МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

«ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ»

НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

**Лабораторна робота №4**

**з курсу “Основи системного аналізу”**

**З теми: “** **Гарантоване функціонування фізичних моделей кіберфізичних систем в умовах багатофакторних ризиків”**

**Виконали:**

студенти 4 курсу групи КА-93

Теванян Роман

Соболь Марія

Шевченко Родіон

**Перевірила:**

Панкратова Н.Д.

**Київ – 2023**

**Мета**

* Розробка та реалізація алгоритму обчислення для діагностики Складної Технічної Системи (СТС), враховуючи наступні модулі:

- приведення інформації до формалізованого стандартного виду, що забезпечуює можливість формування функціональних залежностей (ФЗ);

- Відновлення ФЗ за емпіричними дискретно заданим вибірками;

- Встановлення зв’язку відновлених наближаючих функцій з функціями ризику;

- Виконання процедури відновлення ФЗ по заданим вибіркам, зміщених на значень;

- Побудова процедури прогнозування функціональних залежностей на p вибірок;

- Побудова процедури своєчасного виявлення нештатних ситуацій;

- Візуалізація процесу діагностики в процессі функціонування СТС у штатному та нештатному режимах;

- Відображення процесу діагностуваняя на інформаційному табло, графічне зображення динаміки процесу.

* Знайти і привести наявну літературу з даного питання, зробити огляд.

**Постанова задачі**

Розглядається система, що відповідає аварійній ситуації при русі реанімаційного автомобіля. Реаніммобіль має на меті підтримку життя пацієнта на борту за допомогою бортової електромережі.

Основне обладнання автомобілю:

1. ДВС1 - осноний двигун внутрішнього сгоряння (ДВС).
2. Г1 - Основний генератор потужності 1.1 кВат - обертання коленвалу 220 рад/с (при падінні нижче 200 рад/с вимикається, а при перевищенні 220 рад/с - вмикається).
3. КПП - коробка перемикання передач (передаточні числа: 1 - 4.05, 2 - 2.34, 3 - 1.39, 4 - 1, 5 - 0.85).
4. ДВС2 і Г2 - Вторинний двигун за потужністю 1.1 кВт (використовується у аварійних ситуаціях, витрачає 0.5 л/год.).
5. АБ - аккумуляторна батарея, що забезпечує живлення обладнання, коли генератори не виробляють електроенергію.
6. ПР - пристрій розподілу електроенергії, що забезпечує живлення від батареї, генераторів або від батареї і генераторів одночасно.
7. ПБ - бак для пального (на початковий момент 47 л).

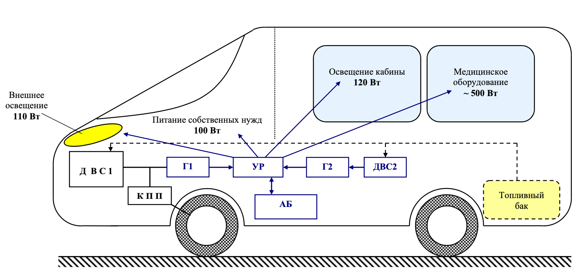
Напруга бортової мережі залежить від роботи генераторів Г1, Г2 та АБ.

При моделюванні розглядаються наступні функціональні елементи, що вимагають живлення:

1. Медичне обладнання 500 Вт
2. Освітлення основної кабіни = 120 Вт
3. Зовнішнє освітлення (фари) = 110 Вт
4. Власні потреби автомобіля = 100 Вт

Зарядний струм аккумулятора обмежений рівнем, що відповідає потужності, що витрачається генератором - 200 Вт.

ДВС1 та ДВС2 живляться одним паливним баком. В залежності від швидкості руху змінюється передача, а тому і рівень споживання пального.



**Штатна ситуація:** Автомобіль довозить пацієнта за 11 700 секунд (3.25 години) за напруги не менше від 11.35 В. На кінець шляху пального залишається 4.1 л.

**Перехід у позаштатний режим:** Позаштатна ситуація відбувається з тієї причини, датчик показує хибні значення заряду (акумулятор заряджений). Через це не відбувається заряджання акумулятора, і врешті він розряджається повністю, що і є причиною аварійної ситуації.

**Задача:** Реанімобіль моделі «ГАЗ» має проїхати 70 км при заданих дорожніх умовах при заданій швидкості у темний час доби.

**Припущення:** При зниженні бортової напруги нижче 11,7В, водій зупиняє машину (7323 с), вмикає резервний генератор (7414 с) та прибирає несправність (7863 с). Після цього водій заново починає рух, не вимикаючи резервний генератор. Електроенергії стає достатньо для живлення всіх елементів, та автомобіль добирається до місця призначення (13120 с), а в його баливному баці залишається 3,1 литри палива.

У кожному файлі перший стовбець задає час з шагом 10 с. Структура всіх файлів однакова:



**Критичні змінні:**

- напруга бортової мережі (даний параметр може безпосередньо призвести до аварійної ситуації, якщо напруга бортової мережі стане нижчою за рівень відключення медобладнання)

- кількість палива (зниження до рівня певної відмітки може призвести до нештатної або аварійної ситуації)

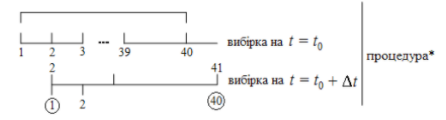
- напруга аккумуляторної батареї (залежить від стану генераторів, сумарного споживання електроенергії)

**1. Опис основного алгоритму**

Реалізація процедури квантування вихідних даних виконується наступним чином:

1. В якості базової еталонної статистики за кожною змінною приймаються статистики випадкової вибірки за цими змінними об’ємом .

2. В якості базової динамічної статистики за тими ж змінними приймаються статистики вибірки динаміки функціонування об’єкта за останні вимірів. Тому при черговому вимірі повинна проводитися процедура відкидання першого виміру існуючої вибірки та перенумерація вимірів.



3. В якості поточних динамічних статистик приймаються статистики

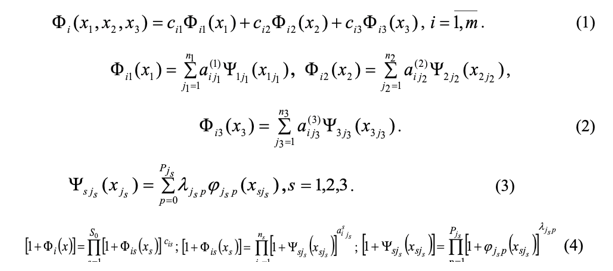
вибірок об’ємом після процедури, зазначеної на графіку вище (\*).

4. В якості зміщених динамічних статистик приймаються статистики

вибірок об’ємом , зсунутих відносно вибірок об’ємом на величину .

Моделі застосовують для екстраполяції реальних даних на множині.

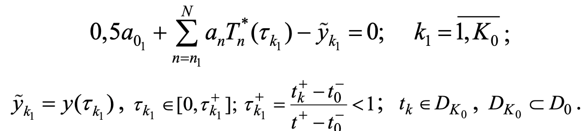
Фукціональні залежності відновлюються у вигляді:



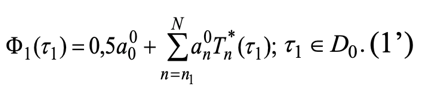
Моделі прогнозування нестаціонарних процесів будуються на основі вихідної вибірки часового ряду для інтервалу і базової моделі динаміки процесів.

