Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет»

Кафедра прикладной математики и информатики

УДК 519.2

**Прогнозирование надежности подшипника с помощью построения полиномов на основе данных о другой аварии**

Научно-исследовательская работа

|  |  |
| --- | --- |
|  | Работу выполнил студент  группы ПМИ-1  магистратуры  механико-математического факультета  Павлов Б. А.  Научный руководитель:  д-р техн. наук, канд. ф.-м. наук, профессор кафедры ПМиИ Гусев А. Л. |

Пермь 2023

**Аннотация**

В данной работе рассматривается возможность прогнозирования значений надежности подшипника, за счет построения полинома на основе другой аварии. Для проведения экспериментов было создано настольное приложение, позволяющее загружать данные из excel-файлов, вычислять коэффициенты надежности и находить наилучшие полиномы для выражения одной аварии через другую.

СОДЕРЖАНИЕ

[Введение 4](#_Toc133490716)

[1 Постановка задачи 7](#_Toc133490717)

[2 Создание настольного приложения для прогнозирования надежности подшипника 8](#_Toc133490718)

[2.1 Сбор данных и загрузка данных в приложении 8](#_Toc133490719)

[2.2 Нахождение значений надежности для второй аварии 9](#_Toc133490720)

[2.3 Нахождение отсечки во второй аварии 10](#_Toc133490721)

[2.4 Нахождение полинома для выражения значений второй аварии через эталонную аварию 11](#_Toc133490722)

[2.5 Сравнение коэффициентов надежности у реальных и предсказанных значений вибросигнала 12](#_Toc133490723)

[3 Исследование по нахождению наилучшей степени полинома 14](#_Toc133490724)

[Заключение 16](#_Toc133490725)

[Список литературы 17](#_Toc133490726)

[Приложение А 19](#_Toc133490727)

[Приложение Б 49](#_Toc133490728)

Введение

Одним из основных источников большинства механических неисправностей производственного оборудования является неисправность подшипника [1, 2]. Поэтому прогнозирование неисправностей подшипников и их устранение на ранних стадиях снизят затраты на нежелательную остановку производственного процесса.

Подшипником называется конструктивный узел машин и механизмов, поддерживающий или направляющий вращающийся вал или ось. Подшипник предназначен для уменьшения трения между движущейся и неподвижной частями машины, так как с трением связаны потери энергии, нагрев и износ [3]. На рисунке 1 представлено устройство однорядного радиального шарикоподшипника.

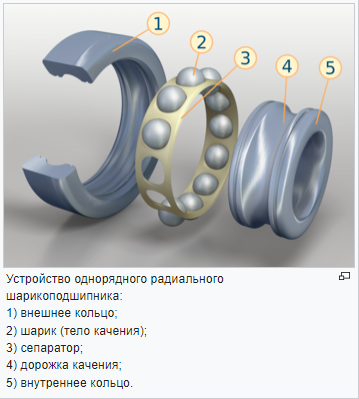


Рисунок –

– Устройство подшипника качения

В качестве наблюдений при прогнозировании аварии подшипника зачастую используется значение виброскорости подшипника [4] или СКЗ виброскорости [5].

Помимо СКЗ есть используются и другие характеристики значений. Так, в своей статье [6] авторы предлагают другие характеристики, которые показывают лучшую точность при прогнозировании. Авторы предлагают следующие характеристики:

* длина волны (waveform length);
* изменения знака наклона (slope sign changes);
* простой знак интеграла (simple sign integral);
* амплитуда Уиллисона (Willison amplitude);
* среднее абсолютное значение (mean absolute value);
* нулевое пересечение (zero crossing).

Одним из способов прогнозирования поломки подшипника является применение нейросетевых технологий. В своих статьях [7, 8] авторы показывают эффективность данного метода при прогнозировании поломки. Однако использование нейросетевых моделей несет в себе ряд недостатков [9, 10]:

* большие затраты во времени в процессе разработки структуры нейросети;
* большие вычислительные затраты при обучении нейросети, по сравнению с традиционными алгоритмами;
* работа по принципу «черного ящика»;
* необходимость в специалисте для обучения и настройки параметров нейросети;
* отсутствие конкретных правил для формирования структуры нейросети.

Также свою эффективность при прогнозировании аварии подшипника показывают более традиционные статистические и математические методы. В своих исследованиях [11, 12] авторы показывают эффективность подобных методов.

В данной работе будет проведено исследование возможности прогнозирования надежности подшипника по данным из другой аварии. Значение надежности будет находиться с помощью построенного полинома n-ой степени.

1. Постановка задачи

Целью данной работы было создание настольного приложения на языке C# с применением технологии Windows Forms для анализа и прогнозирования процента надежности подшипника на основе другой аварии (Листинг программы приведен в Приложении A).

Для достижения цели решались следующие задачи:

* изучение предметной области;
* изучение необходимых статистических методов и алгоритмов;
* сбор данных с СКЗ виброскорости;
* проектирование настольного приложения;
* программирование настольного приложения;
* проведение экспериментов для выявления наилучшей степени полинома.

1. Создание настольного приложения для прогнозирования надежности подшипника
   1. Сбор данных и загрузка данных в приложении

Данные, содержащие значения вибросигнала, были предоставлены НПП «ТИК». Данные были собраны с тестовых стендов, всего было собрано данных о шести различных авариях (Приложение Б). Одно значение вибросигнала представляет собой среднее квадратичное значение (СКЗ) от 65535 значений виброскорости в секунду.

В реализованном приложении присутствует возможность загрузки данных об авариях из excel-файлов (.xlsx, .xls). Загрузка данных происходит в фоновом режиме и не блокирует главный процесс приложения. На рисунке 2 представлен процесс загрузки данных из excel-файла в приложении.

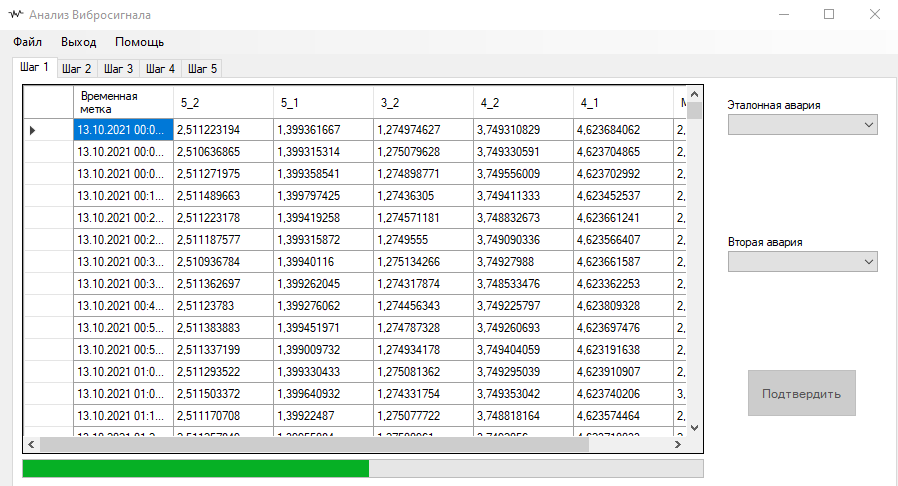


Рисунок –

– Загрузка дынных из файла

* 1. Нахождение значений надежности для второй аварии

На основе некоторого числа наблюдений, которое задается программно, вычисляется максимальный уровень сигнала, при котором наблюдается нормальная работа. Максимальный уровень вибросигнала при нормальной работе находится по формуле (1)

(1)

где – среднее значение вибросигнала в выбранном интервале, – стандартное отклонение вибросигнала в выбранном интервале.

Далее интервал от максимального уровня вибросигнала при нормальной работе до максимального вибросигнала (авария) делится на 100 частей, так находится цена одного деления надежности. Далее каждому отдельному значению вибросигнала в соответствие ставится значение надежности из полученного интервала. На рисунке 3 представлена вкладка приложения, в которой вибросигналу ставится в соответствие значение надежности.

