализации биологической модели. Программа состоит из двух основных частей. Первая часть - это численная реализация модели, вторая - это графическая интерпретация вычислений с использованием библиотек OpenGL. Описаны функции и переменные, использованные в программе, наиболее важные константы, необходимые для реализации модели.

Keywords: neuron, function process, initiated report, global variables, local variables, binary, three-dimensional array.

The software implementation of neural networks formation (process of axon's growth) is considered in the framework of the biological model. The programming language C++ and programming environment Microsoft visual studio 2010 are used. We describe the stages of the biological model realization. The program consists of two main parts. The first part is a numerical implementation of the model, the second part is a graphical interpretation of the calculation using the libraries OpenGL. The most important constants needed for our model, and location of neurons, the dynamics of their states during the evolution of the system are discussed.

Введение

Существует множество способов программно реализовать искусственные нейронные сети. Мы для реализации своего проекта используем среду программирования Microsoft Visual Studio и язык программирования C++. Внешний интерфейс состоит из одной формы, реализованной на основе графических библиотек OpenGL. При запуске приложения начинается вычисление координат для перемещения конуса роста кончиков аксонов нейронов. После каждой итерации, т.е. вычисления позиций кончиков роста нейронов, все изменения сразу же реализуются графически. Этот метод дает возможность в реальном времени наблюдать за изменениями, происходящими в системе.

Подключаемые библиотеки

С помощью директивы препроцессора «#include» подключены следующие библиотеки: #include "stdafx.h", #include "gl\glut.h", #include "math.h". Здесь: "stdafx.h" - стандартная библиотека VS, glut.h – заголовочный файл для подключения графической библиотеки OpenGL, math.h – библиотека математических функций.

Глобальные константы

В начале программы определены целочисленные константы: int const Kol_Neu = 9, mernost = 2, iterat_end = 100; Здесь: const Kol_Neu = 9 это количество нейронов в модели, mernost = 2 мерность модели (в данном случае двухмерная модель), iterat_end = 100 количество итераций, которые реализует система.

Далее определены все действительные константы, используемые в программе: double const Jpor= 0.51, kdegr = 10e-3, Dif2 = 6e-11, mash = 1e-4, lamb = 4e-5, alpha = 1e-5, tau = 10, diam_neuron = 3e-5, pi = 3.14; double const time_step = 1, timeRemmAktiv = 600. Здесь: Jpor = 0.51 - пороговое значение, kdegr = 10e-3 - коэффициент деградации, Dif2 = 6e-11 - коэффициент диффузии, lamb = 4e-5 , tau = 10 - коэффициент чувствительности и подвижности аксона, diam_neuron = 3e-5 - коэффициент затухания активности и диаметр нейрона в метрах, mash = 1e-4 - масштаб в метрах (величина на которую умножаются расстояние между нейронами), alpha = 1e-5 - параметр описывающий количество вещества выпускаемое в межнейронное пространство за одну секунду [1-4], time_step = 1 – шаг итерации, timeRemmAktiv = 6000 время памяти активности.

Глобальные переменные

Глобальные переменные выбраны в основном в виде массивов. Double neuron [Kol_Neu] [mernost], axon_is_live [Kol_Neu], ExtJ [Kol_Neu], time = 0,Date [Kol_Neu] [3] [iterat_end+1]; int iterat=0, Vesa [Kol_Neu] [Kol_Neu]. Двухмерный массив «double neuron [Kol_Neu] [mernost]» хранит данные о координатах всех нейронов системы. Первый индекс [Kol_Neu] определяет количество нейронов в системе, второй индекс [mernost] определяет мерность системы. Одномерный массив ExtJ [Kol_Neu] содержит внешнюю активность (импульс или внешний раздражитель). Начальный момент времени записан в переменной time=0. Трехмерный массив Date [Kol_Neu] [3] [iterat_end+1] самый громоздкий, порядка нескольких гигабайт.

Первый индекс определяет количество нейронов, второй индекс рассматривается в пространстве [5] и в плоскости [6]. Третий индекс - это порядковая итерация, для недопущения переполнения массива берется на единицу больше. Данный массив хранит все координаты всех аксонов и активность всех нейронов в каждый момент итерации. Этот массив очень удобен для хранения на диске в виде бинарного файла и последующего считывания. Одномерный массив axon_is_live [Kol_Neu]

определяет, активен ли аксон данного нейрона. Итерация обычно начинается с нуля int iterat=0. Двумерный массив Vesa [Kol_Neu] [Kol_Neu] определяет веса межнейронной связи.

