ДОДАТОК А

Лістинг коду

```
import numpy as np
from bson import ObjectId
import plotly.graph_objs as go
import dash
from dash import Dash, dcc, html, Input, Output, State,
ALL
from gray_wolf_algorithm import CGWOAlgorithm,
ObjectiveFunction, create_figure
from datetime import datetime
from pymongo import MongoClient
import json

# MongoDB setup
client = MongoClient("mongodb://localhost:27017/")
db = client['gwo_database']
params_collection = db['params']
functions_collection = db['functions']
history_collection = db['history']

# Placeholder for algorithm parameters and history
algorithm_parameters = {'iterations': 10, 'wolves': 10,
'param_variations': [], 'objective_functions': []}
history = []


# Load saved data from MongoDB
def load_data():
    global algorithm_parameters, history
    algorithm_parameters['param_variations'] =
list(params_collection.find())
    algorithm_parameters['objective_functions'] =
list(functions_collection.find())
    history.extend(list(history_collection.find()))

    # Add default parameter variations if none are found
in the database
    if not algorithm_parameters['param_variations']:
        default_variations = [
            {'name': 'GWO no crazy', 'y': 0, 'd': 0,
'sigma2': 0},
            {'name': 'default1', 'y': 0.5, 'd': 0.1,
'sigma2': 0.5},
            {'name': 'default2', 'y': 0.4, 'd': 0.2,
'sigma2': 0.3}
        ]

algorithm_parameters['param_variations'].extend(default_variat
ions)
        params_collection.insert_many(default_variations)
        # Add default functions if none are found in the
```

```
database
        if not algorithm_parameters['objective_functions']:
            default_functions = [
                {'name': 'Sphere', 'code': 'x**2 + y**2',
'lb': 0, 'ub': 50},
                {'name': 'Rosenbrock', 'code': '100*(y-
x**2)**2 + (1-x)**2', 'lb': 0, 'ub': 80000},
                {'name': 'Rastrigin', 'code': '10*2 + (x**2 -
10*np.cos(2*np.pi*x)) + (y**2 - 10*np.cos(2*np.pi*y))',
                 'lb': 0, 'ub': 80},
                {'name': 'Alkley',
                 'code': '-20 * np.exp(-0.2 * np.sqrt(0.5 * (x
** 2 + y ** 2))) - np.exp(0.5 * (np.cos(2 * np.pi * x) +
np.cos(2 * np.pi * y))) + 20 + np.e',
                 'lb': 0, 'ub': 15}

            ]

algorithm_parameters['objective_functions'].extend(default_fun
ctions)

functions_collection.insert_many(default_functions)


    load_data()

    app = Dash(__name__)
    app.config.suppress_callback_exceptions = True

    app.layout = html.Div([
        dcc.Tabs(id='tabs', value='tab-1', children=[
            dcc.Tab(label='Test Algorithm', value='tab-1',
style={'backgroundColor': '#333333', 'color': '#FFFFFF'}),
            dcc.Tab(label='Create Algorithm', value='tab-2',
style={'backgroundColor': '#333333', 'color': '#FFFFFF'}),
            dcc.Tab(label='Evolution', value='tab-3',
style={'backgroundColor': '#333333', 'color': '#FFFFFF'}),
            dcc.Tab(label='Create Function', value='tab-4',
                    style={'backgroundColor': '#333333',
'color': '#FFFFFF'}),
            dcc.Tab(label='History', value='tab-5',
style={'backgroundColor': '#333333', 'color': '#FFFFFF'}),
            dcc.Tab(label='Compare Results', value='tab-6',
style={'backgroundColor': '#333333', 'color': '#FFFFFF'}),
        ], style={'backgroundColor': '#2A2A2A', 'color':
'#FFFFFF'}),
        html.Div(id='tabs-content', style={'backgroundColor':
'#1E1E1E', 'height': '100%', 'padding': '20px',
                                           'font-family':
'Arial, sans-serif'})

    ])
```

```
    @app.callback(
        Output('tabs-content', 'children'),
        Input('tabs', 'value')
    )
    def render_content(tab):
        if tab == 'tab-1':
            return html.Div([
                html.Div([
                    html.Label('Iterations:', style={'color':
'#FFFFFF', 'font-size': '16px', 'margin-right': '10px'}),
                    dcc.Input(id='iterations', type='number',
value=algorithm_parameters['iterations'], min=1,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-right': '20px'}),