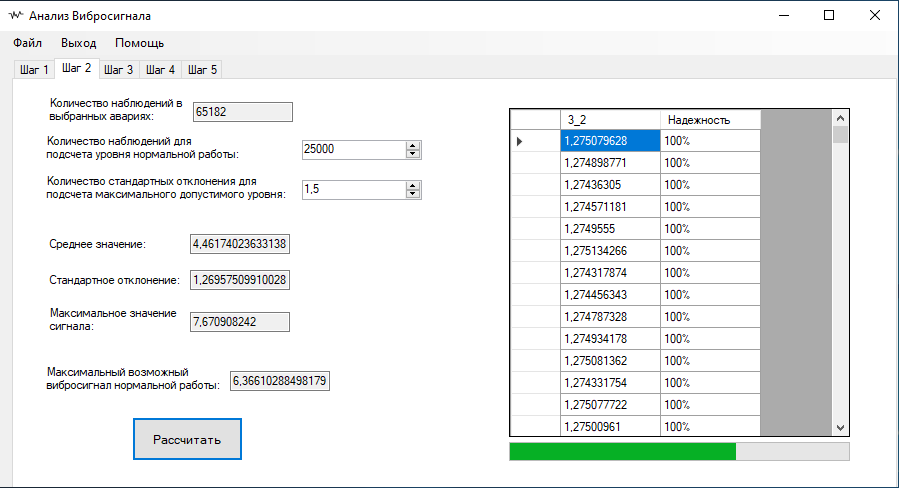


Рисунок –

– Вычисление надежности для вибросигнала

* 1. Нахождение отсечки во второй аварии

Из эталонной аварии берется достаточно большой интервал вибросигнала, который задается программно. Этот интервал накладывается на значения вибросигнала начала из второй аварии и между этими значениями находится коэффициент корреляции Пирсона (2)

(2)

где – средние значения вибросигнала.

Далее, с каждым шагом интервал во второй аварии сдвигается на одно значение и каждый раз находится коэффициент корреляции Пирсона. После этого вычисляется максимальное значение коэффициента корреляции по модулю и это место является отсечкой во второй аварии, с которой она будет рассматриваться. На рисунке 4 представлена вкладка приложения, в которой вычисляется место отсечки во второй аварии.

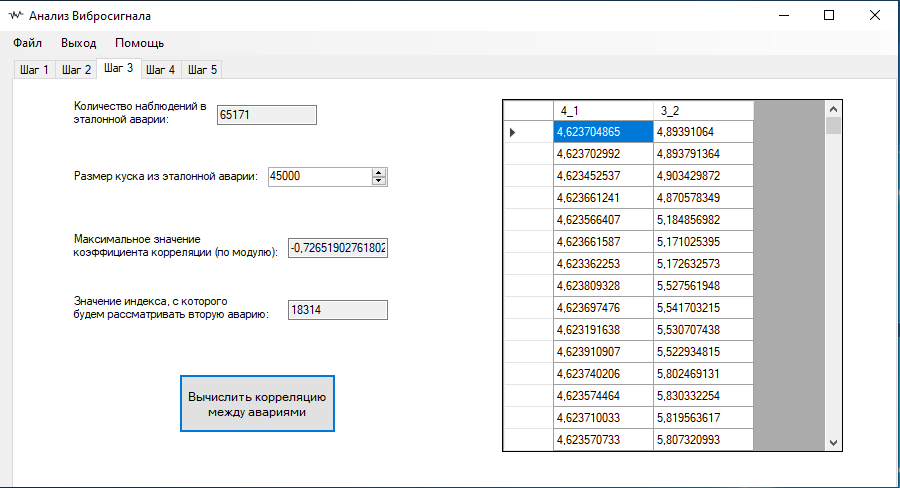


Рисунок –

– Нахождение отсечки во второй аварии

* 1. Нахождение полинома для выражения значений второй аварии через эталонную аварию

Берется достаточно большой интервал наблюдений из эталонной аварии и из второй аварии с места найденной отсечки, данный интервал задается программно.

Далее с помощью полиномов формируются уравнения для определения второй аварии через эталонную. Максимальная допустимая степень полиномов задается программно. Коэффициенты полиномов находятся с помощью метода наименьших квадратов (3)

(3)

где Z – матрица, первый столбец которой состоит из «1»; Y – вектор, содержащий наблюдения из второй аварии.

Для определения наилучшей степени полинома используется скорректированный коэффициент детерминации (4)

(4)

где n – количество наблюдений в наборе данных, k – количество параметров модели,

– сумма квадратов остатков регрессии,

– общая дисперсия.

На рисунке 5 представлена вкладка приложения, в которой происходит вычисление наилучшей степени полинома.

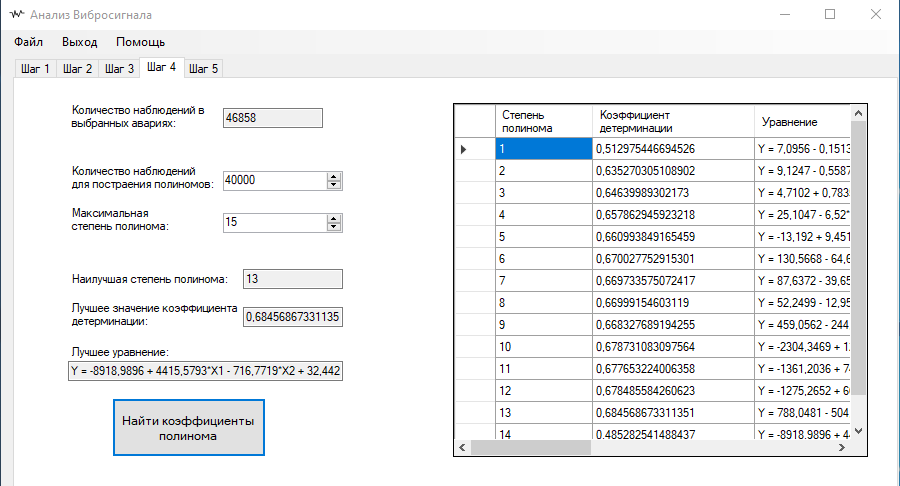


Рисунок –

– Нахождение наилучшего полинома

* 1. Сравнение коэффициентов надежности у реальных и предсказанных значений вибросигнала

С помощью наилучшего полинома, найденного на предыдущем шаге, нужно преобразовать последние n значений эталонной аварии в значения не эталонной аварии, значение n – задается программно. Для полученных значений расставляются коэффициенты надежности. Далее новые коэффициенты надежности сравниваются с реальными значениями надежности для второй аварии. На рисунке 6 представлена вкладка приложения, в которой сравниваются коэффициенты надежности для реальных и предсказанных значений второй аварии.

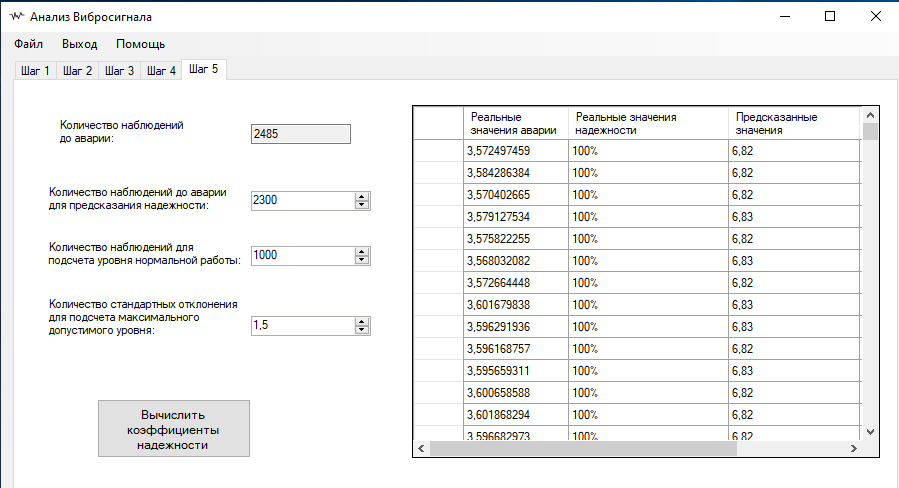


Рисунок –

– Сравнение коэффициентов надежности

1. Исследование по нахождению наилучшей степени полинома

Для исследования будут использованы данные о шести различных стендовых авариях подшипников.