Функция инициализации

```
void initsaliz(void)
{neuron[0][0] = - 2.5*mash;  neuron[0][1] =  2.5*mash;
neuron[1][0] = - 2.5*mash;  neuron[1][1] =    0.0;
neuron[2][0] = - 2.5*mash;  neuron[2][1] =  - 2.5*mash;
neuron[3][0] =   0.0;           neuron[3][1] =    2.5*mash;
neuron[4][0] =   0.0;          neuron[4][1] =    0.0;
neuron[5][0] =   0.0;          neuron[5][1] =  - 2.5*mash;
neuron[6][0] =   2.5*mash; neuron[6][1] =    2.5*mash;
neuron[7][0] =   2.5*mash;  neuron[7][1] =    0.0;
neuron[8][0] =   2.5*mash;  neuron[8][1] =  - 2.5*mash;

for(int i=0; i<Kol_Neu; i++)
        {       Date[i][0][iterat]=neuron[i][0];
                Date[i][1][iterat]=neuron[i][1];
                Date[i][2][iterat]=0;
                ExtJ[i]=0;
                axon_is_live[i]=true;
                for(int j=0; j<Kol_Neu; j++)
                {       Vesa[i][j]=0;}}}
```

Эта функция отвечает за инициализацию массивов, которая необходима для начала работы программы. Сначала необходимо задать координаты расположения нейронов. Нейронов девять [7], рассматриваемый случай двухмерный, поочередно задаем каждому элементу массива neuron [Kol_Neu] [mernost] конкретное значение абсциссы и ординаты. Значение координат умножается на величину mash, определяющую рассматриваемый масштаб.

neuron[0][0] = - 2.5*mash; - Абсцисса.
neuron[0][1] = 2.5*mash; - Ордината.

После инициализации месторасположения нейронов приступаем к инициализации начального месторасположения конуса роста аксонов. В каждый момент времени конусы роста нейронов хранятся в массиве. В нашей модели в начальный момент времени координата конуса роста аксонов совпадает с центром нейрона, поэтому начальным координатам аксона присваиваем координаты нейрона (аксон вырастает из нейрона).

Date[i][0][iterat]=neuron[i][0]; - Абсцисса.
Date[i][1][iterat]=neuron[i][1]; - Ордината.

Также для начального момента времени мы задаем нулевые значения активности. Date[i][2][iterat]=0. Внешние сигналы к моменту первой итерации отсутствуют: ExtJ[i]=0. Все аксоны нейронов активны: axon_is_live[i]=true. Обнуляем значение весов: Vesa[i][j]=0. Веса межнейронных связей являются двухмерным массивом для его инициализации требуются вложенные циклы. Все вышеперечисленные действия необходимо выполнить для каждого нейрона, т.е Kol_Neu раз.

Функция движения (move)

С помощью функции движения рассчитывается вектор движения [8] конуса роста аксона i нейрона n_i. Вектор представлен в виде проекций на оси координат. Каждая проекция рассчитывается по отдельности. Функция использует следующие переменные: double r_i - величина проекции рассчитываемой в данный момент (абсцисса или ордината), double J_i – величина активности нейрона, для аксона которого рассчитывается величина проекции на этом шаге итерации, int komp – номер проекции (komp=0 абсцисса, komp=1 ордината), double rx – проекция абсциссы на данном шаге итерации, double ry – проекция ординаты на данном шаге итерации, double tb – время начала отсчета, int iterat_loc – начало итерации, не равно нулю в том случае, когда данные прошлого теряют актуальность.

double move (double r_i, double J_i, int komp, double rx, double ry, double tb, int iterat_loc)

Для работы функции инициализируем локальные переменные: double gradient=0 – значение градиента концентрации вещества, r_ri – расстояние между конусом роста аксона и нейроном (рассчитывается для каждого нейрона), F_J – значение ступенчатой функции.