**Прогнозування нестаціонарних процесів**

Задля цього застосовуються поліноми Чебишева - за допомогою них проводиться пошук найкращого наближення в сенсі критерію оптимальності Чебишева. Вихідні дані нормуються у інтервалі , який включає вихідний інтервал спостереження. Далі за вихідними даними у мультиплікативній (4) чи адитивній (1) формі шукається наближення завдяки розв’язку несумісної системи рівнянь:



Модель динамічного процесу в інтервалі спостережень визначається як результат розв’язку системи і описуються відношеннями:



**Дослідження нештатного режиму**

Припускаємо, що фактори ризику є незалежними і зміни в часі відбуваються за випадковим законом, розподіл якого є апріорно невідомим, вплив яких розповсюджується одночасно на декілька чи на всі показники. Ситуація впливу факторів ризику буде нештатною, якщо хоча б два показника одночасно, без керуючої дії, синхронно і синфазно змінять свої значення протягом декількох часових вимірів.

Процедура виявлення нештатної ситуації побудована на використанні порогових значень для датчиків. Тобто якщо є такий j, при якому виконується yj < (або >) yj нешт , то ситуацію вважаємо нештатною. Аналогічно з аварійною ситуацією: якщо yj < (або >) yj авар , то ситуацію вважаємо аварійною

Введемо значення у для нештатних та аварійних випадків для різних датчиків:

у1 нешт = 945 (тобто мала напруга бортової мережі, щоб в машині здійснювались всі функції)

у2 нешт = залишок шляху в годинах / 0,5 л/год. (тобто чи достатньо палива до кінця шляху)

у3 нешт = 5000000 (тобто мала напруга АБ, щоб в машині здійснювались всі функції)

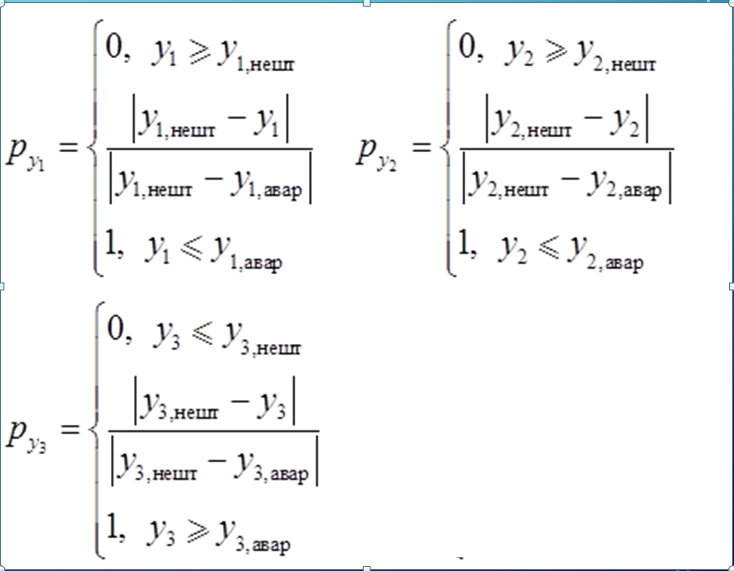
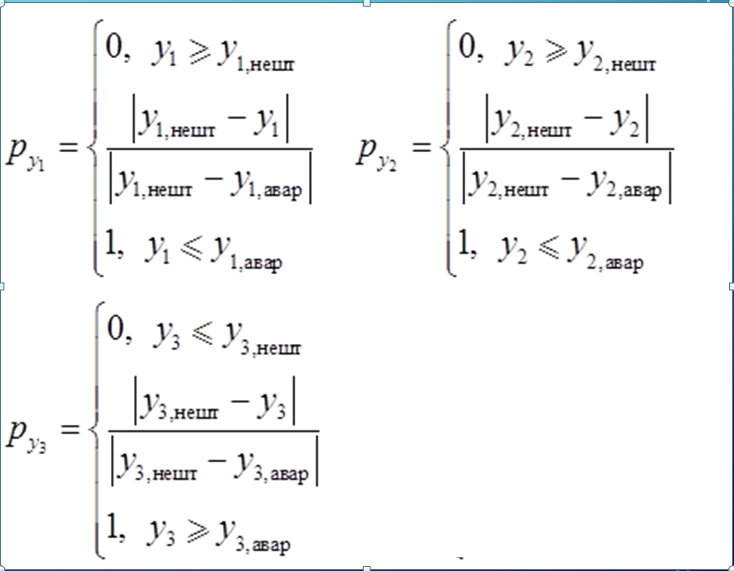
у1 авар = у2 авар = у3 авар = 0

у всіх випадках ми використовуємо знак <.

**Попоредження водія про настання нештатної ситуації**

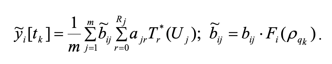
На основі прийнятих припущень представимо додатково моделі і умови

розпізнавання нештатної ситуації. Введемо наступні фактори ризику:



Після знаходження фактору ризику за кожним параметром окремо, виконується оцінка степеню загального ризику:

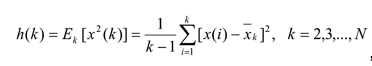
Формула для знаходження величини показника з урахуванням впливу факторів:



За відсутності дії факторів ризику формула дає точне значення

**Ситуація збою датчиків**

Також можлива ситуація збою датчиків, що може привести до аварії. Для уникнення такої ситуації відбудеться повідомлення про те, що трапився збій. Для його дослідження умовна дисперсія процесу вираховується послідовно для дослідного ряду:

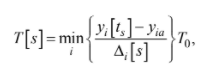


Де  – умовне математичне сподівання.

Та якщо та на інших датчиках не відображається таких відхилень, то це вважаємо за збій датчика.

Також розраховуємо ресурс допустимого ризику аварійної ситуації:

** –** максимальна зміна кожного показника за один часовий відрізок в нештатному режимі за всіма даними вибірки,

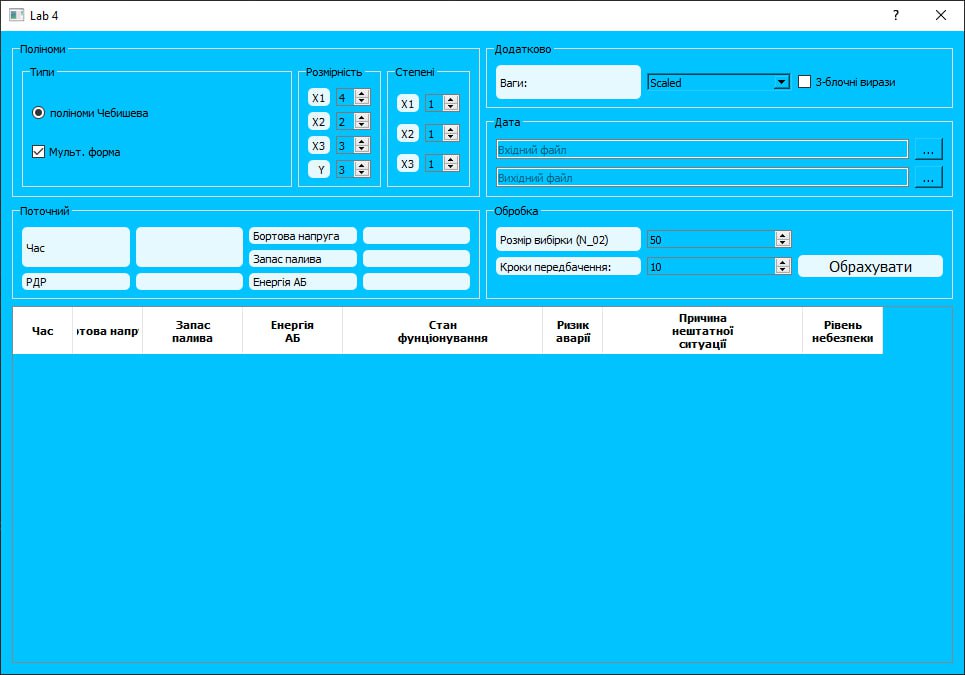
 – ресурс допустимого ризику (оцінюється ресурс

допустимого ризику, як час, що залишився до аварії, при найгіршому варіанті розвитку ситуації), де *Т0* – тривалість часового відрізку.