Для нахождения наилучшей степени полинома будет исследована каждая пара аварий, то есть каждая из аварий будет эталонной, и каждая эталонная авария будет сравниваться со всеми другими авариями. В таблице 1 представлены результаты исследования. Среди лучших степеней полинома чаще всего встречается 11.

Таблица

– Результаты исследования по нахождению наилучшей степени полинома

|  |  |  |  |
| --- | --- | --- | --- |
| № Эталонной аварии | № Второй аварии | Лучшая степень полинома | Лучший коэффициент детерминации |
| 1 | 2 | 14 | 0.9359 |
| 3 | 14 | 0.4371 |
| 4 | 9 | 0.3538 |
| 5 | 14 | 0.1429 |
| 6 | 8 | 0.0574 |
| 2 | 1 | 11 | 0.8676 |
| 3 | 12 | 0.4289 |
| 4 | 12 | 0.2077 |
| 5 | 12 | 0.1974 |
| 6 | 11 | 0.0491 |
| 3 | 1 | 14 | 0.5652 |
| 2 | 14 | 0.6964 |
| 4 | 9 | 0.9093 |
| 5 | 8 | 0.1351 |
| 6 | 11 | 0.0579 |
| 4 | 1 | 11 | 0.1157 |
| 2 | 7 | 0.1605 |
| 3 | 7 | 0.5771 |
| 5 | 7 | 0.9797 |
| 6 | 9 | 0.0168 |
| 5 | 1 | 7 | 0.1375 |
| 2 | 12 | 0.2302 |
| 3 | 7 | 0.6605 |
| 4 | 7 | 0.9844 |
| 6 | 6 | 0.0249 |
| 6 | 1 | 11 | 0.0763 |
| 2 | 11 | 0.0511 |
| 3 | 11 | 0.0682 |
| 4 | 10 | 0.0024 |
| 5 | 10 | 0.0039 |

Заключение

В ходе выполнения научно-исследовательской работы было сделано следующее:

* изучена предметная область;
* изучены необходимые статистические методы и алгоритмы;
* собраны данные с СКЗ виброскорости;
* спроектировано настольное приложение на языке C# с использованием технологии Windows Forms;
* реализовано настольное приложение;
* проведены эксперименты для выявления наилучшей степени полинома.

Результатом данной работы является настольное приложение, реализованное на языке программирования C# с использованием технологии Windows Forms, которое позволяет работать с данными, содержащими СКЗ вибросигнала. Данное приложение позволяет загружать данные; находить наилучший полином, описывающий одну аварию через вторую, для предсказания значения надежности подшипника; вычислять надежность подшипника, исходя из значения сигнала.

По результатам проведенного исследования – наилучшей степенью полинома является 11 степень.

Цель данной работы была достигнута.

Список литературы

1. Report of Large Motor Reliability Survey of Industrial and Commercial Installations Part I, IEEE Transactions on Industry Applications, vol. IA-21, no. 5, pp. 853-864, July 1985.
2. Bearings Fault Detection Using Inference Tools. Miguel Delgado Prieto, Jordi Cusido i Roura, Jose Lius Romeral Martinez. 2011.
3. «Энциклопедия Кругосвет». Универсальная научно-популярная онлайн-энциклопедия. URL: <https://www.krugosvet.ru/enc/nauka_i_tehnika/tehnologiya_i_promyshlennost/PODSHIPNIK.html> (дата обращения 24.04.2023).
4. Виброускорение, виброскорость и виброперемещение // Вибро-Центр [Электронный ресурс] // URL: <https://clck.ru/34FstU> (дата обращения 24.04.2023)
5. Что такое СКЗ? // Вибро-Центр [Электронный ресурс] // URL: <https://clck.ru/34FtYZ> (дата обращения 24.04.2023).
6. Nayana B. R., Geethanjali P. Analysis of Statistical Time-Domain Features Effectiveness in Identification of Bearing Faults From Vibration Signal. IEEE. July 2017, vol. 17, Issue: 17, pp. 5618-5625. <https://doi.org/10.1109/JSEN.2017.2727638>
7. Ewert P., Orlowska-Kowalska T., Jankowska K. Effectiveness Analysis of PMSM Motor Rolling Bearing Fault Detectors Based on Vibration Analysis and Shallow Neural Networks. Energies. 2021; 14(3):712. <https://doi.org/10.3390/en14030712>
8. Zarei J., Tajeddini M. A., Karimi H. R. Vibration analysis for bearing fault detection and classification using an intelligent filter. Mechatronics. March 2014, vol. 24, pp. 151-157. <https://doi.org/10.1016/j.mechatronics.2014.01.003>
9. Maad M. Mijwel. Artificial Neural Networks Advantages and Disadvantages. January 2018.
10. 4 Disadvantages of Neural Networks and Deep Learning [Электронный ресурс] // URL: <https://builtin.com/data-science/disadvantages-neural-networks> (дата обращения 24.04.2023).
11. Xiaohang Jin, Mingbo Zhao, Tommy W. S. Chow, Michael Pecht. Motor Bearing Fault Diagnosis Using Trace Ratio Linear Discriminant Analysis. IEEE. August 2013, vol. 61, Issue: 5, pp. 2441-2451. <https://doi.org/10.1109/TIE.2013.2273471>
12. Ocak H., Loparo K. A. A new bearing fault detection and diagnosis scheme based on hidden Markov modeling of vibration signals. IEEE. August 2002. <https://doi.org/10.1109/ICASSP.2001.940324>

Приложение А

Листинг программы:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace Vibration\_Analisys2 {

public partial class MainForm : Form {

// Excel connector

Microsoft.Office.Interop.Excel.Application xlapp = new Microsoft.Office.Interop.Excel.Application();

/// <summary>

/// List for reference fault values

/// </summary>

public List<double> ReferenceFault { get; set; } = new List<double>();

/// <summary>

/// List for second fault values

/// </summary>

public List<double> SecondFault { get; set; } = new List<double>();

/// <summary>

/// List for selected inverval reference fault values

/// </summary>

public List<double> SelectIntervalRefFault { get; set; } = new List<double>();

/// <summary>

/// List for selected inverval second fault values

/// </summary>

public List<double> SelectIntervalSecFault { get; set; } = new List<double>();

/// <summary>

/// Values before fault in reference fault

/// </summary>

public List<double> BeforeRefFault { get; set; } = new List<double>();

/// <summary>

/// Values before fault in second fault

/// </summary>

public List<double> BeforeSecFault { get; set; } = new List<double>();

/// <summary>

/// Header for reference fault

/// </summary>

public (string, int) ReferenceFaultHeader { get; set; }

/// <summary>

/// Header for second fault

/// </summary>

public (string, int) SecondFaultHeader { get; set; }

/// <summary>

/// Vector of best coefficients values

/// </summary>

public List<double> BestCoeffs { get; set; } = new List<double>();

/// <summary>

/// Filename of excel file with fault data

/// </summary>

public string FilenameExcel { get; set; }

/// <summary>

/// Max value of signal for normal work for second fault

/// </summary>

public double MaxNormalVibraitonSignalLevel { get; set; }

// BackgroundWorkers for each step

BackgroundWorker workerStep1 = new BackgroundWorker();

BackgroundWorker workerStep2 = new BackgroundWorker();

BackgroundWorker workerStep3 = new BackgroundWorker();

BackgroundWorker workerStep4 = new BackgroundWorker();

BackgroundWorker workerStep5 = new BackgroundWorker();

public MainForm() {

InitializeComponent();

// Centered Main From on the screen

this.CenterToScreen();

// Locks all tabs except the first one

foreach (TabPage tab in allSteps.TabPages){

tab.Enabled = false;