Для вычисления градиента концентрации в любой точке пространства необходимо учесть влияние всех нейронов. В рассматриваемой системе находятся нейроны, которые влияют на аксоны путем испускания вещества. Для вычисления градиента концентрации в точке нахождения конуса роста аксона суммируется влияние всех нейронов с учетом их расстояния от конкретного аксона и времени, прошедшего с момента испускания вещества. При выполнении цикла перебираются все нейроны:

```
for(int j=0; j<Kol_Neu; j++)
{
```

Вычисляется расстояние между рассматриваемыми аксоном и нейроном:

```
r_ri = pow (neuron[j][0] - rx, 2) +
        pow( neuron[j][1] - ry, 2).
```

Инициализируем переменную для перебора значений активности Date[j][2][i] нейрона в разные моменты времени:

```
int i=iterat_loc;
```

Инициализируем переменную начала отчета времени:

```
double t=tb.
```

Перебираем все моменты времени на каждом шаге time_step:

```
while(t < time)
{
```

Вычисляем градиент. Суммируем все компоненты градиента для каждого момента времени:

```
gradient += - (Date[j][2][i] *
(r_i – neuron[j][komp]))/
(8*pi*Dif2*Dif2*pow(( time - t),2))*
exp( -kdegr * ( time - t ) - r_ri / ( 4 * Dif2 *( time - t))).
```

Счетчик итерации:

```
i++.
```

Счетчик времени:

```
t+=time_step;
}
}
```

В зависимости от порогового значения Jpor, функция активации F_J принимается равной либо единице, либо нулю.

```
If ((Jpor-J_i)>0){F_J = 1;}
else      {F_J = 0;}
```

Для получения значения смещения проекции радиус-вектора движения кончика аксона умножаем градиент концентрации на константы и ступенчатую функцию:

```
moveing=alpha*lamb*F_J*gradient;
return moveing;
}
```

Функция вычисления активности (Aktiv)

Задачей данной функции является вычисление изменения активности каждого нейрона для следующего шага итерации. Для работы функции задаются переменные double J_t – активность нейрона в данном шаге итерации, double Ext_J_t – внешний активационный сигнал, int j – порядковый номер нейрона, для которого рассчитывается активность.

```
double Aktiv(double J_t, double Ext_J_t, int j)
{
```

Также используются следующие локальные переменные:

```
double  Sum=0, func,
```

где Sum - это математическая сумма всех входящих сигналов после их взвешивания, func -это функция активации.

При работе цикла перебираются все нейроны системы, и суммируется их вклад в активность рассчитываемого нейрона:

```
for(int i=0; i < Kol_Neu; i++)
 {
```

Обязательно задается условие невозможности стыковки аксона со своим нейроном:

```
if(i!=j)
        {
```

Суммируются все взвешенные входящие сигналы:

```
Sum+= Vesa[i][j] * Date[i][2][iterat];
        }
}
```

Полученное значение суммируется с внешним {

Полученная функция является аргументом функции активации:

```
if(func < 0) func = 0;
if(func > 1) func = 1;
```

Далее, согласно формуле вычисления активности, вычисляем изменение активности на следующем шаге итерации:

```
J_t=(double)1 / tau * (-J_t + func);
return J_t;
}
```

Функция процесс (protses)

Функция процесс - это узловая функция всей программы. Она является относительно большой. Ее задача - расчет и сохранение в массив Date[Kol_Neu][3][iterat_end+1] активности всех нейронов системы и перемещение их аксонов в каждый момент времени. Эта функция не возвращает никаких значений, она просто заполняет массив. Этот массив может занимать несколько гигабайт памяти. Размеры массива указывают, какое количество итераций можно сохранить на дисковый носитель в виде бинарного файла. Для вызова функции также не требуется никаких значений.

```
void protses(void)
{
```

Для работы функции необходимы локальные переменные: moveing – перемещение в следующем шаге итерации, max_moveing=1.5e-5 – максимально возможное перемещение (аксон не может двигаться быстрее), X_Proek – проекция перемещения на абсциссу, Y_Proek – проекция перемещения на ординату, ara – зависимость между рассматриваемым нейроном и аксоном, tb – начало отсчета.

```
double moveing, max_moveing=1.5e-5, X_Proek, Y_Proek, ara, tb
```

Для работы с двумерными массивами инициализируем переменную int j, и переменную для локальной итерации iterat_loc.

```
int j=0, iterat_loc
```

Если количество итераций не превысило максимально допустимого, функция производит следующую проверку

```
if(iterat<iterat_end)
{
```

Все основные составные части функции для читабельности разделены пунктирными линиями с подзаголовком.