Введемо наступну класифікацію рівнів ризику:

| 0 | Безпечна ситуація |
| --- | --- |
| 1 | Нештатна ситуація за одним параметром |
| 2 | Нештатна ситуація за декількома параметрами |
| 3 | Спостерігається загроза аварії |
| 4 | Висока загроза аварії |
| 5 | Критична ситуація |
| 6 | Шанс уникнути аварії винятково малий |
| 7 | Аварія |

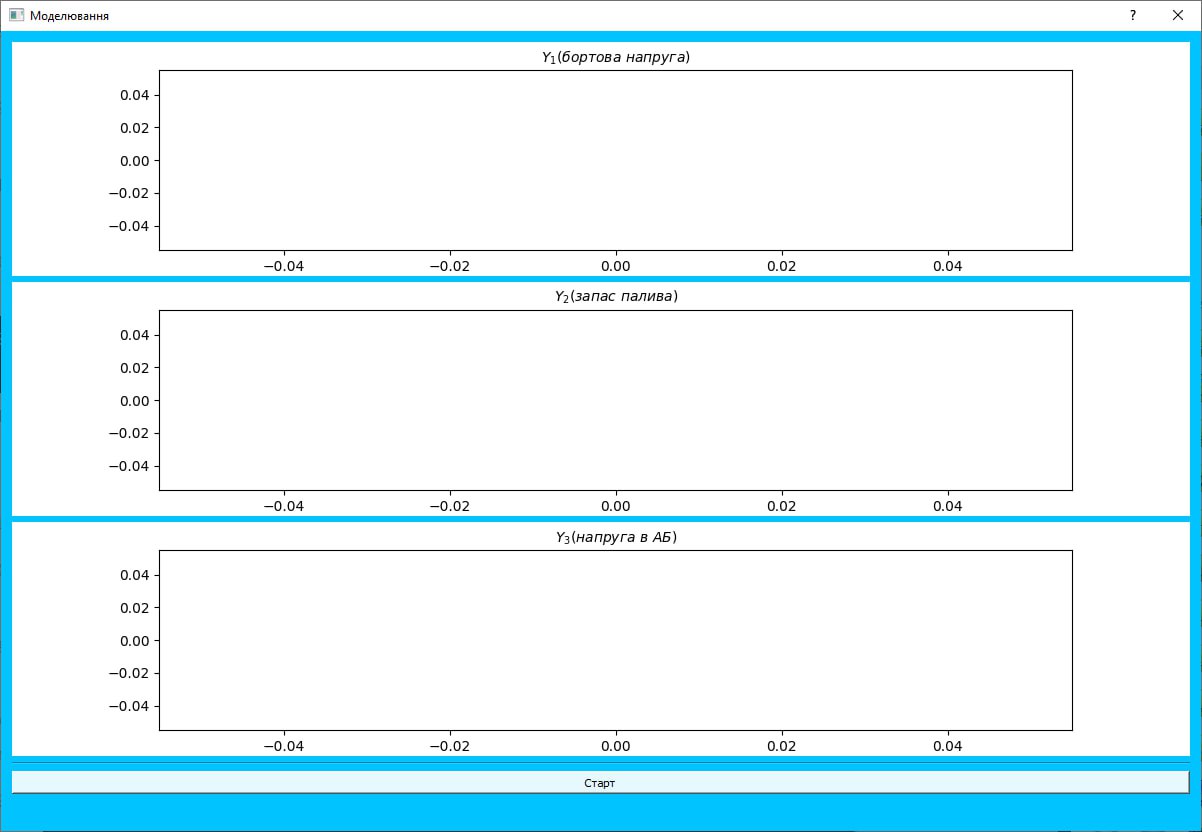
**2**. **Опис інтерфейсу:**



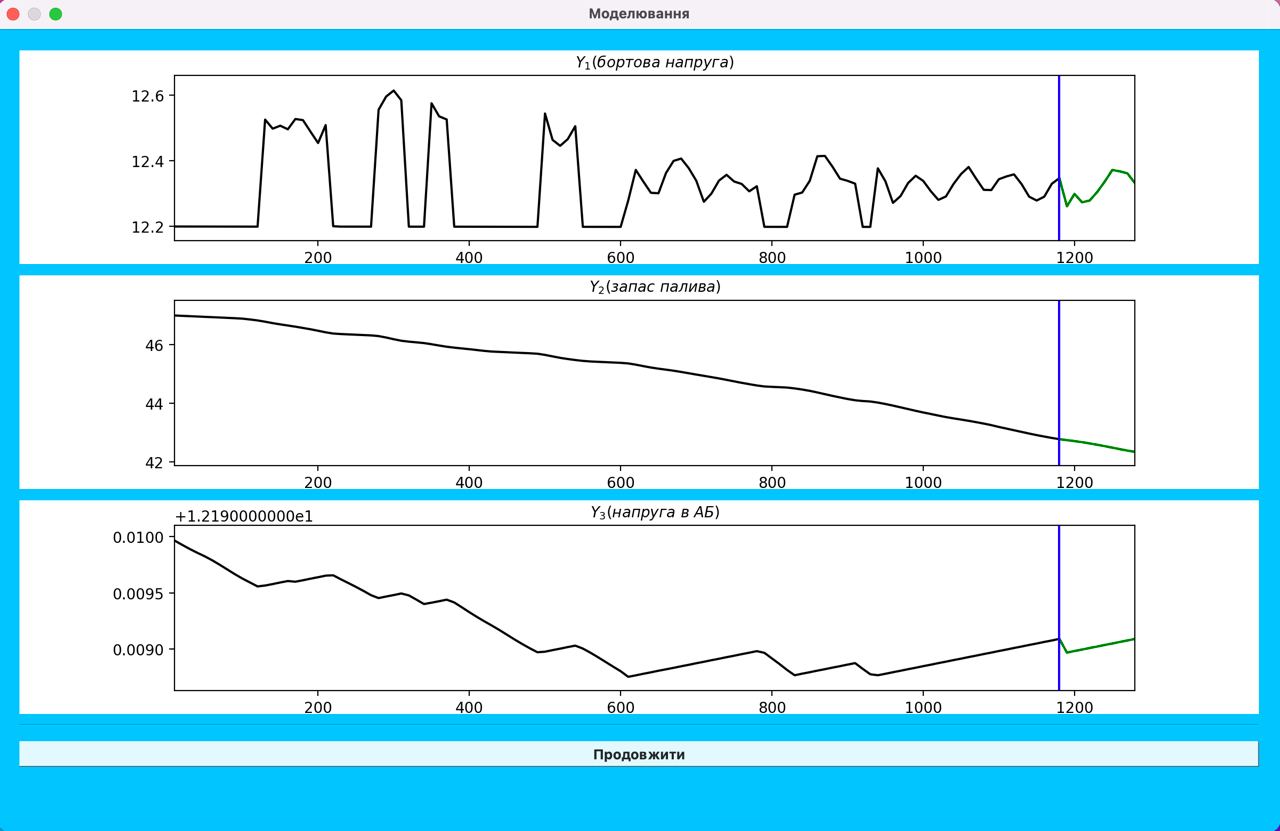
* У блоці **Типи** вибираємо тип поліномів, що фігурують у побудові функцій . Також передбачена мультипликативна форма поліномів.
* У блоці **Розмірність** задаємо відповідно розмірності векторів Х1, Х2, Х3, Y
* У блоці **Степені** вибираємо степені поліномів, що фігурють у побудові
* У блоці **Додатково** ми передбачили вибір одного з двох варіантів визначення (scaled & average). Передбачено функціонал визначення з 3 систем рівнянь
* У блоці **Дата** задаємо файл вхідних даних, вихідних даних
* У блоці **Обробка** задаємо Розмір вибірки та кроки передбачення
* У блоці **Поточний** виводиться поточний стан системи (Час, Бортова напруга, Запас палива, Енергія АБ)
* Кнопка "**Обрахувати**" починає роботу програми.
* У нижньому полі виводимо результати роботи алгоритму.