}

step1.Enabled = true;

maxPolynomDegree.Maximum = Decimal.MaxValue;

numberOfStdForMaxLevel.Maximum = Decimal.MaxValue;

numberOfStdInPredicted.Maximum = Decimal.MaxValue;

helpAllSteps.ToolTipText = StepsInfo.Step1;

}

/// <summary>

/// Open Excel File with Vibration Data

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void openExcelFile\_Click(object sender, EventArgs e) {

ClearControlsStep1();

allSteps.SelectTab(step1);

// Open xlsx file dialog

using (OpenFileDialog ofd = new OpenFileDialog()) {

ofd.Filter = "Excel Files Only | \*.xlsx; \*.xls";

ofd.Title = "Choose the file";

if (ofd.ShowDialog() == DialogResult.OK) {

FilenameExcel = ofd.FileName;

// Run background worker for load data from excel file

workerStep1.ProgressChanged += new ProgressChangedEventHandler(ProgressExcelLoadChanged);

workerStep1.DoWork += new DoWorkEventHandler(LoadData);

workerStep1.WorkerReportsProgress = true;

dataGV.Size = new Size(682, 370);

progressBarDataLoad.Value = 0;

progressBarDataLoad.Visible = true;

workerStep1.RunWorkerAsync();

}

}

}

/// <summary>

/// Function for clear controls on step1

/// </summary>

private void ClearControlsStep1() {

ClearDataGVHeaders(dataGV);

referenceFaultBox.Items.Clear();

secondFaultBox.Items.Clear();

ClearControlsStep2();

}

/// <summary>

/// Function for clear controls on step2

/// </summary>

private void ClearControlsStep2() {

numberOfValuesForNormalWorkLevel.Value = 1;

numberOfStdForMaxLevel.Value = 1;

allValuesInFaults.Clear();

meanValueForNormalWork.Clear();

stdValueForNormalWork.Clear();

faultSignal.Clear();

maxVibrationSignal.Clear();

ClearDataGVHeaders(dataSignalReliability);

ClearControlsStep3();

}

/// <summary>

/// Function for clear controls on step3

/// </summary>

private void ClearControlsStep3() {

numericPieceOfRefFault.Value = 1;

bestCorrelCoefTextBox.Clear();

bestIndexSecFaultTextBox.Clear();

numValuesInRefFault.Clear();

ClearDataGVHeaders(dataGVbestIntervalsOfFault);

ClearControlsStep4();

}

/// <summary>

/// Function for clear controls on step4

/// </summary>

private void ClearControlsStep4() {

numberOfValuesInSelectedInterval.Clear();

numberOfValuesForPolynomes.Value = 1;

maxPolynomDegree.Value = 15;

ClearDataGVHeaders(dataGVBestPoly);

bestPolyDegreeValue.Clear();

bestDetermCoeffValue.Clear();

bestEquation.Clear();

ClearControlsStep5();

}

/// <summary>

/// Function for clear controls on step5

/// </summary>

private void ClearControlsStep5() {

ClearDataGVHeaders(dataGVPredReliability);

numberOfValuesBeforeFault.Clear();

valuesBeforeFault.Value = 1;

numberOfValuesForNormalWorkPredict.Value = 1;

numberOfStdInPredicted.Value = (decimal)1.0;

}

/// <summary>

/// Clear headers from dataGridView

/// </summary>

/// <param name="data">dataGridView</param>

private void ClearDataGVHeaders(DataGridView data) {

data.Rows.Clear();

data.ColumnHeadersVisible = false;

data.Refresh();

}

/// <summary>

/// Function for load data from excel file to DataGridView

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void LoadData(object sender, DoWorkEventArgs e) {

Microsoft.Office.Interop.Excel.Workbook workbook = xlapp.Workbooks.Open(FilenameExcel);

Microsoft.Office.Interop.Excel.Worksheet sheet;

Microsoft.Office.Interop.Excel.Range range;

try {

sheet = (Microsoft.Office.Interop.Excel.Worksheet)workbook.ActiveSheet;

range = sheet.UsedRange;

dataGV.Invoke(new Action<int>((n) => dataGV.ColumnCount = n), range.Columns.Count);

int progress = 0;

int step = range.Rows.Count / 100;

int oneBarInProgress = 1;

if (range.Rows.Count < 100) {

step = 1;

oneBarInProgress = (100 / range.Rows.Count) + 1;

}

workerStep1.ReportProgress(progress);

for (int row = 1; row <= range.Rows.Count; row++) {

// Find progress

if (row % step == 0) {

progress += oneBarInProgress;

workerStep1.ReportProgress(progress);

}

// Add values from excel sheet to dataGridView

List<string> rowValues = new List<string>();

for (int col = 1; col <= range.Columns.Count; col++) {

rowValues.Add(range.Cells[row, col].Text);

}

if (row == 1) {

dataGV.Invoke(new Action<List<string>>((s) => ExcelDataSetHeaders(s)), rowValues);

continue;

}

dataGV.Invoke(new Action<List<string>>((s) => AddRowFunc(s)), rowValues);

}

workbook.Close(true);

}

catch (Exception ex) {

MessageBox.Show(ex.Message);

workbook.Close(0);

xlapp.Quit();

return;

}

// Create a header lists

List<string> headerList = new List<string>();

for (int i = 1; i < dataGV.ColumnCount; i++) {

headerList.Add(dataGV.Columns[i].HeaderText);

}

referenceFaultBox.Invoke(new Action<List<string>>((s) => AddHeadersInReferenceFaultBox(s)), headerList);

secondFaultBox.Invoke(new Action<List<string>>((s) => AddHeadersInSecondFaultBox(s)), headerList);

referenceFaultBox.Invoke(new Action<int>((n) => referenceFaultBox.SelectedIndex = n), 0);

secondFaultBox.Invoke(new Action<int>((n) => secondFaultBox.SelectedIndex = n), 1);

progressBarDataLoad.Invoke(new Action<bool>((b) => progressBarDataLoad.Visible = b), false);

dataGV.Invoke(new Action<Size>((size) => dataGV.Size = size), new Size(682, 395));

workerStep1.DoWork -= new DoWorkEventHandler(LoadData);

}

/// <summary>

/// Set headers for main excel dataGV

/// </summary>

/// <param name="headers">List of headers</param>

private void ExcelDataSetHeaders(List<string> headers) {

SetDataGVColumnHeaders(headers, dataGV, false);

}

/// <summary>

/// Change progress bar value

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ProgressExcelLoadChanged(object sender, ProgressChangedEventArgs e) {

if (e.ProgressPercentage > 100) {

progressBarDataLoad.Value = 100;

}

else {

progressBarDataLoad.Value = e.ProgressPercentage;

}

}

/// <summary>

/// Add values to reference fault comboBox

/// </summary>

/// <param name="headers">List of headers</param>

private void AddHeadersInReferenceFaultBox(List<string> headers) {

referenceFaultBox.Items.AddRange(headers.ToArray());

}

/// <summary>

/// Add values to second fault comboBox

/// </summary>

/// <param name="headers">List of headers</param>

private void AddHeadersInSecondFaultBox(List<string> headers) {

secondFaultBox.Items.AddRange(headers.ToArray());

}

/// <summary>

/// Add row to dataGridView

/// </summary>

/// <param name="nums">List of values</param>

private void AddRowFunc(List<string> nums) {

dataGV.Rows.Add(nums.ToArray());

}

/// <summary>

/// Select reference fault and second fault

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void acceptFaultsButton\_Click(object sender, EventArgs e) {

ReferenceFaultHeader = (referenceFaultBox.Text, referenceFaultBox.SelectedIndex + 1);

SecondFaultHeader = (secondFaultBox.Text, secondFaultBox.SelectedIndex + 1);

ReferenceFault.Clear();

SecondFault.Clear();

try {

// Create second and reference fault lists

for (int rowNumber = 1; rowNumber < dataGV.Rows.Count; rowNumber++) {

try {

ReferenceFault.Add(Convert.ToDouble(dataGV[ReferenceFaultHeader.Item2, rowNumber].Value));