Ниже следует часть кода с подзаголовком «girlyanda» (эта часть программы «включает» нейроны словно лампочки елочных гирлянд [9]). Внешний активационный сигнал через массив ExtJ[Kol_Neu] был подан нейронам с порядковыми номерами 2 и 4.

```
//----------------------girlyanda-------------------------------//
ExtJ[2]=1;
ExtJ[4]=1;
//--------------------------------------------------------------//
```

Далее следует обращение и работа с функций активности. Перебирая все нейроны в цикле, по методу Эйлера вычисляется активность для каждого нейрона системы. Для вычисления активности в следующем шаге итерации к активности на данном шаге прибавляется изменение активности, умноженное на шаг по времени.

```
//---------------------------------aktiv-----------------------//
for(int i=0; i<Kol_Neu; i++)
        {
Date[i][2][iterat+1]=Date[i][2][iterat]+
time_step*Aktiv(Date[i][2][iterat], ExtJ[i], i);
        }
//--------------------------------------------------------------//
```

Для вычисления смещения конуса роста аксона необходимо знать градиент концентрации вещества в месте расположения аксона. Для вычисления градиента концентрации нужно знать расстояние до каждого конкретного нейрона системы от этой точки и активность нейронов в каждый момент времени с начала отсчета. Дело в том, что в функции движения (move) есть фрагмент: while (t <

time){…}. Здесь t - это начало отчета. С течением времени, при отдалении от начала отсчета, с каждым шагом итерации расчеты для получения значения градиента концентрации усложняются. Это связано с тем, что вещество, образованное нейронами, исчезает не сразу, а постепенно деградирует. Скорость зависит от коэффициента деградации. Поэтому активность нейрона, которая была несколько десятков минут назад, будет оказывать влияние на рост аксона в вычисляемом на данный момент шаге итерации. С течением времени влияние активности уже пассивного нейрона будет все меньше и меньше, и в какой-то момент времени, можно будет пренебречь данным влиянием, и взять за начало отсчета уже не ноль. По истечении определенного отрезка времени timeRemmAktiv (время актуальности памяти активности) полагаем, что начало отчета равно time_begin=time-timeRemmAktiv. Таким образом, рассматривается только определенный отрезок времени, который находится экспериментально и служит направлением для дальнейших исследований.

Если не отбрасывать прошлую память после некоторого необходимого отрезка времени, то количество вычислений с каждой итерацией будет возрастать по арифметической прогрессии, и скорость эволюции системы будет падать.

//----------time_begin----------//

Если текущее время больше, чем время сохранения памяти активности то

```
if (time>timeRemmAktiv)
{
```

Время начала отсчета:

```
tb=time-timeRemmAktiv;
```

Итерацию при вычислении градиента начинать с

```
iterat_loc=iterat-(timeRemmAktiv/time_step);
}
else
{
```

Если текущее время меньше, чем время сохранения активности то

```
tb=0;
iterat_loc=0;
}
```

//----------//

Далее в программе идет блок по непосредственному управлению движением конуса роста аксонов. Также перебираем все нейроны в цикле:

```
for(int i=0; i<Kol_Neu; i++)
{
```

Условие активности аксона, т.е. момент времени до стыковки с нейроном:

```
if(axon_is_live[i])
{
```

Методом Эйлера вычисляем проекции смещения конуса роста аксона. При вычислении вызываем функцию движения (move). При инициализации функции используем следующие переменные: Date[i][0][iterat] – проекция абсциссы на данном шаге итерации (для нахождения X_Proek (проекции абсциссы на следующем шаге)), Date[i][2][iterat] – активность нейрона, для аксона которого мы вычисляем смещение, 0 –номер проекции, означает абсциссу (1 - ордината), Date[i][0][iterat] – проекция абсциссы на данном шаге, Date[i][1][iterat] – проекция ординаты на данном шаге, tb – время начала отсчета, iterat_loc – начало итерации для вычисления градиента.

//----------move----------//

```
X_Proek=time_step*move(Date[i][0][iterat],
Date[i][2][iterat], 0, Date[i][0][iterat], Date[i][1][iterat],
tb, iterat_loc);
Y_Proek=time_step*move(Date[i][1][iterat],
Date[i][2][iterat], 1, Date[i][0][iterat], Date[i][1][iterat],
tb, iterat_loc);
```

//----------//

Как было указано выше, аксон не может двигаться бесконечно быстро. Есть биофизические факторы, которые ограничивают скорость роста аксона. В представленной модели также присутствует максимальная скорость max_moveing, быстрее которой аксон расти не будет. Поэтому в функции «процесс» есть часть кода, названная условно «tormoz». Задача этого фрагмента «притормозить» слишком быстрые аксоны, не давая двигаться быстрее максимально допустимого значения.