**3. Виконання роботи програми:**

При натистанні на кнопку **Обрахувати** з’являється додаткове вікно, на якому є кнопка **Старт**, що починає процес «руху» автомобілю



Після натискання **Старт**у, починається робота

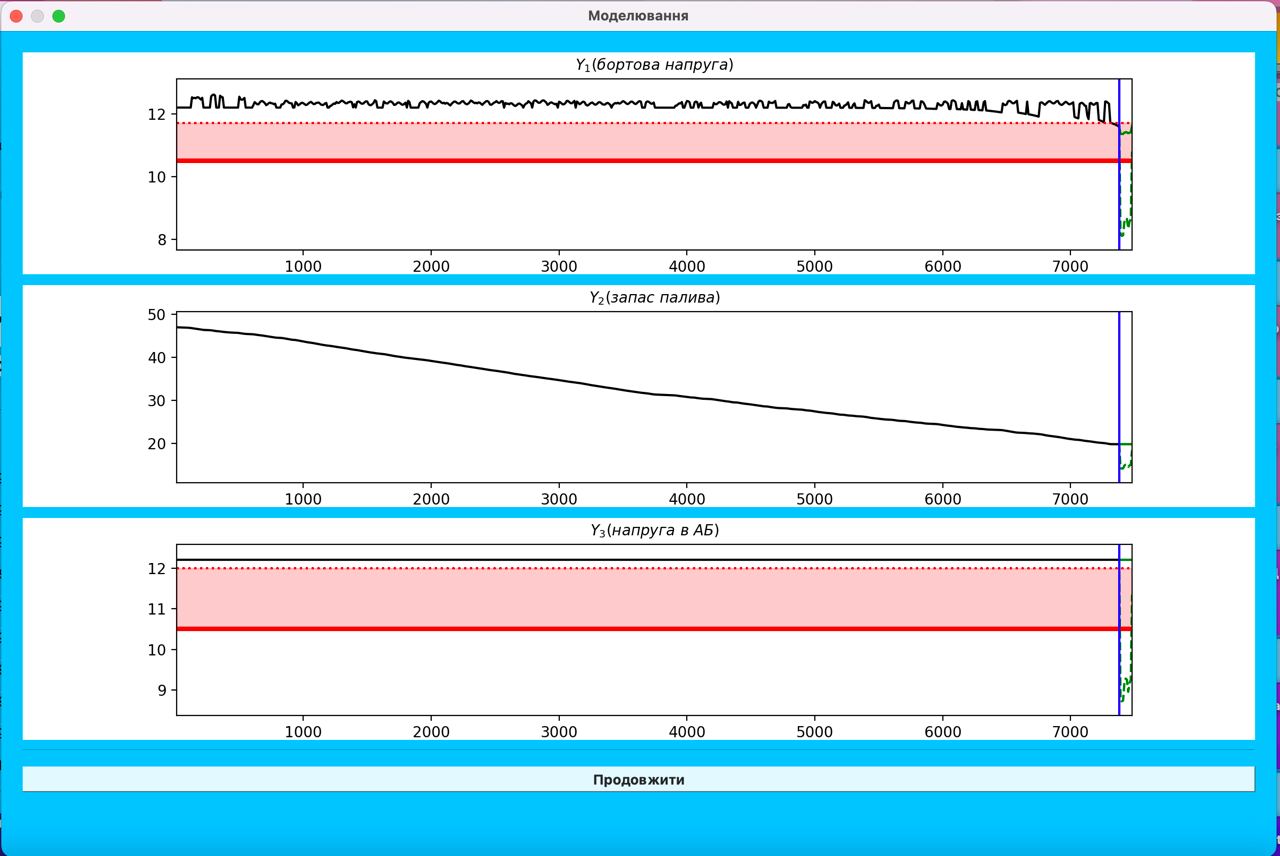


Чорна лінія відповідає введеним значенням системи, а зелена – прогнозованим. Вертикальна синя лінія позначає поточний стан системи. Червоний пунктир і площина під ним відповідає аварійному стану.

Водночас в початковому вікні виводитися стан системи більш детально в табличному форматі із числовими значеннями:



Далі ми застосували дані з позаштатною ситуацією:





**4. Висновок**

В процесі виконання, та із отриманням результатів даної лабораторної роботи ми на практиці застосували теоретичну і практичну складову з пройденого курсу системного аналізу. Зокрема, ми відтворили функціональні залежності у мультиплікативній та аддитивній формах. Також було на практиці помічено і застосовано в аналізі множину взаємозалежних факторів, а отже нами була виконана задача реального системного аналізу із враховуванням багатофакторних ризиців. В тому числі були засвоєні навички роботи із задачами прогнозування і управління, попередження нештатних і аварійних режимів функціонування складної системи та повернення її у штатний режим. Врешті ми створили систему прогнозування та оцінки ризиців, що показала свою ефективність в умовах наявності багатофакторних ризиків.

Програма була виконана за допомогою мови програмування Python та фрейворку PyQt.

**5. Список використаної літератури**

* "Основи системного аналізу" М.З. Згуровський, Н.Д. Панкратова - К.: Видавнича група BHV, 2007;
* "Аналіз часових рядів" П. І. Бідюк, В. Д. Романенко, О. Л. Тимощук. – Київ : НТУУ «КПІ», 2010;

* Панкратова Н.Д. «Системний аналіз в динаміці диагностування складних технічних систем» Київ : НТУУ «КПІ», 2008 (*Запропоновано інструментарій системного аналізу діагностування складних технічних систем, що базується на своєчасному виявленні ситуацій ризику та оперативному запобіганні переходу штатних ситуацій в критичні, надзвичайні або аварійні як основи стратегії управління ризиками та забезпечення гарантованої безпеки у динаміці функціонування складних технічних систем*);
* Ермолаев В.А., Н.К. Юрков, Романенко Ю.А «Риски отказов сложных технических систем» (*На основі припущення про схожість статистичних характеристик помилок і дефектів Програмного Забезпечення були отримані вирази для оцінки максимальної правдоподібності, на основі чого можна виявляти середінй час до появи дефекта СТС. Автори розглянули випадки, коли точність моделі також залежить в тому числі і від ПЗ, показали, що із даною задачею добре справляються поліноми Чебишева*);
* Червоненкіс А.Я. Комп'ютерний аналіз даних, 2009. (*Головною перевагою цієї книги є дуже зрозумілий виклад складних методів, які часто використовуються в комп'ютерних програмах статистичного аналізу даних. Книга А. Червоненкіса — це перша повна наукова праця з теорії машинного навчання, головною проблемою якого ставиться саме обґрунтування якості моделей, що навчаються. Проблема обґрунтування для автора є джерелом пошуку нових шляхів розвитку теорії машин, що навчаються*);
* А.М. Лепихин, В.В. Москвичев, С.В. Доронин «Надежность, живучесть и безопасность сложных технических систем» (*В роботі була розроблена технологі моніторингу та прогнозування катастроф, досліджено причино-наслідковий комплекс аварій і катастроф, живучесті СТС в аварійних ситуаціях, приведен математичний апарат для дослідження нових видів помилок (наприклад, новий метод оцінки міри ризику) і порушено багато питань з приводу раціональності дослідження складних моделей різноманітних оцінок, свідчення різноманіття ситуацій до обмеженого числа схем, достатнього для вирішення*);
* Fuzzy Systems and Data Mining : Proceedings of FSDM 2016 (Frontiers in Artificial Intelligence and Applications) *(На прикладі прогнозування ризику збою в роботі ЕОМ будують різні моделі, тільки в аналізі ризику використовують модель класифікації вивчиною за принципом ‘навчання з учителем’);*
* Python 3.8.0 Documentation.

