Федеральное государственное бюджетное образовательное учреждение высшего образования   
«Саратовский государственный технический университет имени Ю. А. Гагарина»

Кафедра прикладных информационных технологий

**Практическая работа по курсу**

**Теория информационных процессов и систем**

**на тему:**

**«Разработка информационной системы для прогнозирования и предотвращения критических отказов в роботизированных сварочных комплексах»**

Выполнили студенты группы

Б1-ИФСТ-42

Парамонова Анна,

Гудкова Анна,

Гудошникова Анастасия

Проверил

Доктор технических наук, профессор кафедры ПИТ

Кушников Вадим Алексеевич

Саратов, 2025

**Ссылка на сервер:**

<https://annayellowcat.pythonanywhere.com/>

**Цель работы** - разработка информационной системы для прогнозирования и предотвращения критических отказов в роботизированных сварочных комплексах. Система будет основана на методах анализа данных и вероятностного моделирования для выявления критических сочетаний событий в реальном времени. Реализация данной системы позволит минимизировать количество дефектов сварных швов, сократить время простоя оборудования и повысить общую надежность технологического процесса, что будет подтверждено апробацией на модели технологического процесса.

Базовая модель системной динамики



Х1 – количество забракованных балок на 100 единиц продукции;

Х2 – численность операторов РТК;

Х3 – среднее количество остановок РТК на один цикл;

Х4 – средняя длина дефектных сварных швов на 1 единицу продукции;

Х5 – выполненные работы по плановому обслуживанию РТК;

Х6 – численность программистов;

Х7 –численность наладчиков сварочного оборудования;

Х8 – численность контролеров ОТК;

Х9 – численность цеховых технологов;

Х10 – количество дней просрочки поставки материалов и запчастей для ремонта РТК;

Х11 – среднее отклонение напряжения сварочной дуги;

Х12 – среднее отклонение тока на двигателе подающего блока;

Х13 – среднее отклонение манипулятора от программной траектории;

Х14 – наличие на рабочих местах необходимой технологической документации;

Х15 – отклонение давления защитного газа;

Х16 – отклонение давления сжатого воздуха;

Х17 – план производства на заданный период в единицах продукции;

Х18 – количество балок, сданных ОТК с первого предъявления.

O0 – численность операторов РТК на начало периода;

Oin – численность принятых операторов РТК за период;

Oout – численность уволенных операторов РТК за период;

Sm – сменность работы производства;

Rw – количество РТК, задействованных в производственном процессе;

Nst – количество остановок РТК за период;

S\* – допустимое количество остановок РТК за один сварочный цикл;

Ld– общая длина дефектных швов за период;

L\* – расчетная длина дефектных швов за период;

Mf – количество выполненных мероприятий из графика ППР по обслуживанию РТК;

Mp – количество запланированных мероприятий из графика ППР по обслуживанию РТК;

P0 – численность программистов на начало периода;

Pin – численность принятых программистов за период;

Pout – численность уволенных программистов за период;

R0 – численность наладчиков сварочного оборудования на начало периода; Rin – численность принятых наладчиков сварочного оборудования за период;

Rout – численность уволенных наладчиков сварочного оборудования за период;

С0 – численность контролеров ОТК на начало периода;

Сin – численность принятых контролеров ОТК за период;

Сout – численность уволенных контролеров ОТК за период;

Т0 – численность технологов на начало периода;

Тin – численность принятых технологов за период;

Тout – численность уволенных технологов за период;

Nr – длительность ремонта РТК в днях;

Df – фактический срок поставки запчастей и материалов для ремонта РТК; Dp – запланированный срок поставки запчастей и материалов для ремонта РТК;

ΔU – среднее отклонение напряжения сварочной дуги от номинального значения;

Δ\*U – допустимое отклонение напряжения сварочной дуги от номинального значения;

ΔI – среднее отклонение тока на двигателе подающего механизма от номинального значения;

Δ\*I – допустимое отклонение тока на двигателе подающего механизма от номинального значения;

ΔT – среднее отклонение манипулятора от запрограммированной траектории;

Δ\*T – допустимое манипулятора от запрограммированной траектории;

Tdf – фактическое количество документов по технологическому процессу; Tdp – необходимое количество документов по технологическому процессу;

ΔPG – среднее отклонение давления защитного газа;

Δ\*PG – допустимое отклонение давления защитного газа;

ΔPV – среднее отклонение давления сжатого воздуха;

Δ\*PV – допустимое отклонение давления сжатого воздуха;

NTP – количество балок, собранных в соответствии с технологическим процессом;

Nd – количество балок, сданных с 1-го предъявления;

Ab – количество актов о несоответствующей продукции за период.