SecondFault.Add(Convert.ToDouble(dataGV[SecondFaultHeader.Item2, rowNumber].Value));

}

catch { }

}

ClearControlsStep2();

step2.Enabled = true;

numberOfValuesForNormalWorkLevel.Maximum = SecondFault.Count;

allValuesInFaults.Text = Math.Min(ReferenceFault.Count, SecondFault.Count).ToString();

allSteps.SelectTab(step2);

}

catch (Exception ex) {

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Check reference fault box rules

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void referenceFaultBox\_SelectedIndexChanged(object sender, EventArgs e) {

CheckAcceptButtonRule();

}

/// <summary>

/// Check second fault box rules

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void secondFaultBox\_SelectedIndexChanged(object sender, EventArgs e) {

CheckAcceptButtonRule();

}

/// <summary>

/// Check rules for accepting fault button

/// </summary>

private void CheckAcceptButtonRule() {

if (IsFaultBoxesAcceptable()) {

acceptFaultsButton.Enabled = true;

}

else {

acceptFaultsButton.Enabled = false;

}

}

/// <summary>

/// Rule for fault box acceptable

/// </summary>

/// <returns></returns>

private bool IsFaultBoxesAcceptable() {

return (referenceFaultBox.Text != secondFaultBox.Text) &&

(referenceFaultBox.Text != "") && (secondFaultBox.Text != "");

}

/// <summary>

/// Exit application

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ExitButton\_Click(object sender, EventArgs e) {

var exitForm = new ExitForm();

exitForm.Show();

}

/// <summary>

/// For validate value in Text box (only int numbers)

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void numberOfValuesForNormalWorkLevel\_KeyPress(object sender, KeyPressEventArgs e) {

e.Handled = CheckNumericIntValue(e);

}

/// <summary>

/// Check if pressed numeric or backspace

/// </summary>

/// <param name="e"></param>

/// <returns></returns>

private bool CheckNumericIntValue(KeyPressEventArgs e) {

return (e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar != 8;

}

/// <summary>

/// Calculation reliability for second fault signal

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void calcReliabilitySignal\_Click(object sender, EventArgs e) {

// Очищаем dataGridView

dataSignalReliability.Rows.Clear();

dataSignalReliability.Refresh();

int intervalSize = ((int)numberOfValuesForNormalWorkLevel.Value);

double meanValueForInterval = SecondFault.GetRange(0, intervalSize).Average();

double stdValueForInterval = StandardDeviation(SecondFault.GetRange(0, intervalSize));

MaxNormalVibraitonSignalLevel = GetMaxNormalVibrLevel(SecondFault.GetRange(0, intervalSize), (double)numberOfStdForMaxLevel.Value);

meanValueForNormalWork.Text = meanValueForInterval.ToString();

stdValueForNormalWork.Text = stdValueForInterval.ToString();

faultSignal.Text = SecondFault.Max().ToString();

maxVibrationSignal.Text = MaxNormalVibraitonSignalLevel.ToString();

getReliabilityForSecondSignal();

ClearControlsStep3();

numericPieceOfRefFault.Maximum = ReferenceFault.Count();

numValuesInRefFault.Text = ReferenceFault.Count.ToString();

step3.Enabled = true;

}

/// <summary>

/// Find Max normal vibration level for vibration signal

/// </summary>

/// <param name="signal">Vibration signal</param>

/// <param name="stdCount">Count of stds</param>

/// <returns>Max normal vibration level</returns>

private double GetMaxNormalVibrLevel(List<double> signal, double stdCount) {

double stdValue = StandardDeviation(signal);

double meanValue = signal.Average();

return meanValue + stdCount \* stdValue;

}

/// <summary>

/// Calculation reliability with backgroung workerStep1

/// </summary>

private void getReliabilityForSecondSignal() {

dataSignalReliability.ColumnCount = 2;

SetDataGVColumnHeaders(new List<string>() { SecondFaultHeader.Item1, "Надежность" }, dataSignalReliability, false);

// Run background worker for load values and reliability of choosen faults into dataGridView

workerStep2.ProgressChanged += new ProgressChangedEventHandler(ProgressReliabilityChanged);

workerStep2.DoWork += new DoWorkEventHandler(getReliability);

workerStep2.WorkerReportsProgress = true;

dataSignalReliability.Size = new Size(341, 329);

progressBarReliability.Value = 0;

progressBarReliability.Visible = true;

workerStep2.RunWorkerAsync();

}

/// <summary>

/// Background workerStep1 for adding rows to DataGridView

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void getReliability(object sender, DoWorkEventArgs e) {

double maxSignalLevel = SecondFault.Max();

int numberOfDivisions = 0;

double oneDivision = (maxSignalLevel - MaxNormalVibraitonSignalLevel) / 99;

int progress = 0;

int step = SecondFault.Count / 100;

int oneBarInProgress = 1;

if (SecondFault.Count < 100) {

step = 1;

oneBarInProgress = (100 / SecondFault.Count) + 1;

}

workerStep2.ReportProgress(progress);

double prevBiggestSignal = MaxNormalVibraitonSignalLevel;

for (int i = 0; i < SecondFault.Count; i++) {

// Find progress

if (i % step == 0) {

progress += oneBarInProgress;

workerStep2.ReportProgress(progress);

}

if (SecondFault[i] > prevBiggestSignal) {

prevBiggestSignal = SecondFault[i];

numberOfDivisions = (int)((SecondFault[i] - (MaxNormalVibraitonSignalLevel)) / oneDivision) + 1;

}

dataSignalReliability.Invoke(new Action<(double, int)>((s) => AddValuePercent(s)), (SecondFault[i], numberOfDivisions));

}

progressBarReliability.Invoke(new Action<bool>((b) => progressBarReliability.Visible = b), false);

dataSignalReliability.Invoke(new Action<Size>((size) => dataSignalReliability.Size = size), new Size(341, 353));

workerStep2.DoWork -= new DoWorkEventHandler(getReliability);

}

/// <summary>

/// Add row to dataGridView with percent of reliability

/// </summary>

/// <param name="values">value and percent of reliability</param>

private void AddValuePercent((double, int) values) {

int reliability = 100 - values.Item2;

dataSignalReliability.Rows.Add(values.Item1, reliability.ToString() + "%");

}

/// <summary>

/// Change progress bar value for reliability bar

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ProgressReliabilityChanged(object sender, ProgressChangedEventArgs e) {

if (e.ProgressPercentage > 100) {

progressBarReliability.Value = 100;

}

else {

progressBarReliability.Value = e.ProgressPercentage;

}

}

/// <summary>

/// Get standard deviation

/// </summary>

/// <param name="values">List of values</param>

/// <returns>Value of standard deviation</returns>

private double StandardDeviation(IEnumerable<double> values) {

double avg = values.Average();

return Math.Sqrt(values.Average(v => Math.Pow(v - avg, 2)));

}

/// <summary>

/// Find the most correlated sections of accidents

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void maxPearsonCoefTwoFaultsButton\_Click(object sender, EventArgs e) {

// Очистить dataGridView

dataGVbestIntervalsOfFault.Rows.Clear();

dataGVbestIntervalsOfFault.Refresh();

findBestIntervalBar.Value = 0;

findBestIntervalBar.Visible = true;

int numberOfValuesInFault = (int)numericPieceOfRefFault.Value;

SelectIntervalRefFault = new List<double>(ReferenceFault.GetRange(0, numberOfValuesInFault));

double bestCorrCoef = 0;

int bestStartIndexSecFault = 0;

int progress = 0;

int step = (SecondFault.Count - numberOfValuesInFault) / 100;

int oneBarInProgress = 1;

if ((SecondFault.Count - numberOfValuesInFault) < 100) {

step = 1;

if (SecondFault.Count == numberOfValuesInFault) {

oneBarInProgress = 100;

}

else {

oneBarInProgress = (100 / (SecondFault.Count - numberOfValuesInFault)) + 1;

}

}

for (int i = 0; i < (SecondFault.Count - numberOfValuesInFault); i++) {

// Find progress

if (i % step == 0) {

progress += oneBarInProgress;

if (progress > 100) {

progress = 100;