**6. Лістинг програми**

Main.py

# coding: utf8

import matplotlib

#matplotlib.use("Qt5Agg", force=True)

import sys

from PyQt5.QtCore import pyqtSlot, pyqtSignal, Qt

from PyQt5.QtGui import QTextDocument, QFont

from PyQt5.QtWidgets import QApplication, QDialog, QFileDialog, QMessageBox, QTableWidgetItem

from PyQt5.uic import loadUiType

from solver\_manager import \* #SolverManager

from bruteforce import BruteForceWindow

from main\_window import Ui\_Form

app = QApplication(sys.argv)

app.setApplicationName('lab4\_sa')

# form\_class, base\_class = loadUiType('data/main\_window.ui')

class MainWindow(QDialog, Ui\_Form):

# signals:

input\_changed = pyqtSignal('QString')

output\_changed = pyqtSignal('QString')

def \_\_init\_\_(self, \*args):

super(MainWindow, self).\_\_init\_\_(\*args)

# setting up ui

self.setupUi(self)

# other initializations

self.dimensions = [self.x1\_dim.value(), self.x2\_dim.value(),

self.x3\_dim.value(), self.y\_dim.value()]

self.degrees = [self.x1\_deg.value(), self.x2\_deg.value(), self.x3\_deg.value()]

self.type = 'null'

if self.radio\_sh\_cheb.isChecked():

self.type = 'sh\_cheb\_doubled'

elif self.radio\_cheb.isChecked():

self.type = 'cheb'

elif self.radio\_sh\_cheb\_2.isChecked():

self.type = 'sh\_cheb\_2'

self.custom\_func\_struct = self.custom\_check.isChecked()

self.input\_path = self.line\_input.text()

self.output\_path = self.line\_output.text()

self.samples\_num = self.sample\_spin.value()

self.lambda\_multiblock = self.lambda\_check.isChecked()

self.weight\_method = self.weights\_box.currentText().lower()

self.manager = None

#set tablewidget

self.tablewidget.verticalHeader().hide()

self.tablewidget.setRowCount(0)

column\_size = [60, 70, 100, 100,200, 60, 200,80]

for index, size in enumerate(column\_size):

self.tablewidget.setColumnWidth(index,size)

return

@pyqtSlot()

def input\_clicked(self):

filename = QFileDialog.getOpenFileName(self, 'Open data file', '.', 'Data file (\*.csv)')[0]

if filename == '':

return

if filename != self.input\_path:

self.input\_path = filename

self.input\_changed.emit(filename)

return

@pyqtSlot('QString')

def input\_modified(self, value):

if value != self.input\_path:

self.input\_path = value

return

@pyqtSlot()

def output\_clicked(self):

filename = QFileDialog.getSaveFileName(self, 'Save data file', '.', 'Spreadsheet (\*.xlsx)')[0]

if filename == '':

return

if filename != self.output\_path:

self.output\_path = filename

self.output\_changed.emit(filename)

return

@pyqtSlot('QString')

def output\_modified(self, value):

if value != self.output\_path:

self.output\_path = value

return

@pyqtSlot(int)

def samples\_modified(self, value):

self.samples\_num = value

return

@pyqtSlot(int)

def dimension\_modified(self, value):

sender = self.sender().objectName()

if sender == 'x1\_dim':

self.dimensions[0] = value

elif sender == 'x2\_dim':

self.dimensions[1] = value

elif sender == 'x3\_dim':

self.dimensions[2] = value

elif sender == 'y\_dim':

self.dimensions[3] = value

return

@pyqtSlot(int)

def degree\_modified(self, value):

sender = self.sender().objectName()

if sender == 'x1\_deg':

self.degrees[0] = value

elif sender == 'x2\_deg':

self.degrees[1] = value

elif sender == 'x3\_deg':

self.degrees[2] = value

return

@pyqtSlot(bool)

def type\_modified(self, isdown):

if (isdown):

sender = self.sender().objectName()

if sender == 'radio\_sh\_cheb':

self.type = 'sh\_cheb\_doubled'

elif sender == 'radio\_cheb':

self.type = 'cheb'

elif sender == 'radio\_sh\_cheb\_2':

self.type = 'sh\_cheb\_2'

return

@pyqtSlot(bool)

def structure\_changed(self, isdown):

self.custom\_func\_struct = isdown

@pyqtSlot()

def plot\_clicked(self):

if self.manager:

try:

self.manager.plot(self.predictBox.value())

except Exception as e:

QMessageBox.warning(self,'Error!','Error happened during plotting: ' + str(e))

return

@pyqtSlot()

def exec\_clicked(self):

self.exec\_button.setEnabled(False)

try:

self.tablewidget.setRowCount(self.predictBox.value())

self.manager = SolverManager(self.\_get\_params())

self.manager.prepare(self.input\_path)

except Exception as e:

QMessageBox.warning(self,'Error!','Error happened during execution: ' + str(e))

self.exec\_button.setEnabled(True)

return

@pyqtSlot()

def bruteforce\_called(self):

BruteForceWindow.launch(self)

return

@pyqtSlot(int, int, int)

def update\_degrees(self, x1\_deg, x2\_deg, x3\_deg):

self.x1\_deg.setValue(x1\_deg)

self.x2\_deg.setValue(x2\_deg)

self.x3\_deg.setValue(x3\_deg)

return

@pyqtSlot(bool)

def lambda\_calc\_method\_changed(self, isdown):

self.lambda\_multiblock = isdown

return

@pyqtSlot('QString')

def weights\_modified(self, value):

self.weight\_method = value.lower()

return

def \_get\_params(self):

return dict(custom\_struct=self.custom\_func\_struct,poly\_type=self.type, degrees=self.degrees,

dimensions=self.dimensions,

samples=self.samples\_num, output\_file=self.output\_path,

weights=self.weight\_method, lambda\_multiblock=self.lambda\_multiblock,

pred\_steps = 10, tablewidget = self.tablewidget, \

lbl = {'rmr':self.lbl\_rmr, 'time': self.lbl\_time, 'y1': self.lbl\_y1,\

'y2':self.lbl\_y2, 'y3':self.lbl\_y3})

# -----------------------------------------------------#

form = MainWindow()

form.setWindowTitle('Lab 4')

form.show()

sys.exit(app.exec\_())