//----------tormoz----------//

После вычислений проекций смещения аксона необходимо вычислить модуль смещения радиус-вектора и сравнить его с максимально допустимым.

```
moveing=pow(pow(X_Proek,2)+pow(Y_Proek,2),0.5).
```

Если модуль смещения радиус-вектора больше максимально допустимого, то используя свойство подобия прямоугольных треугольников, необходимо пропорционально уменьшить проекции до максимально допустимых.

```
if(moveing > max_moveing)
{
X_Proek = X_Proek*max_moveing/moveing;
Y_Proek = Y_Proek*max_moveing/moveing;
}
```

//----------//

После нормализации скорости можно завершить вычисления проекций методом Эйлера, прибавив к предыдущему значению проекций вновь вычисленное смещение проекций.

```
Date[i][0][iterat+1]=Date[i][0][iterat]+X_Proek;
Date[i][1][iterat+1]=Date[i][1][iterat]+Y_Proek;
```

Так получены координаты месторасположения конусов роста аксонов. Аксоны растут в сторону активных нейронов до момента стыковки.

При каждой итерации мы проверяем расстояние ara между конусами роста всех аксонов и координатами всех центров нейронов. Если это значение меньше чем 1.5e-5, то аксон престает быть активным, т.е. прекращает расти. За активность нейронов отвечает массив axon_is_live[i], для придания аксону пассивности нужно присвоить массиву ложное значение axon_is_live[i]=false. В нашей модели, если аксон стал пассивным, он уже больше не активизируется.

//----------axon_is_live----------//

Цикл выполняется только при активном аксоне. В цикле проверяются расстояния между конусом

роста аксона и всеми нейронами в системе, кроме своего собственного.

```
while((j <= Kol_Neu) && axon_is_live[i])
        {
```

Вычисляется расстояние между конусом роста аксона и центром нейрона:

```
ara = pow(pow(Date[i][0][iterat+1] - neuron[j][0],2) +
pow(Date[i][1][iterat+1] - neuron[j][1],2),0.5);
```

Если расстояние ara больше, чем 1.5e-5 метров, и нейрон не свой собственный то:

```
if((ara < 1.5e-5) && (i!=j))
{
```

Аксон становится пассивным.

```
axon_is_live[i]=false;
```

Представленная модель динамична: рассматривается не «готовая» нейронная сеть, а образование новой нейронной сети. При образовании связей между нейронами, обязательно должны появиться весовые коэффициенты этих связей. Весовые коэффициенты образуются по следующей закономерности: если на момент стыковки аксона с нейроном нейрон был активен, то связь отрицательная (подавляющая), а если нейрон был пассивен, то связь положительная. В терминологии нейронных сетей есть понятие пороговой величины активности. Пороговая величина равна 0.51 условных единиц. Т.е., если у нейрона уровень активности больше чем 0.51, то он активен, а если меньше или равен, то - пассивен.

```
//----------Vesa-----------------------------------------//
```

Если аксон приближается достаточно близко к координатам центра нейрона и становится пассивным, то учитывается прошлая активность нейрона:

```
if(Date[j][2][iterat] > 0.51)
        {
    Связь отрицательна, если меньше 0.51;
                Vesa[i][j] = -1;
        }
else
        {
        Связь положительна, если больше 0.51;
                Vesa[i][j] = 1;
        }
//-----------------------------------------------------------//
```

Так переопределяется двумерный массив Vesa[i][j], хранящий весовые коэффициенты. После каждой новообразованной связи выводим его значение на экран консоли:

```
printf("%i\n", Vesa[i][j]);
        }
```

Прибавляем счетчику порядкового номера нейронов следующее значение.

```
                j++;
        }
```

После выхода из цикла обнуляем значения счетчика нейронов.

```
                j=0;
//-----------------------------------------------------------//
        }
```

Если на определенном шаге итерации аксоны уже были пассивны, т.е. состыковались с нейронами раньше, для них вычисление траектории движения не происходит.

```
else
        {
```

Для пассивных нейронов вместо вычисления координат радиус-вектора на каждом шаге итерации происходит присвоение их предыдущих координат.

```
                Date[i][0][iterat+1]=Date[i][0][iterat];
                Date[i][1][iterat+1]=Date[i][1][iterat];
        }
        }
```