**Описание разработанного программного обеспечения**

Программный модуль был разработан на языке Python с использованием специализированных библиотек для научных вычислений. Для численного решения системы дифференциальных уравнений применялась функция solve\_ivp из библиотеки SciPy с методом Рунге-Кутта 4-5 порядка (RK45). Визуализация результатов выполнена с помощью библиотеки matplotlib.

Для построения модели использовалась система из 18 нелинейных дифференциальных уравнений, описывающих динамику параметров роботизированного сварочного комплекса. Каждое уравнение включает полиномиальные функции вида:

* , где коэффициенты a₀, a₁, a₂, a₃ для каждой из 36 функций могут как генерироваться случайным образом, так и вводиться вручную.

**Интерфейс программы**

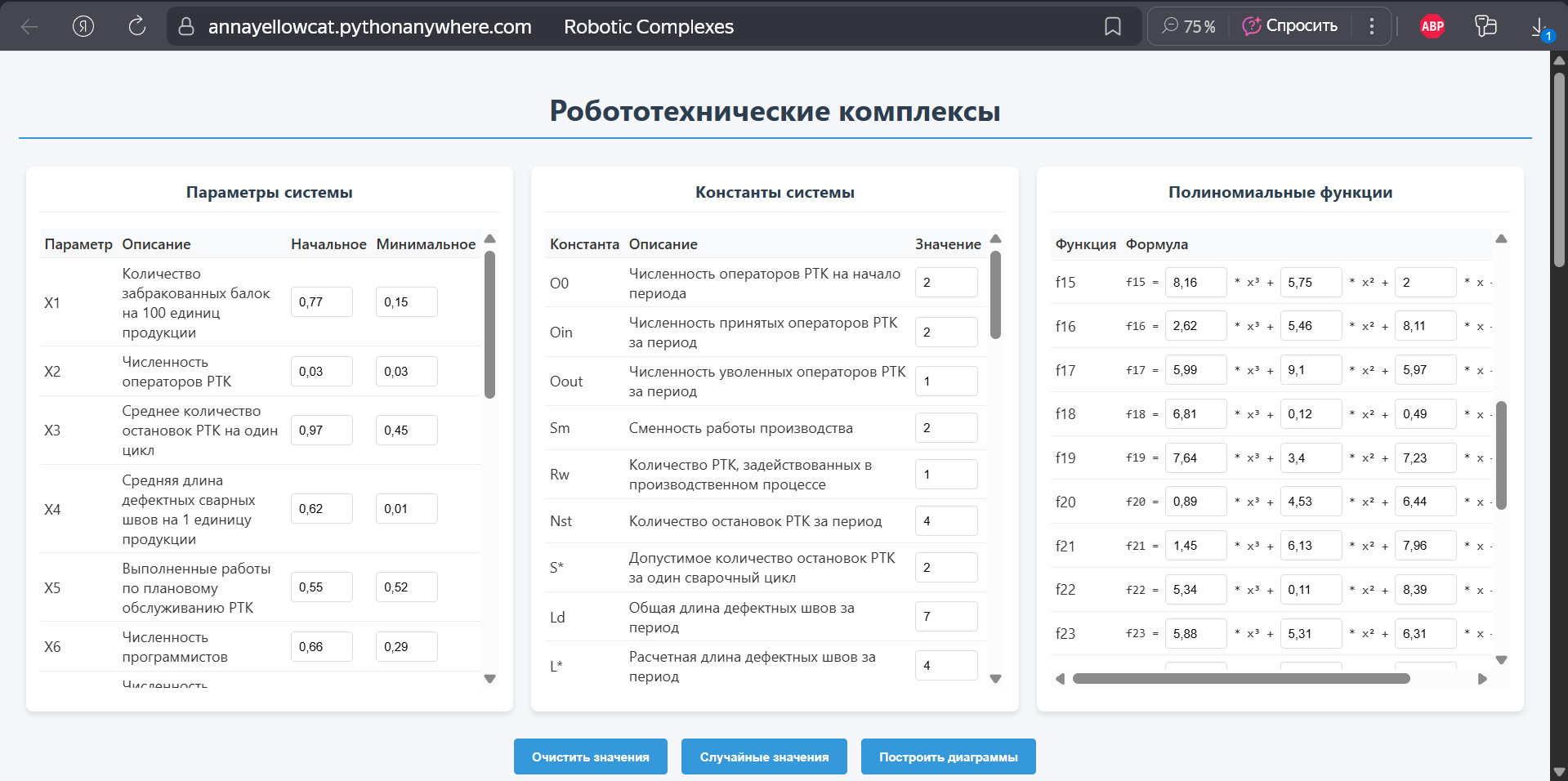
****

Рисунок 1 – Интерфейс системы

В результате программного решения получаем графики системы дифференциальных уравнений.