Presentation.py

import numpy as np

import matplotlib.pyplot as plt

from numpy.polynomial import Polynomial as pnm

from os import name as os\_name

from solve import Solve

import basis\_generator as b\_gen

import tkinter

from tkinter import filedialog

import matplotlib.backends.backend\_tkagg

class PolynomialBuilder(object):

def \_\_init\_\_(self, solution):

assert isinstance(solution, Solve)

self.\_solution = solution

max\_degree = max(solution.deg) - 1

if solution.poly\_type == 'sh\_cheb\_doubled':

self.symbol = 'T'

self.basis = b\_gen.basis\_sh\_chebyshev(max\_degree)

elif solution.poly\_type == 'cheb':

self.symbol = 'K'

self.basis = b\_gen.basis\_chebyshev(max\_degree)

elif solution.poly\_type == 'sh\_cheb\_2':

self.symbol = 'U'

self.basis = b\_gen.basis\_sh\_chebyshev\_2\_shrinked(max\_degree)

assert self.symbol

self.a = solution.a.T.tolist()

self.c = solution.c.T.tolist()

self.minX = [X.min(axis=0).getA1() for X in solution.X\_]

self.maxX = [X.max(axis=0).getA1() for X in solution.X\_]

self.minY = solution.Y\_.min(axis=0).getA1()

self.maxY = solution.Y\_.max(axis=0).getA1()

def \_form\_lamb\_lists(self):

"""

Generates specific basis coefficients for Psi functions

"""

self.lamb = list()

for i in range(self.\_solution.Y.shape[1]): # `i` is an index for Y

lamb\_i = list()

shift = 0

for j in range(3): # `j` is an index to choose vector from X

lamb\_i\_j = list()

for k in range(self.\_solution.dim[j]): # `k` is an index for vector component

lamb\_i\_jk = self.\_solution.Lamb[shift:shift + self.\_solution.deg[j], i].getA1()

shift += self.\_solution.deg[j]

lamb\_i\_j.append(lamb\_i\_jk)

lamb\_i.append(lamb\_i\_j)

self.lamb.append(lamb\_i)

def \_print\_psi\_i\_jk(self, i, j, k):

"""

Returns string of Psi function in special polynomial form

:param i: an index for Y

:param j: an index to choose vector from X

:param k: an index for vector component

:return: result string

"""

strings = list()

for n in range(len(self.lamb[i][j][k])):

strings.append('(1 + {symbol}{deg}(x{1}{2}))^({0:.6f})'.format(self.lamb[i][j][k][n], j + 1, k + 1,

symbol=self.symbol, deg=n))

return ' \* '.join(strings)

def \_print\_phi\_i\_j(self, i, j):

"""

Returns string of Phi function in special polynomial form

:param i: an index for Y

:param j: an index to choose vector from X

:return: result string

"""

strings = list()

for k in range(len(self.lamb[i][j])):

shift = sum(self.\_solution.dim[:j]) + k

for n in range(len(self.lamb[i][j][k])):

strings.append('(1 + {symbol}{deg}(x{1}{2}))^({0:.6f})'.format(self.a[i][shift] \* self.lamb[i][j][k][n],

j + 1, k + 1, symbol=self.symbol, deg=n))

return ' \* '.join(strings)

def \_print\_F\_i(self, i):

"""

Returns string of F function in special polynomial form

:param i: an index for Y

:return: result string

"""

strings = list()

for j in range(3):

for k in range(len(self.lamb[i][j])):

shift = sum(self.\_solution.dim[:j]) + k

for n in range(len(self.lamb[i][j][k])):

strings.append('(1 + {symbol}{deg}(x{1}{2}))^({0:.6f})'.format(self.c[i][j] \* self.a[i][shift] \*

self.lamb[i][j][k][n],

j + 1, k + 1, symbol=self.symbol,

deg=n))

return ' \* '.join(strings)

def \_print\_F\_i\_transformed(self, i):

"""

Returns string of F function in regular polynomial form

:param i: an index for Y

:return: result string

"""

strings = list()

power\_sum = 0

for j in range(3):

for k in range(len(self.lamb[i][j])):

shift = sum(self.\_solution.dim[:j]) + k

power\_sum += self.c[i][j] \* self.a[i][shift] \* self.lamb[i][j][k][0]

for n in range(1, len(self.lamb[i][j][k])):

summands = ['{0}(x{1}{2})^{deg}'.format(self.basis[n].coef[index], j + 1, k + 1, deg=index)

for index in range(1, len(self.basis[n].coef)) if self.basis[n].coef[index] != 0]

if self.basis[n].coef[0] != -1:

summands.insert(0, str(1 + self.basis[n].coef[0]))

strings.append('({repr})^({0:.6f})'.format(self.c[i][j] \* self.a[i][shift] \* self.lamb[i][j][k][n],

j + 1, k + 1, repr=' + '.join(summands)))

strings.insert(0, str((1 + self.basis[0].coef[0]) \*\* (power\_sum)))

return ' \* '.join(strings)

def \_print\_F\_i\_transformed\_recovered(self, i):

"""

Returns string of recovered F function in regular polynomial form

:param i: an index for Y

:return: result string

"""

strings = list()

power\_sum = 0

for j in range(3):

for k in range(len(self.lamb[i][j])):

shift = sum(self.\_solution.dim[:j]) + k

diff = self.maxX[j][k] - self.minX[j][k]

mult\_poly = pnm([- self.minX[j][k] / diff, 1 / diff])

power\_sum += self.c[i][j] \* self.a[i][shift] \* self.lamb[i][j][k][0]

for n in range(1, len(self.lamb[i][j][k])):

res\_polynomial = self.basis[n](mult\_poly) + 1

coeffs = res\_polynomial.coef

summands = ['{0}(x{1}{2})^{deg}'.format(coeffs[index], j + 1, k + 1, deg=index)

for index in range(1, len(coeffs))]

summands.insert(0, str(coeffs[0]))

strings.append('({repr})^({0:.6f})'.format(self.c[i][j] \* self.a[i][shift] \* self.lamb[i][j][k][n],

j + 1, k + 1, repr=' + '.join(summands)))

strings.insert(0, str((self.maxY[i] - self.minY[i]) \* (1 + self.basis[0].coef[0]) \*\* (power\_sum)))

return ' \* '.join(strings) + ' + ' + str((2 \* self.minY[i] - self.maxY[i]))

def get\_results(self):

"""

Generates results based on given solution

:return: Results string

"""

self.\_form\_lamb\_lists()

psi\_strings = ['Psi^{0}\_[{1},{2}]={result} - 1\n'.format(i + 1, j + 1, k + 1,

result=self.\_print\_psi\_i\_jk(i, j, k))

for i in range(self.\_solution.Y.shape[1])

for j in range(3)

for k in range(self.\_solution.dim[j])]

phi\_strings = ['Phi^{0}\_[{1}]={result} - 1\n'.format(i + 1, j + 1, result=self.\_print\_phi\_i\_j(i, j))

for i in range(self.\_solution.Y.shape[1])

for j in range(3)]

f\_strings = ['F^{0} in special basis:\n{result} - 1\n'.format(i + 1, result=self.\_print\_F\_i(i))

for i in range(self.\_solution.Y.shape[1])]

f\_strings\_transformed = [

'F^{0} in standard basis:\n{result} - 1\n'.format(i + 1, result=self.\_print\_F\_i\_transformed(i))

for i in range(self.\_solution.Y.shape[1])]

f\_strings\_transformed\_denormed = ['F^{0} in standard basis '