                    html.Label('Number of Wolves:',
                               style={'color': '#FFFFFF',
'font-size': '16px', 'margin-right': '10px'}),
                    dcc.Input(id='wolves', type='number',
value=algorithm_parameters['wolves'], min=1,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-right': '20px'}),
                    html.Label('Param Variation:',
style={'color': '#FFFFFF', 'font-size': '16px', 'margin-
right': '10px'}),
                    dcc.Dropdown(id='param-variation-
dropdown',
                                 options=[{'label':
param['name'], 'value': param['name']} for param in

algorithm_parameters['param_variations']],

value=algorithm_parameters['param_variations'][0]['name'] if
algorithm_parameters[
                                     'param_variations'] else
None,
                                 style={'backgroundColor':
'#CCCCCC', 'color': '#2A2A2A', 'border': 'none',
                                        'margin-right':
'20px', 'font-size': '16px', 'width': '300px'}),
                    html.Label('Select Function:',
style={'color': '#FFFFFF', 'font-size': '16px', 'margin-
right': '10px'}),
                    dcc.Dropdown(id='function-dropdown',
                                 options=[{'label':
func['name'], 'value': func['name']} for func in

algorithm_parameters['objective_functions'] if 'name' in
func],
```

```
                                    style={'backgroundColor':
'#CCCCCC', 'color': '#2A2A2A', 'border': 'none',
                                           'padding': '5px',
'margin-right': '20px', 'font-size': '16px', 'width':
'300px'}),
                    html.Button('Run', id='run-button',
n_clicks=0,
                                    style={'backgroundColor':
'#4CAF50', 'color': 'white', 'border': 'none',
                                           'padding': '10px 20px',
'cursor': 'pointer', 'font-size': '16px'})

                ], style={'display': 'flex', 'align-items':
'center', 'backgroundColor': '#333333', 'padding': '10px',
                          'border-radius': '5px', 'margin-
bottom': '20px'}),
                html.Div(id='selected-param-details',
                         style={'color': '#FFFFFF',
'marginTop': '10px', 'fontSize': '16px'}),
                html.Div([
                    html.Div([
                        html.H3('Wolves\' Coordinates',
style={'color': '#FFFFFF'}),
                        html.Ul(id='wolf-list',
style={'backgroundColor': '#2A2A2A', 'color': '#FFFFFF',
'padding': '10px',

'border-radius': '5px'})
                    ], style={'flex': '1', 'overflow': 'auto',
'height': '80vh', 'margin-right': '20px',
                              'backgroundColor': '#333333',
'padding': '10px', 'border-radius': '5px'}),
                    dcc.Graph(id='optimization-plot',
                              style={'flex': '3',
'backgroundColor': '#2A2A2A', 'border-radius': '5px'})
                ], style={'display': 'flex',
'backgroundColor': '#333333', 'padding': '10px', 'border-
radius': '5px'}),
                html.Div(id='stats',
                         style={'margin-top': '20px',
'backgroundColor': '#333333', 'color': '#FFFFFF', 'padding':
'10px',
                                'border-radius': '5px'})
            ])
        elif tab == 'tab-2':
            return html.Div([
                html.H3('Create Algorithm', style={'color':
'#FFFFFF'}),
                html.Div([
                    html.Label('Name:', style={'color':
'#FFFFFF', 'font-size': '16px', 'margin-right': '10px'}),
                    dcc.Input(id='param-name', type='text',
                              style={'backgroundColor':
```

```
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                        'margin-right': '20px'}),
                    html.Label('y:', style={'color':
'#FFFFFF', 'font-size': '16px', 'margin-right': '10px'}),
                    dcc.Input(id='y', type='number',
value=0.5, min=0,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                        'margin-right': '20px'}),
                    html.Label('d:', style={'color':