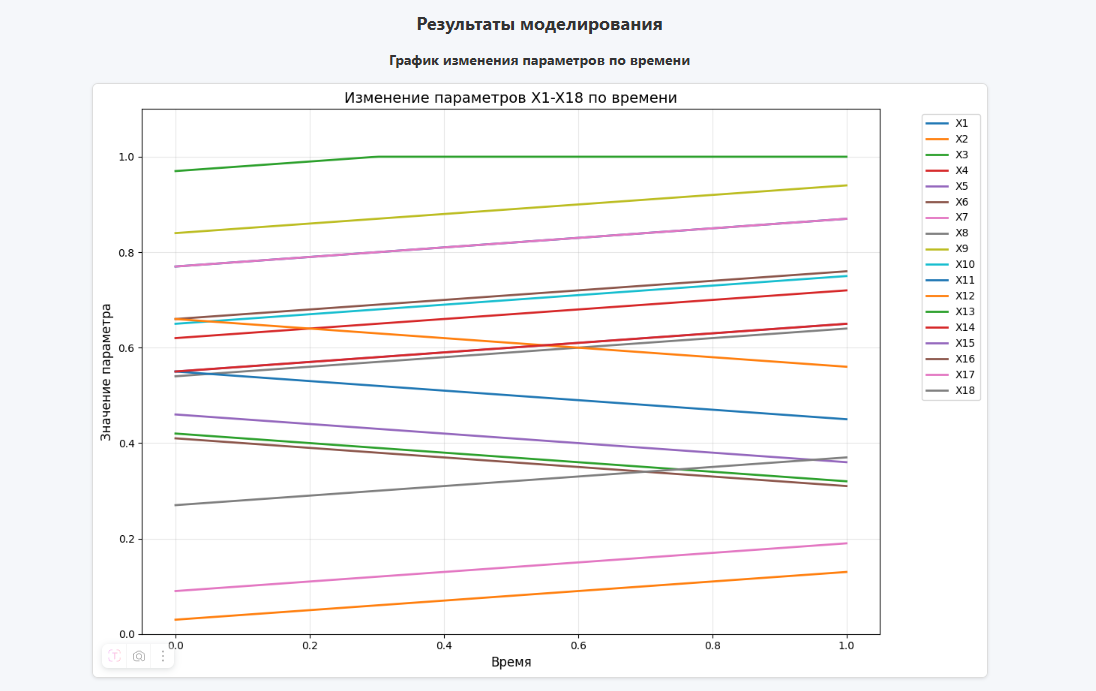


Рисунок 2 – Итоговый график

Далее на рисунке изображены лепестковые диаграммы текущего состояния параметров для t = 0.00, 0.25, 0,49, 0.75, 1.00.



Рисунок 3 – Лепестковые диаграммы

**Программное решение**

Класс Calculator:

import numpy as np

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

from scipy.integrate import solve\_ivp

import random

import os

import io

import base64

class Calculator:

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

# все параметры системы (словарь)

self.parameters = self.generate\_system\_parameters()

# временные точки от 0 до 1 с шагом 0.01

self.time\_points = np.linspace(0, 1, 100)

# переменная для решения системы

self.solution = None

# генерация словаря случайных значений параметров системы

def generate\_system\_parameters(self):

parameters = {}

# параметры X1-X18 (от 0.01 до 1.00)

for i in range(1, 19):

param\_name = f"X{i}"

parameters[param\_name] = round(random.uniform(0.01, 1.0), 2)

parameters[f"{param\_name}\_min"] = round(random.uniform(0.01, parameters[param\_name]), 2)