'denormed:\n{result}\n'.format(i + 1, result=

self.\_print\_F\_i\_transformed\_recovered(i))

for i in range(self.\_solution.Y.shape[1])]

return '\n'.join(psi\_strings + phi\_strings + f\_strings + f\_strings\_transformed + f\_strings\_transformed\_denormed)

def plot\_graphs(self):

fig, axes = plt.subplots(2, self.\_solution.Y.shape[1])

if self.\_solution.Y.shape[1] == 1:

axes[0] = [axes[0]]

axes[1] = [axes[1]]

for index in range(self.\_solution.Y.shape[1]):

ax = axes[0][index] # real and estimated graphs

norm\_ax = axes[1][index] # abs residual graph

ax.set\_xticks(np.arange(0, self.\_solution.n + 1, 5))

ax.plot(np.arange(1, self.\_solution.n + 1), self.\_solution.Y\_[:, index],

'r-', label='$Y\_{0}$'.format(index + 1))

ax.plot(np.arange(1, self.\_solution.n + 1), self.\_solution.F\_[:, index],

'b-', label='$F\_{0}$'.format(index + 1))

ax.legend(loc='upper right', fontsize=16)

ax.set\_title('Coordinate {0}'.format(index + 1))

ax.grid()

norm\_ax.set\_xticks(np.arange(0, self.\_solution.n + 1, 5))

norm\_ax.plot(np.arange(1, self.\_solution.n + 1),

abs(self.\_solution.Y\_[:, index] - self.\_solution.F\_[:, index]), 'g-')

norm\_ax.set\_title('Residual {0}'.format(index + 1))

norm\_ax.grid()

manager = plt.get\_current\_fig\_manager()

manager.set\_window\_title('Graph')

if os\_name == 'posix':

fig.show()

else:

plt.show()

def compare\_vals(self, name, real, predicted, reconstructed=None):

fig = plt.figure()

axes = plt.axes()

r = np.arange(len(real))

axes.set\_title(name)

axes.set\_xlim(0, len(real))

axes.grid()

axes.plot(r, predicted, label='predicted')

if reconstructed != None:

axes.plot(r, reconstructed, label='reconstructed')

axes.plot(r, real, label='real')

axes.legend(loc='upper right', fontsize=16)

if os\_name == 'posix':

fig.show()

else:

plt.show()

def plot\_graphs\_with\_prediction(self, steps):

XF, YF = self.\_solution.build\_predicted(steps)

for i, x in enumerate(self.\_solution.X\_):

for j, xc in enumerate(x.T):

self.compare\_vals('X{}{}'.format(i + 1, j + 1), xc.getA1(), XF[i][j])

for i in range(self.\_solution.dim[3]):

self.compare\_vals('Y{}'.format(i + 1), self.\_solution.Y\_[:, i].getA1(), YF[:, i],

self.\_solution.F\_[:, i].getA1())

class PolynomialBuilderExpTh(PolynomialBuilder):

def \_print\_psi\_i\_jk(self, i, j, k, mode=0):

"""

Returns string of Psi function

mode = 0 - in special polynomial form

mode = 1 - in regular polynomial form

mode = 2 - in regular polynomial form with restored X

:param i: an index for Y

:param j: an index to choose vector from X

:param k: an index for vector component

:return: result string

"""

strings = list()

for n in range(len(self.lamb[i][j][k])):

inner = 'stub'

if mode == 0:

inner = '{symbol}{deg}(x{0}{1})'.format(j + 1, k + 1, symbol=self.symbol, deg=n)

elif mode == 1:

inner = str(self.basis[n].coef[0])

if n > 0:

inner += ' + ' + ' + '.join('({coef})(x{0}{1})^{deg}'.format(j + 1, k + 1, coef=coef, deg=index)

for index, coef in enumerate(self.basis[n].coef) if index > 0)

elif mode == 2:

diff = self.maxX[j][k] - self.minX[j][k]

mult\_poly = pnm([- self.minX[j][k] / diff, 1 / diff])

cur\_poly = self.basis[n](mult\_poly)

inner = str(cur\_poly.coef[0])

if n > 0:

inner += ' + ' + ' + '.join('({coef})(x{0}{1})^{deg}'.format(j + 1, k + 1, coef=coef, deg=index)

for index, coef in enumerate(cur\_poly.coef) if index > 0)

strings.append('exp({0:.6f}\*tanh({inner}))'.format(self.lamb[i][j][k][n], inner=inner))

return ' \* '.join(strings) + ' - 1'

def \_print\_phi\_i\_j(self, i, j, mode=0):

"""

Returns string of Phi function in special polynomial form

:param i: an index for Y

:param j: an index to choose vector from X

:return: result string

"""

strings = list()

for k in range(len(self.lamb[i][j])):

strings.append('exp({0:.6f}\*tanh({inner}))'.format(self.a[i][sum(self.\_solution.dim[:j]) + k],

inner=self.\_print\_psi\_i\_jk(i, j, k, mode)))

return ' \* '.join(strings) + ' - 1'

def \_print\_F\_i(self, i, mode=0):

"""

Returns string of F function in special polynomial form

:param i: an index for Y

:return: result string

"""

strings = list()

for j in range(3):

strings.append('exp({0:.6f}\*tanh({inner}))'.format(self.c[i][j], inner=self.\_print\_phi\_i\_j(i, j, mode)))

if mode == 2:

strings.insert(0, str(self.maxY[i] - self.minY[i]))

return ' \* '.join(strings) + ' + (' + str((2 \* self.minY[i] - self.maxY[i])) + ')'

else:

return ' \* '.join(strings) + ' - 1'

def get\_results(self):

"""

Generates results based on given solution

:return: Results string

"""

self.\_form\_lamb\_lists()

psi\_strings = ['Psi^{0}\_[{1},{2}]={result}\n'.format(i + 1, j + 1, k + 1,

result=self.\_print\_psi\_i\_jk(i, j, k))

for i in range(self.\_solution.Y.shape[1])

for j in range(3)

for k in range(self.\_solution.dim[j])]

phi\_strings = ['Phi^{0}\_[{1}]={result}\n'.format(i + 1, j + 1, result=self.\_print\_phi\_i\_j(i, j))

for i in range(self.\_solution.Y.shape[1])

for j in range(3)]

f\_strings = ['F^{0} in special basis:\n{result}\n'.format(i + 1, result=self.\_print\_F\_i(i))

for i in range(self.\_solution.Y.shape[1])]

f\_strings\_transformed = ['F^{0} in standard basis:\n{result}\n'.format(i + 1, result=self.\_print\_F\_i(i, mode=1))

for i in range(self.\_solution.Y.shape[1])]

f\_strings\_transformed\_denormed = ['F^{0} in standard basis '

'denormed:\n{result}\n'.format(i + 1, result=self.\_print\_F\_i(i, mode=2))

for i in range(self.\_solution.Y.shape[1])]

return '\n'.join(psi\_strings + phi\_strings + f\_strings + f\_strings\_transformed + f\_strings\_transformed\_denormed)

Solver\_manager.py

# -\*- encoding: utf-8 -\*-

from solve import \*

from solve\_custom import SolveExpTh

from read\_data import read\_data

from operator\_view import OperatorViewWindow

from PyQt5.QtCore import pyqtSlot, pyqtSignal, Qt

from PyQt5.QtGui import QTextDocument, QFont

from PyQt5.QtWidgets import QApplication, QDialog, QFileDialog, QMessageBox, QTableWidgetItem

from PyQt5.uic import loadUiType

import numpy as np

START\_FUEL = 47

TIME\_DELTA = 10

reason = [u'Мала бортова напруга\n',u'Недостатній запас палива\n',u'Низький рівень заряду АБ\n']