}

findBestIntervalBar.Value = progress;

}

double corrCoef = CorrelCoef(SelectIntervalRefFault, SecondFault.GetRange(i, numberOfValuesInFault));

if (Math.Abs(corrCoef) > Math.Abs(bestCorrCoef)) {

bestCorrCoef = corrCoef;

bestStartIndexSecFault = i;

}

}

findBestIntervalBar.Visible = false;

bestCorrelCoefTextBox.Text = bestCorrCoef.ToString();

bestIndexSecFaultTextBox.Text = (bestStartIndexSecFault + 1).ToString();

SelectIntervalRefFault = new List<double>(ReferenceFault.GetRange(0, SecondFault.Count - bestStartIndexSecFault));

SelectIntervalSecFault = new List<double>(SecondFault.GetRange(bestStartIndexSecFault, SecondFault.Count - bestStartIndexSecFault));

WriteBestIntervalsIntoDataGridView();

ClearControlsStep4();

step4.Enabled = true;

numberOfValuesInSelectedInterval.Text = SelectIntervalSecFault.Count.ToString();

numberOfValuesForPolynomes.Maximum = SelectIntervalSecFault.Count;

maxPolynomDegree.Value = 15;

numberOfValuesForPolynomes.Value = SelectIntervalSecFault.Count;

}

/// <summary>

/// Get correlation coefficient (Pearson coefficient)

/// </summary>

/// <param name="list1">First list of values</param>

/// <param name="list2">Second list of values</param>

/// <param name="app">Excel application</param>

/// <returns>Value of Correlation coefficient</returns>

private double CorrelCoef(IEnumerable<double> list1, IEnumerable<double> list2) {

return xlapp.WorksheetFunction.Correl(list1.ToArray(), list2.ToArray());

}

/// <summary>

/// Function for writing best intervals into data grid view with background worker

/// </summary>

private void WriteBestIntervalsIntoDataGridView() {

dataGVbestIntervalsOfFault.ColumnCount = 2;

SetDataGVColumnHeaders(new List<string>() { ReferenceFaultHeader.Item1, SecondFaultHeader.Item1 }, dataGVbestIntervalsOfFault, false);

// Run background worker for load best intervals of reference and second fault

workerStep3.ProgressChanged += new ProgressChangedEventHandler(ProgressSelectIntervalChanged);

workerStep3.DoWork += new DoWorkEventHandler(WriteBestIntervalsToDataGridAsync);

workerStep3.WorkerReportsProgress = true;

dataGVbestIntervalsOfFault.Size = new Size(341, 329);

progressBarSelectedInterval.Value = 0;

progressBarSelectedInterval.Visible = true;

workerStep3.RunWorkerAsync();

}

/// <summary>

/// Change progress bar value for select interval bar

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ProgressSelectIntervalChanged(object sender, ProgressChangedEventArgs e) {

if (e.ProgressPercentage > 100) {

progressBarSelectedInterval.Value = 100;

}

else {

progressBarSelectedInterval.Value = e.ProgressPercentage;

}

}

/// <summary>

/// Write value of selected intervals to dataGridView

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void WriteBestIntervalsToDataGridAsync(object sender, DoWorkEventArgs e) {

int progress = 0;

int step = SelectIntervalRefFault.Count / 100;

int oneBarInProgress = 1;

if (SelectIntervalRefFault.Count < 100) {

step = 1;

oneBarInProgress = (100 / SelectIntervalRefFault.Count) + 1;

}

workerStep3.ReportProgress(progress);

for (int i = 0; i < SelectIntervalRefFault.Count; i++) {

// Find progress

if (i % step == 0) {

progress += oneBarInProgress;

workerStep3.ReportProgress(progress);

}

(double, double) newRow = (SelectIntervalRefFault[i], SelectIntervalSecFault[i]);

dataGVbestIntervalsOfFault.Invoke(new Action<(double, double)>((values) => WriteRowOfBestInterval(values)), newRow);

}

progressBarSelectedInterval.Invoke(new Action<bool>((b) => progressBarSelectedInterval.Visible = b), false);

dataGVbestIntervalsOfFault.Invoke(new Action<Size>((size) => dataGVbestIntervalsOfFault.Size = size), new Size(341, 353));

workerStep3.DoWork -= new DoWorkEventHandler(WriteBestIntervalsToDataGridAsync);

workerStep3.Dispose();

}

/// <summary>

/// Write new row of best intervals to dataDV

/// </summary>

/// <param name="values">Tuple of two elements: reference fault and second fault</param>

private void WriteRowOfBestInterval((double, double) values) {

dataGVbestIntervalsOfFault.Rows.Add(values.Item1, values.Item2);

}

/// <summary>

/// Find best polynom

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void FindPolynomButton\_Click(object sender, EventArgs e) {

// Очистить dataGridView

dataGVBestPoly.Rows.Clear();

dataGVBestPoly.Refresh();

dataGVBestPoly.ColumnCount = 3;

SetDataGVColumnHeaders(new List<string>() { "Степень полинома", "Коэффициент детерминации", "Уравнение" }, dataGVBestPoly, true);

dataGVBestPoly.Columns[1].SortMode = DataGridViewColumnSortMode.Automatic;

// Run background worker for finding polynom coeffs

workerStep4.ProgressChanged += new ProgressChangedEventHandler(ProgressFindBestPolyChanged);

workerStep4.DoWork += new DoWorkEventHandler(FindBestPolynom);

workerStep4.WorkerReportsProgress = true;

dataGVBestPoly.Size = new Size(415, 329);

progressBestPoly.Value = 0;

progressBestPoly.Visible = true;

workerStep4.RunWorkerAsync();

ClearControlsStep5();

step5.Enabled = true;

int valuesCountBeforeFault = Math.Min(ReferenceFault.IndexOf(ReferenceFault.Max()), SecondFault.IndexOf(SecondFault.Max()));

numberOfValuesBeforeFault.Text = valuesCountBeforeFault.ToString();

valuesBeforeFault.Maximum = valuesCountBeforeFault;

valuesBeforeFault.Value = valuesCountBeforeFault;

numberOfValuesForNormalWorkPredict.Maximum = valuesCountBeforeFault;

}

/// <summary>

/// Find best polynom in background

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void FindBestPolynom(object sender, DoWorkEventArgs e) {

int numberValuesForPolynom = (int)numberOfValuesForPolynomes.Value;

int maxDegree = (int)maxPolynomDegree.Value;

int bestPolynomialDegree = 1;

double bestDetermCoef = 0;

// New selected intervals of faults

SelectIntervalRefFault = new List<double>(SelectIntervalRefFault.GetRange(0, numberValuesForPolynom));

SelectIntervalSecFault = new List<double>(SelectIntervalSecFault.GetRange(0, numberValuesForPolynom));

List<double> Y = new List<double>(SelectIntervalSecFault);

List<List<double>> Z = new List<List<double>>();

List<List<double>> coeffVectors = new List<List<double>>();

Z.Add(OnesList(numberValuesForPolynom));

// Find one bar value in progress bar

int progress = 0;

int step = maxDegree / 100;

int oneBarInProgress = 1;

if (maxDegree < 100) {

step = 1;

oneBarInProgress = (100 / maxDegree) + 1;

}

workerStep4.ReportProgress(progress);

for (int i = 1; i <= maxDegree; i++) {

// Find progress

if (i % step == 0) {

progress += oneBarInProgress;

workerStep4.ReportProgress(progress);

}

Z.Add(PowList(SelectIntervalRefFault, i));

// Find vector of coefficients

List<double> coeffVector = FindVectorOfCoeffs(Z, Y);

// Adjusted coefficient of determination

double determCoeff = GetDetermCoeff(Z, coeffVector, Y);

if (determCoeff > bestDetermCoef) {

BestCoeffs = new List<double>(coeffVector);

bestDetermCoef = determCoeff;

bestPolynomialDegree = i;