# константы системы

constants = {

'O0': (1, 5), 'Oin': (1, 3), 'Oout': (1, 3),

'Sm': (1, 2), 'Rw': (1, 5), 'Nst': (1, 10),

'S\_star': (1, 3), 'Ld': (1, 10), 'L\_star': (1, 5),

'Mf': (1, 10), 'Mp': (1, 10), 'P0': (1, 5),

'Pin': (1, 2), 'Pout': (1, 2), 'R0': (1, 5),

'Rin': (1, 2), 'Rout': (1, 2), 'C0': (1, 5),

'Cin': (1, 2), 'Cout': (1, 2), 'T0': (1, 5),

'Tin': (1, 2), 'Tout': (1, 2), 'Nr': (1, 5),

'Df': (1, 10), 'Dp': (1, 5), 'DeltaU': (1, 5),

'Delta\_star\_U': (1, 3), 'DeltaI': (1, 5),

'Delta\_star\_I': (1, 3), 'DeltaT': (1, 5),

'Delta\_star\_T': (1, 3), 'Tdf': (1, 10),

'Tdp': (1, 10), 'DeltaPG': (1, 5),

'Delta\_star\_PG': (1, 3), 'DeltaPV': (1, 5),

'Delta\_star\_PV': (1, 3), 'NTP': (1, 20),

'Nd': (1, 20), 'Ab': (1, 10),

}

for const\_name, (min\_val, max\_val) in constants.items():

parameters[const\_name] = random.randint(min\_val, max\_val)

# коэффициенты для полиномиальных функций f1-f36

for i in range(1, 37):

parameters[f"f{i}\_a3"] = round(random.uniform(0.1, 10.0), 2)

parameters[f"f{i}\_a2"] = round(random.uniform(0.1, 10.0), 2)

parameters[f"f{i}\_a1"] = round(random.uniform(0.1, 10.0), 2)

parameters[f"f{i}\_a0"] = round(random.uniform(0.1, 10.0), 2)

return parameters

# вычисление значения полинома f\_n(x) = a3\*x^3 + a2\*x^2 + a1\*x + a0

# в параметрах (значение икса, n-номер для fn)

def polynomial\_value(self, x, fn\_index):

a3 = self.parameters.get(f"f{fn\_index}\_a3", 0)

a2 = self.parameters.get(f"f{fn\_index}\_a2", 0)

a1 = self.parameters.get(f"f{fn\_index}\_a1", 0)

a0 = self.parameters.get(f"f{fn\_index}\_a0", 0)

# ограничиваем значение x для избежания переполнения

x = np.clip(x, 0, 1) # X всегда от 0 до 1

return a3\*x\*\*3 + a2\*x\*\*2 + a1\*x + a0

# система дифференциальных уравнений

def system\_equations(self, t, X):

try:

dXdt = np.zeros(18)

params = self.parameters

# ограничиваем значения X от 0 до 1

X = np.clip(X, 0, 1)

# сами функции производных для Х1-Х18,

# параметр Хn указывается с индексом на 1 меньше, чем на картинке

# номер функции f соответствует картинке

dXdt[0] = params['Rw'] \* self.polynomial\_value(X[2], 1) \* self.polynomial\_value(X[10], 2) \* \

self.polynomial\_value(X[11], 3) \* self.polynomial\_value(X[12], 4) - \

params['Nst'] \* self.polynomial\_value(X[1], 5) \* self.polynomial\_value(X[7], 6) \* \

self.polynomial\_value(X[16], 7)

dXdt[1] = (params['O0'] + params['Oin']) \* self.polynomial\_value(X[16], 12) - \

(params['Sm'] + params['Rw'] + params['Oout'])

dXdt[2] = params['Nst'] / max(params['Rw'], 0.001) \* self.polynomial\_value(X[9], 8) \* \

self.polynomial\_value(X[14], 9) \* self.polynomial\_value(X[15], 10) - \

params['S\_star'] \* self.polynomial\_value(X[1], 11)

dXdt[3] = params['Ld'] \* self.polynomial\_value(X[14], 13) \* self.polynomial\_value(X[15], 14) - \

params['L\_star'] \* self.polynomial\_value(X[1], 15)

dXdt[4] = params['Mf'] \* self.polynomial\_value(X[5], 16) \* self.polynomial\_value(X[6], 17) - \

params['Mp'] \* self.polynomial\_value(X[9], 18)

dXdt[5] = (params['P0'] + params['Pin']) \* self.polynomial\_value(X[16], 19) - \

(params['Sm'] + params['Rw'] + params['Pout'])

dXdt[6] = (params['R0'] + params['Rin']) \* self.polynomial\_value(X[16], 20) - \

(params['Sm'] + params['Rw'] + params['Rout'])

dXdt[7] = (params['C0'] + params['Cin']) \* self.polynomial\_value(X[16], 21) - \

