ПРИЛОЖЕНИЕ 1

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

# КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

НЕЙРОСЕТЕВЫЕ МЕТОДЫ АППРОКСИМИЗАЦИИ И ПРОГНОЗИРОВАНИЯ ФУНКЦИИ

ТЕКСТ ПРОГРАММЫ

КР.ИИ17.190289-01 81 00

## Листов: 10

|  |  |
| --- | --- |
|  |  |
| Руководитель | В.А. Головко |
| Выполнил | И.Г. Ковалевич |
| Консультант |  |
| по ЕСПД | В.А. Головко |
|  |  |
|  |  |

Брест, 2021

**СОДЕРЖАНИЕ**

Program.cs – главный файл

Runge\_Kutta.cs – решение системы уравнений методом Рунге-Кутты 4-го порядка

Dataset.cs – формирование и вывод датасета

NeuralNetwork.cs – нейронная сеть

myFile.cs – вывод результатов аппроксимации и тестирования в файл

**Program.cs:**

class Program

{

static void Main(string[] args)

{

const int hiddenCount = 7;

const double alpha = 0.00055;

const double Em = 0.1;

const int formLearnNum = 500;

const int formLearnSize = 1;

const int formTestNum = 400;

Runge\_Kutta r\_k = new Runge\_Kutta(formLearnNum, formTestNum);

Dataset dataset = new Dataset(r\_k.x, r\_k.y, r\_k.z, formLearnNum, formTestNum, formLearnSize);

NeuralNetwork neuralNetwork = new NeuralNetwork(hiddenCount, alpha, Em, dataset.getApproximationDataset(), dataset.getStandartDataset());

neuralNetwork.startApproximationCycle();

myFile.Write(neuralNetwork.show\_resultOutput(), dataset.getStandartDataset(), 0);

myFile.Write(neuralNetwork.functionPrediction(dataset.getForecastingDataset()), dataset.getStandartForecastingDataset(), 1);

}

}

**Runge\_Kutta.cs:**

public class Runge\_Kutta

{

const int M = 6, Q = 10;

const double T = 1.5;

const double h = 0.1;

int NumberOfForm, NumberOfTestForm;

public List<double> x, y, z;

public Runge\_Kutta(int NumberOfForm, int NumberOfTestForm)

{

this.NumberOfForm = NumberOfForm;

this.NumberOfTestForm = NumberOfTestForm;

x = new List<double> { 0.1 };

y = new List<double> { 0.1 };

z = new List<double> { 0.1 };

calculation();

}

/// <summary>

/// Производная функции X()

/// </summary>

/// <returns></returns>

private double X(double x, double y, double z)

{

return (M \* z \* Math.Exp(-Math.Pow(z, 2)) - x) / T;

}

/// <summary>

/// Производная функции Y()

/// </summary>

/// <returns></returns>

private double Y(double x, double y, double z)

{

return x - z;

}

/// <summary>

/// Производная функции Z()

/// </summary>

/// <returns></returns>

private double Z(double x, double y, double z)

{

return y - z / Q;

}

/// <summary>

/// Вычислить XYZ методом Рунге-Кутты 4-го порядка

/// </summary>

private void calculation()

{

double[,] k = new double[4, 3];

for (int i = 0; i < NumberOfForm + NumberOfTestForm; i++)

{

k[0, 0] = X(x[i], y[i], z[i]);

k[0, 1] = Y(x[i], y[i], z[i]);

k[0, 2] = Z(x[i], y[i], z[i]);

k[1, 0] = X(x[i] + k[0, 0] / 2 \* h, y[i] + k[0, 1] / 2 \* h, z[i] + k[0, 2] / 2 \* h);

k[1, 1] = Y(x[i] + k[0, 0] / 2 \* h, y[i] + k[0, 1] / 2 \* h, z[i] + k[0, 2] / 2 \* h);

k[1, 2] = Z(x[i] + k[0, 0] / 2 \* h, y[i] + k[0, 1] / 2 \* h, z[i] + k[0, 2] / 2 \* h);

k[2, 0] = X(x[i] + k[1, 0] / 2 \* h, y[i] + k[1, 1] / 2 \* h, z[i] + k[1, 1] / 2 \* h);

k[2, 1] = Y(x[i] + k[1, 0] / 2 \* h, y[i] + k[1, 1] / 2 \* h, z[i] + k[1, 1] / 2 \* h);

k[2, 2] = Z(x[i] + k[1, 0] / 2 \* h, y[i] + k[1, 1] / 2 \* h, z[i] + k[1, 1] / 2 \* h);

k[3, 0] = X(x[i] + k[2, 0] \* h, y[i] + k[2, 1] \* h, z[i] + k[2, 2] \* h);

k[3, 1] = Y(x[i] + k[2, 0] \* h, y[i] + k[2, 1] \* h, z[i] + k[2, 2] \* h);

k[3, 2] = Z(x[i] + k[2, 0] \* h, y[i] + k[2, 1] \* h, z[i] + k[2, 2] \* h);

x.Add(x[i] + (k[0, 0] + 2 \* k[1, 0] + 2 \* k[2, 0] + k[3, 0]) / 6 \* h);

y.Add(y[i] + (k[0, 1] + 2 \* k[1, 1] + 2 \* k[2, 1] + k[3, 1]) / 6 \* h);

z.Add(z[i] + (k[0, 2] + 2 \* k[1, 2] + 2 \* k[2, 2] + k[3, 2]) / 6 \* h);