}

coeffVectors.Add(coeffVector);

string newEquation = GetEquation(coeffVector);

dataGVBestPoly.Invoke(new Action<(int, double, string)>((data) => WriteRowOfPolynom(data)), (i, determCoeff, newEquation));

}

bestPolyDegreeValue.Invoke(new Action<string>((s) => bestPolyDegreeValue.Text = s), bestPolynomialDegree.ToString());

bestDetermCoeffValue.Invoke(new Action<string>((s) => bestDetermCoeffValue.Text = s), bestDetermCoef.ToString());

bestEquation.Invoke(new Action<string>((s) => bestEquation.Text = s), dataGVBestPoly[2, bestPolynomialDegree - 1].Value);

progressBestPoly.Invoke(new Action<bool>((b) => progressBestPoly.Visible = b), false);

dataGVBestPoly.Invoke(new Action<Size>((size) => dataGVBestPoly.Size = size), new Size(415, 354));

workerStep4.DoWork -= new DoWorkEventHandler(FindBestPolynom);

workerStep4.Dispose();

}

/// <summary>

/// Change progress bar value for select interval bar

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ProgressFindBestPolyChanged(object sender, ProgressChangedEventArgs e) {

if (e.ProgressPercentage > 100) {

progressBestPoly.Value = 100;

}

else {

progressBestPoly.Value = e.ProgressPercentage;

}

}

/// <summary>

/// Function that returns list of ones

/// </summary>

/// <param name="n">List size</param>

/// <returns>List of ones</returns>

private List<double> OnesList(int n) {

List<double> ones = new List<double>();

for (int i = 0; i < n; i++) {

ones.Add(1.0);

}

return ones;

}

/// <summary>

/// Function that pow list of numbers

/// </summary>

/// <param name="numList">List of numbers</param>

/// <param name="m">Pow value</param>

/// <returns>Pow list</returns>

private List<double> PowList(List<double> numList, int m) {

List<double> powList = new List<double>();

foreach (double elem in numList) {

powList.Add(Math.Pow(elem, m));

}

return powList;

}

/// <summary>

/// Find vector of coefficients

/// </summary>

/// <param name="Z">Z values</param>

/// <param name="Y">Vector of real Y-values</param>

/// <returns>Vector of equation coefficients</returns>

private List<double> FindVectorOfCoeffs(List<List<double>> Z, List<double> Y) {

List<List<double>> transposeZ = Transpose(Z);

return MultMatrixVector(MultTwoMatrix(InverseMatrix(MultTwoMatrix(transposeZ, Z)), transposeZ), Y);

}

/// <summary>

/// Transpose matrix

/// </summary>

/// <param name="matrix">Input matrix</param>

/// <returns>Transposed matrix</returns>

private List<List<double>> Transpose (List<List<double>> matrix) {

List<List<double>> transposeMatrix = new List<List<double>>();

for (int i = 0; i < matrix[0].Count; i++) {

List<double> nextRow = new List<double>();

for (int j = 0; j < matrix.Count; j++) {

nextRow.Add(matrix[j][i]);

}

transposeMatrix.Add(nextRow);

}

return transposeMatrix;

}

/// <summary>

/// Get matrix that represent mult of two matrices

/// </summary>

/// <param name="matrixA">First matrix</param>

/// <param name="matrixB">Second matrix</param>

/// <returns>Result mult two matrices</returns>

private List<List<double>> MultTwoMatrix(List<List<double>> matrixA, List<List<double>> matrixB) {

List<List<double>> resultMatrix = new List<List<double>>();

for (int colB = 0; colB < matrixB.Count; colB++) {

List<double> nextCol = new List<double>();

for (int rowA = 0; rowA < matrixA[0].Count; rowA++) {

double nextElem = 0;

for (int colA = 0; colA < matrixA.Count; colA++) {

nextElem += matrixA[colA][rowA] \* matrixB[colB][colA];

}

nextCol.Add(nextElem);

}

resultMatrix.Add(nextCol);

}

return resultMatrix;

}

/// <summary>

/// Inverse matrix

/// </summary>

/// <param name="matrix">Matrix</param>

/// <returns>Inversed matrix</returns>

private List<List<double>> InverseMatrix(List<List<double>> matrix) {

List<List<double>> additionalMatrix = new List<List<double>>(matrix);

int numCols = additionalMatrix.Count;

// Add ones matrix to input matrix

for (int i = 0; i < numCols; i++) {

List<double> nextCol = new List<double>();

for (int j = 0; j < numCols; j++) {

if (i == j) {

nextCol.Add(1.0);

continue;

}

nextCol.Add(0.0);

}

additionalMatrix.Add(nextCol);

}

// Gauss-Jordan Algorithm

for (int row = 0; row < numCols; row++) {

double diagElem = additionalMatrix[row][row];

// Divide row elements by diagElem

for (int col = row; col < numCols \* 2; col++) {

additionalMatrix[col][row] = additionalMatrix[col][row] / diagElem;

}

// Substracting permit row from other rows

for (int rowSub = 0; rowSub < numCols; rowSub++) {

if (rowSub != row) {

double otherDiagElem = additionalMatrix[row][rowSub];

for (int colSub = row; colSub < numCols \* 2; colSub++) {

additionalMatrix[colSub][rowSub] -= additionalMatrix[colSub][row] \* otherDiagElem;

}

}

}

}

return additionalMatrix.GetRange(numCols, numCols);

}

/// <summary>

/// Get vector that represent mult matrix by vector

/// </summary>

/// <param name="matrix">Matrix</param>

/// <param name="vector">Vector</param>

/// <returns>Result mult matrix by vector</returns>

private List<double> MultMatrixVector(List<List<double>> matrix, List<double> vector) {

List<double> resultVector = new List<double>();

for (int row = 0; row < matrix[0].Count; row++) {

double nextElem = 0;

for (int col = 0; col < matrix.Count; col++) {

nextElem += matrix[col][row] \* vector[col];

}

resultVector.Add(nextElem);

}

return resultVector;

}

/// <summary>

/// Get Adjusted coefficient of determination

/// </summary>

/// <param name="Z">Z-values</param>

/// <param name="coeffs">Vector of coefficients</param>

/// <param name="Y">Vector of real Y-values</param>

/// <returns>Adjusted coefficient of determination</returns>

private double GetDetermCoeff(List<List<double>> Z, List<double> coeffs, List<double> Y) {

int n = Y.Count;

int k = Z.Count - 1;

double rss = 0.0;

double tss = 0.0;

double meanY = Y.Average();

List<double> predictedY = GetPredicted(Z, coeffs);

for (int i = 0; i < Y.Count; i++) {

rss += Math.Pow((Y[i] - predictedY[i]), 2);

tss += Math.Pow((Y[i] - meanY), 2);

}

return 1 - ((rss / Convert.ToDouble(n - k)) / (tss / Convert.ToDouble(n - 1)));

}

/// <summary>

/// Get vector of predicted values

/// </summary>

/// <param name="Z">Z values</param>

/// <param name="coeffs">Сoefficients</param>

/// <returns>Vector of predicted values</returns>

private List<double> GetPredicted(List<List<double>> Z, List<double> coeffs) {

List<double> predicted = new List<double>();

for (int rows = 0; rows < Z[0].Count; rows++) {

double nextY = 0;

for (int cols = 0; cols < coeffs.Count; cols++) {

nextY += coeffs[cols] \* Z[cols][rows];

}

predicted.Add(nextY);

}

return predicted;

}

/// <summary>

/// Get equation of polynom

/// </summary>

/// <param name="coeffs"></param>

/// <returns>Equation in string</returns>

private string GetEquation(List<double> coeffs) {

string equation = "Y = " + Math.Round(coeffs[0], 4);

for (int i = 1; i < coeffs.Count; i++) {

if (coeffs[i] < 0) {

equation += " - " + Math.Abs(Math.Round(coeffs[i], 4)).ToString() + "\*X" + i.ToString();

continue;

}

equation += " + " + Math.Round(coeffs[i], 4).ToString() + "\*X" + i.ToString();

}

return equation;