(params['Sm'] + params['Rw'] + params['Cout'])

dXdt[8] = (params['T0'] + params['Tin']) \* self.polynomial\_value(X[16], 22) - params['Tout']

dXdt[9] = (params['Nr'] + params['Df']) \* self.polynomial\_value(X[16], 23) - params['Dp']

dXdt[10] = params['DeltaU'] - params['Delta\_star\_U'] \* self.polynomial\_value(X[4], 24)

dXdt[11] = params['DeltaI'] - params['Delta\_star\_I'] \* self.polynomial\_value(X[4], 25)

dXdt[12] = params['DeltaT'] - params['Delta\_star\_T'] \* self.polynomial\_value(X[4], 26)

dXdt[13] = params['Tdf'] \* self.polynomial\_value(X[8], 27) - params['Tdp']

dXdt[14] = params['DeltaPG'] - params['Delta\_star\_PG'] \* self.polynomial\_value(X[16], 28)

dXdt[15] = params['DeltaPV'] - params['Delta\_star\_PV'] \* self.polynomial\_value(X[16], 29)

dXdt[16] = params['NTP'] \* self.polynomial\_value(X[8], 30) - params['Rw']

dXdt[17] = params['Nd'] \* self.polynomial\_value(X[5], 31) \* self.polynomial\_value(X[6], 32) \* \

self.polynomial\_value(X[7], 33) \* self.polynomial\_value(X[13], 34) - \

(params['Ab'] + params['Ld']) \* self.polynomial\_value(X[0], 35) \* self.polynomial\_value(X[3], 36)

# ограничиваем производные для стабильности и положительных значений

dXdt = np.clip(dXdt, -0.1, 0.1)

return dXdt

except Exception as e:

print(f"Ошибка в вычислении производных: {e}")

return np.zeros(18)

# решение системы дифференциальных уравнений

def solve\_system(self):

try:

# начальные условия из параметров, список значени Х1-Х18 на момент времени 0

X0 = [self.parameters[f"X{i}"] for i in range(1, 19)]

# решение системы

self.solution = solve\_ivp(

self.system\_equations,

[0, 1],

X0,

t\_eval=self.time\_points,

method='RK45',

rtol=1e-3,

atol=1e-3

)

# срез значений по оси y от 0 до 1

self.solution.y = np.clip(self.solution.y, 0, 1)

return self.solution

except Exception as e:

print(f"Ошибка при решении системы: {e}")

# Возвращаем фиктивное решение для отладки

self.solution = type('obj', (object,), {

't': self.time\_points,

'y': np.random.uniform(0, 1, (18, len(self.time\_points)))

})

return self.solution

# отрисовка графика изменений Х1-Х18 по времени

def plot\_time\_series(self):

if self.solution is None:

self.solve\_system()

fig = plt.figure(figsize=(12, 8))

for i in range(18):

plt.plot(self.solution.t, self.solution.y[i], label=f'X{i+1}', linewidth=2)

plt.xlabel('Время', fontsize=12)

plt.ylabel('Значение параметра', fontsize=12)

plt.title('Изменение параметров X1-X18 по времени', fontsize=14)

plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left')

plt.grid(True, alpha=0.3)

plt.ylim(0, 1.1) # Ограничиваем ось Y от 0 до 1

plt.tight\_layout()

# сохраняем в буфер

time\_series\_buffer = io.BytesIO()

plt.savefig(time\_series\_buffer, format='png', dpi=100, bbox\_inches='tight')

time\_series\_buffer.seek(0)

time\_series\_b64 = base64.b64encode(time\_series\_buffer.getvalue()).decode()

plt.close(fig)

return time\_series\_b64

# отрисовка 5 лепестковых диаграмм

def plot\_radar\_charts(self):

if self.solution is None:

self.solve\_system()

time\_points = [0, 0.25, 0.5, 0.75, 1]

time\_indices = [np.abs(self.solution.t - t).argmin() for t in time\_points]

categories = [f'X{i+1}' for i in range(18)]