#reason = [u'Малий прибуток; ',u'Недостатній запас ходу; ',u'Низький рівень заряду АБ; ']

def calculate\_rdr\_delta(ycurrent, yf, yd):

maxl = np.max(yf[:-1] - yf[1:])

return (ycurrent - yd)/ maxl

def lblText(lbl, text):

lbl.setText(str(text)[:10])

return

def prob(x, xmax, xmin):

res = np.fabs((x - xmax) / (xmax - xmin))

r = np.ma.array(res, mask=np.array(x >= xmax), fill\_value=0)

return r.filled()

def insert\_data(tw, row, data):

assert len(data) <= 8

try:

for i, d in enumerate(data):

item = QTableWidgetItem(d)

item.setTextAlignment(Qt.AlignHCenter)

tw.setItem(row, i, item)

except Exception as e:

raise ('insert data in table' + str(e))

def classify\_danger\_rating(level):

if 0 <= level <= 0.07:

return 0, u"Безпечна ситуація"

elif 0.07 < level <= 0.25:

return 1, u"Нештатна ситуація по одному параметру"

elif 0.25 < level <= 0.375:

return 2, u"Нештатна ситуація по декількох параметрах"

elif 0.375 < level <= 0.5:

return 3, u"Спостерігається загроза аварії"

elif 0.5 < level <= 0.625:

return 4, u"Висока загроза аварії"

elif 0.625 < level <= 0.75:

return 5, u"Критична ситуація"

elif 0.75 < level <= 0.875:

return 6, u"Шанс уникнути аварії дуже малий"

elif 0.875 < level <= 1:

return 7, u"Аварія"

class SolverManager(object):

Y\_C = np.array([[11.7], [1], [12]]) # warning value

Y\_D = np.array([[10.5], [0.0], [10.5]]) # failure value

def \_\_init\_\_(self, d):

self.custom\_struct = d['custom\_struct']

d['dimensions'][3] = 1

if d['custom\_struct']:

self.solver = SolveExpTh(d)

else:

self.solver = Solve(d)

self.first\_launch = True

self.batch\_size = d['samples']

self.forecast\_size = d['pred\_steps']

self.operator\_view = OperatorViewWindow(warn=self.Y\_C, fail=self.Y\_D, callback=self,

descriptions=[u'бортова\ напруга', u'запас\ палива',

u'напруга\ в\ АБ'],

tail=self.forecast\_size)

self.current\_iter = 1

self.y\_influenced = None

self.data\_window = None

self.tablewidget = d['tablewidget']

self.reason = np.array([],dtype = str) # init warning reason

self.lbl = d['lbl']

def prepare(self, filename):

self.time, self.data = read\_data(filename)

increment = self.time[-1] - self.time[-2]

self.time = np.append(self.time, np.arange(1, 1 + self.forecast\_size) \* increment + self.time[-1])

self.N\_all\_iter = len(self.time)

self.operator\_view.show()

self.operator\_view.status\_bar.showMessage('Loaded successfully.', 1000)

def start\_machine(self):

self.operator\_view.start\_process()

def launch(self):

if self.current\_iter + self.batch\_size < len(self.data):

self.fit(self.current\_iter, self.batch\_size)

self.current\_iter += 1

else:

self.operator\_view.timer.stop()

def fit(self, shift, n):

# print('fit', shift, n)

self.data\_window = self.data[shift:shift + n]

self.solver.load\_data(self.data\_window[:, :-2]) # y2 and y3 not used

self.solver.prepare()

self.predict(shift, n)

self.check\_sensors\_consistency()

self.risk() # really suspicious realisation

self.y\_influenced = [y \* (1 - self.f) for y in self.y\_forecasted]

if self.first\_launch:

self.operator\_view.initial\_graphics\_fill(real\_values=self.data\_window[:, -3:],

predicted\_values=self.y\_forecasted,

risk\_values=self.y\_influenced,

time\_ticks=self.time[shift:shift + n + self.solver.pred\_step])

self.first\_launch = False

else:

self.operator\_view.update\_graphics(self.data\_window[-1, -3:], self.y\_forecasted, self.y\_influenced,

self.time[shift + n - 1:shift + n + self.solver.pred\_step])

fuel = (13120 - self.time[shift] ) / 13120 \* self.data\_window[-1, -3:][1]

self.y\_current = np.array([self.solver.Y\_[-1,0], fuel, self.solver.X\_[0][-1,0]])

# self.y\_current = np.array([self.solver.Y\_[-1,0], self.solver.X\_[0][-1,3], self.solver.X\_[1][-1,2]])

self.rdr\_calc()

self.current\_data()

self.table\_data\_forecasted()

def check\_sensors\_consistency(self):

# mask\_positive = np.array([True, True, True, True, True, True, False,

# True, True, True, True, True, False,

# False, True, True, True, False])

mask\_positive = np.array([False, False, False, False, False, False, False,

False, False, ])

result\_positive = (self.data\_window[:, :-3] < 0).max(axis=0) \* mask\_positive

if result\_positive.any():

self.operator\_view.status\_bar.showMessage('Sensors {} have problems'.format(

str(np.where(result\_positive)[0].tolist())), 1000)

else:

self.operator\_view.status\_bar.showMessage('OK',1000)

def risk(self):

self.p = prob(self.y\_forecasted, self.Y\_C, self.Y\_D)

# print('p', self.p)

self.f = 1 - (1 - self.p[0, :]) \* (1 - self.p[1, :]) \* (1 - self.p[2, :])

#calculate reason warning situation

self.reason = np.array([],dtype = str)

for i in range(self.forecast\_size):

string = ''

for j in range(3):

if self.p[j, i] > 0:

string = string + reason[j]

if string == '':

self.reason = np.append(self.reason, '-')

else:

self.reason = np.append(self.reason, string)

assert len(self.reason) == self.forecast\_size

self.danger\_rate = np.array([classify\_danger\_rating(i) for i in self.f])

def predict(self, shift, n):

fuel = np.array(list(

[self.data[self.current\_iter + self.batch\_size + i][-2] #+ random.randrange(-1, 1) / 50

for i in range(self.forecast\_size)]

))

voltage = np.array(list(

[x[0] for x in self.solver.X\_[0][-10:, 0].tolist()]

))

self.y\_forecasted = [self.solver.YF, fuel, voltage]

return

def table\_data\_forecasted(self):

t = self.time[self.batch\_size + self.current\_iter: self.batch\_size + self.current\_iter + self.solver.pred\_step]

y1 = self.y\_forecasted[0]

y2 = self.y\_forecasted[1]

y3 = self.y\_forecasted[2]

# print(self.danger\_rate)

state = self.danger\_rate[:, 1]

risk = self.f

reason =self.reason

rate = self.danger\_rate[:, 0]

data = np.array([t, y1, y2, y3, state, risk, reason, rate]).T

assert data.shape == (self.solver.pred\_step, 8)

for i ,j in enumerate(data):

insert\_data(self.tablewidget, i, j)

return

def current\_data(self):

lblText(self.lbl['time'], self.time[self.batch\_size + self.current\_iter -1])

lblText(self.lbl['y1'], self.y\_current[0])

lblText(self.lbl['y2'], self.y\_current[1])

lblText(self.lbl['y3'], self.y\_current[2])

lblText(self.lbl['rmr'], self.rdr)

def rdr\_calc(self):

self.rdr = np.inf

rdr = np.inf

for i in range(3):

if self.y\_current[i]>=self.Y\_C[i,0]:

continue

if self.y\_current[i]<= self.Y\_D [i,0]:

rdr = 0

continue

s = calculate\_rdr\_delta(self.y\_current[i], self.y\_influenced[i], self.Y\_D[i,0])

if s <= 0:

continue

t =(self.y\_current[i] - self.Y\_D[i,0])/s

if t < rdr:

rdr = t

if rdr != np.inf:

self.rdr = rdr