}

/// <summary>

/// Write row to data that contains polynoms

/// </summary>

/// <param name="row">Row values</param>

private void WriteRowOfPolynom((int, double, string) row) {

dataGVBestPoly.Rows.Add(row.Item1, row.Item2, row.Item3);

}

/// <summary>

/// Set column headers and column settings to dataGV

/// </summary>

/// <param name="headers">List of column headers</param>

/// <param name="dataGV">DataGridView</param>

/// <param name="autoSize">AutoSize column width</param>

private void SetDataGVColumnHeaders(List<string> headers, DataGridView dataGV, bool autoSize) {

for (int i = 0; i < dataGV.Columns.Count; i++) {

dataGV.Columns[i].HeaderText = headers[i];

dataGV.Columns[i].SortMode = DataGridViewColumnSortMode.NotSortable;

if (autoSize) {

dataGV.Columns[i].AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;

}

}

dataGV.ColumnHeadersVisible = true;

}

/// <summary>

/// Close excel connector

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void MainForm\_FormClosing(object sender, FormClosingEventArgs e) {

xlapp.Quit();

}

/// <summary>

/// Get predicted reliability

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void GetPredictedReliability\_Click(object sender, EventArgs e) {

// Очистить dataGridView

dataGVPredReliability.Rows.Clear();

dataGVPredReliability.Refresh();

dataGVPredReliability.ColumnCount = 4;

SetDataGVColumnHeaders(new List<string>() { "Реальные значения аварии", "Реальные значения надежности", "Предсказанные значения", "Предсказанная надежность" }, dataGVPredReliability, true);

// Run background worker for calc predicted fault values

workerStep5.ProgressChanged += new ProgressChangedEventHandler(ProgressCalcPredValuesChanged);

workerStep5.DoWork += new DoWorkEventHandler(FindPredictedValuesReliability);

workerStep5.WorkerReportsProgress = true;

dataGVPredReliability.Size = new Size(468, 329);

predReliableProgress.Value = 0;

predReliableProgress.Visible = true;

workerStep5.RunWorkerAsync();

}

/// <summary>

/// Change progress bar value for select interval bar

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ProgressCalcPredValuesChanged(object sender, ProgressChangedEventArgs e) {

if (e.ProgressPercentage > 100) {

predReliableProgress.Value = 100;

}

else {

predReliableProgress.Value = e.ProgressPercentage;

}

}

/// <summary>

/// Background function for finding predicted values of reliability

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void FindPredictedValuesReliability(object sender, DoWorkEventArgs e) {

int countBeforeFault = (int)valuesBeforeFault.Value;

int intervalSize = (int)numberOfValuesForNormalWorkPredict.Value;

double numberOfStd = (double)numberOfStdInPredicted.Value;

// Create polynom for predict

List<double> valuesForPredict = new List<double>(ReferenceFault.GetRange(Convert.ToInt32(numberOfValuesBeforeFault.Text) - countBeforeFault, countBeforeFault + 1));

List<List<double>> predictPolynom = new List<List<double>>();

predictPolynom.Add(OnesList(countBeforeFault + 1));

for (int i = 1; i <= Convert.ToInt32(bestPolyDegreeValue.Text); i++) {

predictPolynom.Add(PowList(valuesForPredict, i));

}

// Get predicted and real second falut values

List<double> predictedSecond = new List<double>(GetPredicted(predictPolynom, BestCoeffs));

List<double> realSecond = new List<double>(SecondFault.GetRange(SecondFault.IndexOf(SecondFault.Max()) - countBeforeFault, countBeforeFault + 1));

// Get max normal level for predicted and real second fault

double predictMaxNormLevel = GetMaxNormalVibrLevel(predictedSecond.GetRange(0, intervalSize), numberOfStd);

double realMaxNormLevel = GetMaxNormalVibrLevel(realSecond.GetRange(0, intervalSize), numberOfStd);

// Get max levels for perdicted and real second fault

double predictMaxLevel = predictedSecond.Max();

double realMaxLevel = realSecond.Max();

// Get one divisions for predicted and real second fault

double predictOneDivision = (predictMaxLevel - predictMaxNormLevel) / 99;

double realOneDivision = (realMaxLevel - realMaxNormLevel) / 99;

// Number of divisions for values

int predictNumberOfDivisions = 0;

int realNumberOfDivisions = 0;

// Keep last biggest signal value

double predictPrevBiggestSignal = predictMaxNormLevel;

double realPrevBiggestSignal = realMaxNormLevel;

int progress = 0;

int step = predictedSecond.Count / 100;

int oneBarInProgress = 1;

if (predictedSecond.Count < 100) {

step = 1;

oneBarInProgress = (100 / predictedSecond.Count) + 1;

}

workerStep5.ReportProgress(progress);

for (int i = 0; i < predictedSecond.Count; i++) {

// Find progress

if (i % step == 0) {

progress += oneBarInProgress;

workerStep5.ReportProgress(progress);

}

if (predictedSecond[i] > predictPrevBiggestSignal) {

predictPrevBiggestSignal = predictedSecond[i];

predictNumberOfDivisions = (int)((predictedSecond[i] - (predictMaxNormLevel)) / predictOneDivision) + 1;

}

if (realSecond[i] > realPrevBiggestSignal) {

realPrevBiggestSignal = realSecond[i];

realNumberOfDivisions = (int)((realSecond[i] - (realMaxNormLevel)) / realOneDivision) + 1;

}

dataGVPredReliability.Invoke(new Action<double, int, double, int>((a, b, c, d) => AddRowToDataGVPredictReliability(a, b, c, d)),

realSecond[i], realNumberOfDivisions, predictedSecond[i], predictNumberOfDivisions);

}

predReliableProgress.Invoke(new Action<bool>((b) => predReliableProgress.Visible = b), false);

dataGVPredReliability.Invoke(new Action<Size>((size) => dataGVPredReliability.Size = size), new Size(468, 353));

workerStep5.DoWork -= new DoWorkEventHandler(FindPredictedValuesReliability);

}

private void numberOfValuesForNormalWorkPredict\_ValueChanged(object sender, EventArgs e) {

CheckStep5Rule();

}

private void valuesBeforeFault\_ValueChanged(object sender, EventArgs e) {

CheckStep5Rule();

}

/// <summary>

/// Check the rule

/// </summary>

private void CheckStep5Rule() {

if (Step5Rule()) {

GetPredictedReliability.Enabled = true;

}

else {

GetPredictedReliability.Enabled = false;

}

}

/// <summary>

/// Rule about number of values for predicted

/// </summary>

/// <returns></returns>

private bool Step5Rule() {

return numberOfValuesForNormalWorkPredict.Value <= valuesBeforeFault.Value;

}

/// <summary>

/// Add next row to dataGridView with predict reliability

/// </summary>

/// <param name="realValue">Real vibration signal value</param>

/// <param name="realRel">Real reliability value</param>

/// <param name="predictValue">Predict vibration signal value</param>

/// <param name="predictRel">Predict reliability value</param>

private void AddRowToDataGVPredictReliability(double realValue, int realRel, double predictValue, int predictRel) {

string realReliab = (100 - realRel).ToString() + "%";

string predictReliab = (100 - predictRel).ToString() + "%";

dataGVPredReliability.Rows.Add(realValue, realReliab, Math.Round(predictValue, 2), predictReliab);

}

/// <summary>

/// Change information text about step

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void allSteps\_Selected(object sender, TabControlEventArgs e) {

switch (allSteps.SelectedIndex) {

case 0:

helpAllSteps.ToolTipText = StepsInfo.Step1;

break;

case 1:

helpAllSteps.ToolTipText = StepsInfo.Step2;

break;

case 2:

helpAllSteps.ToolTipText = StepsInfo.Step3;

break;

case 3:

helpAllSteps.ToolTipText = StepsInfo.Step4;

break;

case 4:

helpAllSteps.ToolTipText = StepsInfo.Step5;

break;

}

}

}

}

Приложение Б

Excel-файл с СКЗ виброскорости по 6 авариями доступны на Яндекс.Диск (<https://disk.yandex.ru/d/4CeVTAInUksV_A>).