'#FFFFFF', 'font-size': '16px', 'margin-right': '10px'}),
                    dcc.Input(id='d', type='number',
value=0.1, min=0,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                        'margin-right': '20px'}),
                    html.Label('sigma2:', style={'color':
'#FFFFFF', 'font-size': '16px', 'margin-right': '10px'}),
                    dcc.Input(id='sigma2', type='number',
value=0.5, min=0,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                        'margin-right': '20px'}),
                    html.Button('Save', id='save-algorithm-
button', n_clicks=0,
                                style={'backgroundColor':
'#4CAF50', 'color': 'white', 'border': 'none',
                                       'padding': '10px 20px',
'cursor': 'pointer', 'font-size': '16px'})
                ], style={'display': 'flex', 'align-items':
'center', 'backgroundColor': '#333333', 'padding': '10px',
                          'border-radius': '5px', 'margin-
bottom': '20px'}),
                html.H3('Current Parameter Variations',
style={'color': '#FFFFFF'}),
                html.Ul(id='param-variation-list', children=[
                    html.Li([
                        html.Span(f"{param['name']}
(y={param['y']}, d={param['d']}, sigma2={param['sigma2']})",
                                  style={'color': '#FFFFFF'}),
                        html.Button('Delete', id={'type':
'delete-param-button', 'index': i}, n_clicks=0,
                                    style={'backgroundColor':
'#FF4136', 'color': 'white', 'border': 'none',
                                           'padding': '5px
10px', 'cursor': 'pointer', 'margin-left': '10px'})
                    ], style={'display': 'flex', 'align-
items': 'center', 'margin-bottom': '10px'})
                    for i, param in
```

```
enumerate(algorithm_parameters['param_variations'])
                ], style={'backgroundColor': '#2A2A2A',
'color': '#FFFFFF', 'padding': '10px', 'border-radius':
'5px'})
            ])
        elif tab == 'tab-3':
            return html.Div([
                html.H3('Evolutionary Optimization',
style={'color': '#FFFFFF'}),
                html.Div([
                    html.Label('Population Size:',
                               style={'color': '#FFFFFF',
'font-size': '16px', 'margin-bottom': '10px'}),
                    dcc.Input(id='evolution-population-size',
type='number', value=10, min=1,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-bottom':
'20px'}),
                    html.Label('Number of Iterations:',
                               style={'color': '#FFFFFF',
'font-size': '16px', 'margin-bottom': '10px'}),
                    dcc.Input(id='evolution-iterations',
type='number', value=10, min=1,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-bottom':
'20px'}),
                    html.Label('Number of Evolution
Iterations:',
                               style={'color': '#FFFFFF',
'font-size': '16px', 'margin-bottom': '10px'}),
                    dcc.Input(id='evolution-evolution-
iterations', type='number', value=10, min=1,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-bottom':
'20px'}),
                    html.Label('Number of Wolf Populations:',
                               style={'color': '#FFFFFF',
'font-size': '16px', 'margin-bottom': '10px'}),
                    dcc.Input(id='evolution-wolf-populations',
type='number', value=5, min=1,
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-bottom':
'20px'}),
                    html.Label('Sigma Limits:',
style={'color': '#FFFFFF', 'font-size': '16px', 'margin-
```

```
bottom': '10px'}),
                    dcc.Input(id='sigma-limits', type='text',
value='0.1,1',  # Format: min,max
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-bottom':
'20px'}),
                    html.Label('Y Limits:', style={'color':
'#FFFFFF', 'font-size': '16px', 'margin-bottom': '10px'}),
                    dcc.Input(id='y-limits', type='text',
value='0.1,1',  # Format: min,max
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-bottom':