}

}

}

**Dataset.cs:**

class Dataset

{

int NumberOfForm, NumberOfTestForm, formLearnSize;

private List<double> x, y, z;

public Dataset(List<double> x, List<double> y, List<double> z, int NumberOfForm, int NumberOfTestForm, int formLearnSize)

{

this.x = x;

this.y = y;

this.z = z;

this.NumberOfForm = NumberOfForm;

this.NumberOfTestForm = NumberOfTestForm;

this.formLearnSize = formLearnSize;

}

public double[,] getApproximationDataset()

{

double[,] MultiLearnData = new double[NumberOfForm - 1, formLearnSize \* 3];

for (int i = 0; i < MultiLearnData.GetLength(0); i++)

{

for (int j = 0, it = 0; j < MultiLearnData.GetLength(1); j += 3, it++)

{

MultiLearnData[i, j] = x[i + it];

MultiLearnData[i, j + 1] = y[i + it];

MultiLearnData[i, j + 2] = z[i + it];

}

}

return MultiLearnData;

}

public double[,] getStandartDataset()

{

double[,] MultiLearnData = new double[NumberOfForm - 1, 3];

for (int i = 0, j = formLearnSize; i < MultiLearnData.GetLength(0); i++, j++)

{

MultiLearnData[i, 0] = x[j];

MultiLearnData[i, 1] = y[j];

MultiLearnData[i, 2] = z[j];

}

return MultiLearnData;

}

public double[,] getForecastingDataset()

{

double[,] MultiTestData = new double[NumberOfTestForm, formLearnSize \* 3];

for (int i = 0, k = NumberOfForm - 1; i < MultiTestData.GetLength(0); i++, k++)

{

for (int j = 0, it = 0; j < MultiTestData.GetLength(1); j += 3, it++)

{

MultiTestData[i, j] = x[k + it];

MultiTestData[i, j + 1] = y[k + it];

MultiTestData[i, j + 2] = z[k + it];

}

}

return MultiTestData;

}

public double[,] getStandartForecastingDataset()

{

double[,] MultiTestData = new double[NumberOfTestForm, 3];

for (int i = 0, k = NumberOfForm - 1 + formLearnSize; i < MultiTestData.GetLength(0); i++, k++)

{

MultiTestData[i, 0] = x[k];

MultiTestData[i, 1] = y[k];

MultiTestData[i, 2] = z[k];

}

return MultiTestData;

}

}

**NeuralNetwork.cs:**

class NeuralNetwork

{

const int OutputLayerSize = 3;

const double k = 1;

private double[,] w1, w2, y1, y2;

private double[] T1,T2;

private double[,] x, t;

private double[,] error1, error2;

private double alpha, Em, Es;

private uint epoch;

private double activationOutputFunction(double S)

{

return k\*S;

}

private double activationHiddenFunction(int form, int i)

{

return 1 / (1 + Math.Pow(Math.E, -S(form, i)));

}

private double dHiddenFunction(int form, int i)

{

return y1[form, i] \* (1 - y1[form, i]);

}

private double dOutputFunction()

{

return 1;

}

public NeuralNetwork(in int hiddenCount, in double alpha, in double Em, in double[,] x, in double[,] standart)

{

this.alpha = alpha;

this.Em = Em;

this.x = x;

this.t = standart;

y1 = new double[standart.GetLength(0), hiddenCount];

y2 = new double[standart.GetLength(0), OutputLayerSize];

error1 = new double[standart.GetLength(0), hiddenCount];

error2 = new double[standart.GetLength(0), OutputLayerSize];

T1 = new double[hiddenCount];

T2 = new double[y2.GetLength(1)];

w1 = new double[x.GetLength(1), hiddenCount];

w2 = new double[hiddenCount, y2.GetLength(1)];

InitializationWT(hiddenCount);

}

void InitializationWT(int hiddenCount)

{

Random rnd = new Random();

for (int i = 0; i < x.GetLength(1); i++)

{

for (int j = 0; j < hiddenCount; j++)

{

w1[i, j] = (float)rnd.NextDouble();

}

}

for (int i = 0; i < hiddenCount; i++)

{

T1[i] = (float)rnd.NextDouble();

for (int j = 0; j < y2.GetLength(1); j++)