Счетчик текущего времени:

```
time+=time_step;
```

Счетчик текущей итерации:

```
iterat++;
        }
```

Заключение

Показана программная реализация модели роста нейронной сети в реальном времени. Представленный метод компьютерного моделирования был реализован под платформу Microsoft Windows, которая наиболее распространена в научных лабораториях и учебных заведениях Российской Федерации. При реализации разработчик ограничился только наиболее популярными библиотеками, чтобы при распространении программного продукта не возникло проблем с инсталляцией. При сравнительном анализе с реальными аналогичными системами в природе результаты показали полную адекватность. Математическая модель процесса образования нейронной сети, построена на системе дифференциальных уравнений, в перспективе модель может быть уточнена видением дробного интегрирования[10],[11].

Литература

1. G. J. Goodhill, Diffusion in Axon Guidance. *European Journd of Neuroscience*, **9**, 1414–1421 (1997);
2. R. W. Gundersen, J. N. Barrett, Characterization of the Turning Response of Dorsal Root Neurites toward Nerve Growth Factor. *The Journal of cell biology*, **87**, 546–554 (1980);
3. W. Rosoff, J. S. Urbach, M. A. Esrik, R. G. McAllister, L. J. Richards, G. J. Goodhill, A new chemotaxis assay shows the extreme sensitivity of axons to molecular gradients. *Nature Neuroscience*, **7**, 6, 678-682 (2004);
4. T. P. Vogels, K. Rajan, L.F. Abbott, Neural Network Dynamics. *Annu. Rev. Neurosci,* **28**, 357-376(2005);
5. F. M. Gafarov, Self-wiring in neural nets of point-like cortical neurons fails to reproduce cytoarchitectural differences. *Journal of Integrative Neuroscience*, **5**, 2, 159-169 (2006);
6. F. M. Gafarov, N. R. Khusnutdinov, F. A. Galimyanov,
7. Morpholess neurons compromise the development of cortical connectivity. *Journal of Integrative Neuroscience*, **8**, 1, 1-14 (2009);
8. Ф.А. Галимянов, Ф.М. Гафаров, Н.Р. Хуснутдинов, Модель роста нейронной сети. *Математическое моделирование*, **23**, 00, 000-111 (2011);
9. Ф. А. Галимянов, Ф. М. Гафаров, Н. А. Емельянова, Численная реализация уравнения движения, кончика роста аксона нейрона, и выражения для нахождения распределения концентрации вещества, в

межнейронном пространстве. *Вестник Казанского технологического университета*, **17**, 23, 397-399(2014)
10. С. П. Плохотников, О. И. Богомолова, Д. С. Плохотников, В. А. Богомолов, О. Р. Плохотникова, М. С. Нурсубин, Математическое моделирование неизотермической двухфазной фильтрации с модифицированными относительными фазовыми проницаемостями. *Вестник Казанского технологического университета*, **16**, 21, 122-124, (2013)
11. Agachev J.R., Galimyanov A.F., About the convergence of the general projection polynomial method for a class of periodic fractional-integral equations. *Lobachevskii journal of mathematics,* **35**, 3, 211-217(2014)
12. Agachev J.R., Galimyanov A.F., On justification of general polynomial projection method for solving periodic fractional integral equations. *Lobachevskii journal of mathematics,* **36**, 2, 97-102(2015)

© **Ф. А. Галимянов** – асс. каф. информатики и прикладной математики КНИТУ, fanisgalimyanov@gmail.com; **Ф. М. Гафаров** - канд. физ.-мат. наук, доц. каф. информационных систем, отделение информационных технологий в гуманитарной сфере КФУ, fgafarov@yandex.ru; **Н. А. Емельянова** - канд. физ.-мат. наук, доц. каф. железнодорожной автоматика, телемеханики и связи, Московский государственный университет путей сообщения (МИИТ), Казанский филиал, prl70@mail.ru.

© **F. A. Galimyanov** - Assistant, Department of Computer Science and Applied Mathematics, KNRTU, fanisgalimyanov@gmail.com; **F. M. Gafarov** - PhD, Associate Professor, Department of Information Systems, Department of Information Technology in the humanitarian sphere, KFU, fgafarov@yandex.ru**; N. A. Emelyanova**, PhD., Associate Professor, Faculty Railway automatics, telemechanics and communication, Moscow State University of Railway Engineering (MIIT), Kazan Branch, prl70@mail.ru.