N = len(categories)

angles = [n / float(N) \* 2 \* np.pi for n in range(N)]

angles += angles[:1]

fig, axes = plt.subplots(2, 3, figsize=(15, 10), subplot\_kw=dict(polar=True))

axes = axes.flatten()

colors = plt.cm.viridis(np.linspace(0, 1, len(time\_points)))

for i, (t\_idx, ax) in enumerate(zip(time\_indices, axes)):

if i >= len(time\_points):

break

values = self.solution.y[:, t\_idx].tolist()

values += values[:1]

ax.set\_theta\_offset(np.pi / 2)

ax.set\_theta\_direction(-1)

ax.plot(angles, values, color=colors[i], linewidth=2, linestyle='solid')

ax.fill(angles, values, color=colors[i], alpha=0.25)

ax.set\_xticks(angles[:-1])

ax.set\_xticklabels(categories, fontsize=8)

ax.set\_rlabel\_position(0)

ax.set\_yticks([0.2, 0.4, 0.6, 0.8, 1.0])

ax.set\_yticklabels(["0.2", "0.4", "0.6", "0.8", "1.0"], color="grey", size=8)

ax.set\_ylim(0, 1)

ax.set\_title(f'Время t = {self.solution.t[t\_idx]:.2f}', size=11, color=colors[i], pad=10)

for i in range(len(time\_points), len(axes)):

fig.delaxes(axes[i])

plt.suptitle('Лепестковые диаграммы параметров X1-X18', fontsize=14)

plt.tight\_layout()

radar\_buffer = io.BytesIO()

plt.savefig(radar\_buffer, format='png', dpi=100, bbox\_inches='tight')

radar\_buffer.seek(0)

radar\_b64 = base64.b64encode(radar\_buffer.getvalue()).decode()

plt.close(fig)

return radar\_b64

# построение всех графиков

def plot\_all\_results(self):

self.solve\_system()

time\_series\_b64 = self.plot\_time\_series()

radar\_b64 = self.plot\_radar\_charts()

return [time\_series\_b64, radar\_b64]

Веб-приложение:

from flask import Flask, request, render\_template, jsonify, send\_file, session

import os

import io

import base64

from calculator import Calculator

app = Flask(\_\_name\_\_)

CALCULATOR = None

# инициализация калькулятора со случайными значениями

def init\_calculator():

global CALCULATOR

CALCULATOR = Calculator()

return CALCULATOR.parameters

# обновление параметров калькулятора из данных формы

def update\_calculator\_from\_form(form\_data):

global CALCULATOR

CALCULATOR.parameters = {}

for key, value in form\_data.items():

if value and value != "":

try:

if "." in value:

CALCULATOR.parameters[key] = float(value)

else:

CALCULATOR.parameters[key] = int(value)

except ValueError:

CALCULATOR.parameters[key] = value

return CALCULATOR.parameters

# главная страница и внедрение модели калькулятора

@app.route("/")

def index():

global CALCULATOR

if CALCULATOR is None:

parameters = init\_calculator()

else:

parameters = CALCULATOR.parameters

return render\_template("index.html", parameters=parameters)

# очистка всех значений

@app.route("/clear", methods=["POST"])

def clear\_values():

global CALCULATOR

CALCULATOR = None

return jsonify({"status": "success"})

# случайные значения

@app.route("/random", methods=["POST"])

def random\_values():

parameters = init\_calculator()

return jsonify({"status": "success", "parameters": parameters})

# построение диаграмм

@app.route("/plot", methods=["POST"])

def plot\_diagrams():

global CALCULATOR

form\_data = request.json

# проверка на заполненность всех полей

empty\_fields = []

for key, value in form\_data.items():

if value == "":

empty\_fields.append(key)

if empty\_fields:

return jsonify(

{

"status": "error",

"message": "Не все поля заполнены",

"empty\_fields": empty\_fields,

}

)

# обновляем калькулятор из формы

update\_calculator\_from\_form(form\_data)

# создаем картинки диаграмм и выводим их

try:

time\_series\_b64, radar\_b64 = CALCULATOR.plot\_all\_results()

return jsonify(

{

"status": "success",

"time\_series": time\_series\_b64,

"radar\_charts": radar\_b64,

}

)

except Exception as e:

return jsonify(

{"status": "error", "message": f"Ошибка при построении графиков: {str(e)}"}

)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(port=8000, debug=True)

**Вывод**

В ходе работы была успешно разработана и реализована информационная система для прогнозирования критических отказов в роботизированных сварочных комплексах. Созданная математическая модель, основанная на системе из 18 дифференциальных уравнений, позволяет анализировать динамику ключевых параметров технологического процесса. Веб-интерфейс системы предоставляет удобные инструменты для визуализации результатов моделирования в виде временных рядов и лепестковых диаграмм, что позволяет оперативно оценивать вероятность возникновения дефектов и принимать меры по их предотвращению. Цели данной работы были достигнуты.