{

w2[i, j] = (float)rnd.NextDouble();

T2[j] = (float)rnd.NextDouble();

}

}

}

public void startApproximationCycle()

{

do

{

Es = 0;

Epoch();

epoch++;

Console.WriteLine($"Epoch: {epoch}, E = {Es}");

} while (Em < Es);

Console.WriteLine($"Epoch: {epoch}, E = {Es}");

}

private double outputY(int form, int j)

{

double sumW2 = 0;

for (int i = 0; i < y1.GetLength(1); i++)

{

sumW2 += w2[i, j] \* hiddenY(form, i);

}

return activationOutputFunction(sumW2 - T2[j]);

}

private double hiddenY(int form, int i)

{

return y1[form, i] = activationHiddenFunction(form, i);

}

private double S(int form, int i)

{

double sum = 0;

for (int k = 0; k < x.GetLength(1); k++)

{

sum += w1[k, i] \* x[form, k];

}

return sum - T1[i];

}

private double outputLayerError(int form, int j)

{

return y2[form, j] - t[form, j];

}

private double hiddenLayerError(int form, int i)

{

double sumError = 0;

for (int j = 0; j < y2.GetLength(1); j++)

{

sumError += error2[form, j] \* dOutputFunction() \* w2[i, j];

}

return sumError;

}

private void changeOutputW(int form)

{

for (int i = 0; i < y1.GetLength(1); i++)

{

for (int j = 0; j < y2.GetLength(1); j++)

{

w2[i, j] = w2[i, j] - alpha \* error2[form, j] \* dOutputFunction() \* y1[form, i];

}

}

}

private void changeHiddenW(int form)

{

for (int k = 0; k < x.GetLength(1); k++)

{

for (int i = 0; i < y1.GetLength(1); i++)

{

w1[k, i] = w1[k, i] - alpha \* error1[form, i] \* dHiddenFunction(form, i) \* x[form, k];

}

}

}

private void changeOutputT(int form)

{

for (int j = 0; j < y2.GetLength(1); j++)

{

T2[j] = T2[j] + alpha \* error2[form, j] \* dOutputFunction();

}

}

private void changeHiddenT(int form)

{

for (int i = 0; i < y1.GetLength(1); i++)

{

T1[i] = T1[i] + alpha \* error1[form, i] \* dHiddenFunction(form, i);

}

}

private double E(int form)

{

double sum = 0;

for (int j = 0; j < y2.GetLength(1); j++)

{

sum += Math.Pow((y2[form, j] - t[form, j]), 2);

}

return sum;

}

private void Epoch()

{

for (int form = 0; form < t.GetLength(0); form++)

{

for (int j = 0; j < y2.GetLength(1); j++)

{

y2[form, j] = outputY(form, j);

error2[form, j] = outputLayerError(form, j);

}

for (int i = 0; i < y1.GetLength(1); i++)

{

error1[form, i] = hiddenLayerError(form, i);

}

Es += E(form);

}

for (int form = 0; form < t.GetLength(0); form++)

{

changeOutputW(form);

changeHiddenW(form);

changeOutputT(form);

changeHiddenT(form);

}

Es \*= 0.5;

}

public double[,] show\_resultOutput()

{

for (int form = 0; form < t.GetLength(0); form++)

{

Console.WriteLine($"Epoch: {epoch}, Em: {Em}, E: {E(form)}");

Console.WriteLine("t\t \tout y");

for (int j = 0, i = 0; j < 1; j++)

{

if (i < y1.GetLength(1))

{

Console.WriteLine($"{t[form, j]}\t {y2[form, j]}");

i++;

}

else

{

Console.WriteLine($"{t[form, j]}\t \t {y2[form, j]}");

}

}

}

return y2;

}

public double[,] functionPrediction(double[,] x2)

{

x = x2;

double[,] prediction = new double[x2.GetLength(0), 3];

for (int form = 0; form < x.GetLength(0); form++)

{

for (int j = 0; j < prediction.GetLength(1); j++)

{

prediction[form, j] += outputY(form, j);

}

}

return prediction;

}

}

**myFile.cs:**

static class myFile

{

static public void Write(double[,] y, double[,] ey, int type)

{

System.Text.Encoding.RegisterProvider(System.Text.CodePagesEncodingProvider.Instance);

string[] filePath = { @"//.csv", @"//.csv" };

var csv = new StringBuilder();

for (int i = 0; i < ey.GetLength(0); i++)

{

var first = ey[i, 0].ToString();

var second = ey[i, 1].ToString();

var third = ey[i, 2].ToString();

var fourth = y[i, 0].ToString();

var fifth = y[i, 1].ToString();

var sixth = y[i, 2].ToString();

var newLine = string.Format($"{first};{second};{third};{fourth};{fifth};{sixth}");

csv.AppendLine(newLine);

}

File.WriteAllText(filePath[type], csv.ToString());

}

}