'20px'}),
                    html.Label('D Limits:', style={'color':
'#FFFFFF', 'font-size': '16px', 'margin-bottom': '10px'}),
                    dcc.Input(id='d-limits', type='text',
value='0.1,1',  # Format: min,max
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-bottom':
'20px'}),
                ], style={'display': 'flex', 'flex-direction':
'column', 'backgroundColor': '#333333', 'padding': '10px',
                          'border-radius': '5px', 'margin-
bottom': '20px'}),
                html.Div([
                    html.Label('Select Function:',
                               style={'color': '#FFFFFF',
'font-size': '16px', 'margin-bottom': '10px'}),
                    dcc.Dropdown(id='evolution-function-
dropdown',
                                 options=[{'label':
func['name'], 'value': func['name']} for func in

algorithm_parameters['objective_functions'] if 'name' in
func],
                                 style={'backgroundColor':
'#CCCCCC', 'color': '#2A2A2A', 'border': 'none',
                                        'padding': '5px',
'margin-bottom': '20px', 'font-size': '16px', 'width':
'300px'}),
                    html.Button('Evolute', id='evolute-
button', n_clicks=0,
                                style={'backgroundColor':
'#4CAF50', 'color': 'white', 'border': 'none',
                                       'padding': '10px 20px',
'cursor': 'pointer', 'font-size': '16px',
                                       'margin-bottom':
```

```
'20px'})
                ], style={'display': 'flex', 'flex-direction':
'column', 'backgroundColor': '#333333', 'padding': '10px',
                          'border-radius': '5px', 'margin-
bottom': '20px'}),
                html.Div(id='evolution-progress',
                         style={'color': '#FFFFFF',
'marginTop': '10px', 'fontSize': '16px'}),
                html.H3('Best Evoluted Populations',
style={'color': '#FFFFFF', 'marginTop': '20px'}),
                html.Ul(id='evolution-result-list',
                        style={'backgroundColor': '#2A2A2A',
'color': '#FFFFFF', 'padding': '10px',
                               'border-radius': '5px'}),
                html.Div('Best parameters are saved',
id='save-best-message',
                         style={'color': '#4CAF50', 'font-
size': '16px', 'margin-top': '20px'}),
                html.H3('Current Parameter Variations',
style={'color': '#FFFFFF'}),
                html.Ul(id='param-variation-list',
                        style={'backgroundColor': '#2A2A2A',
'color': '#FFFFFF', 'padding': '10px',
                               'border-radius': '5px'})
            ])

        elif tab == 'tab-4':
            return html.Div([
                html.H3('Create Function', style={'color':
'#FFFFFF'}),
                html.Div([
                    html.Label('Function Name:',
style={'color': '#FFFFFF', 'font-size': '16px', 'margin-
right': '10px'}),
                    dcc.Input(id='function-name', type='text',
                              style={'backgroundColor':
'#2A2A2A', 'color': '#FFFFFF', 'border': 'none', 'padding':
'5px',
                                     'margin-right': '20px'}),
                ], style={'margin-bottom': '20px'}),
                dcc.Textarea(id='objective-function',
                             placeholder='Enter the return
expression, e.g., np.sin(np.sqrt(x**2 + y**2))',
                             style={'width': '100%', 'height':
'200px', 'backgroundColor': '#2A2A2A', 'color': '#FFFFFF',
                                    'border': 'none',
'padding': '10px', 'margin-bottom': '20px'}),
                html.Label('Lower Bound (lb):',
style={'color': '#FFFFFF', 'font-size': '16px', 'margin-
right': '10px'}),
                dcc.Input(id='lb', type='number', value=-5,
                          style={'backgroundColor': '#2A2A2A',
'color': '#FFFFFF', 'border': 'none', 'padding': '5px',
```

```
                                    'margin-right': '20px'}),
                html.Label('Upper Bound (ub):',
style={'color': '#FFFFFF', 'font-size': '16px', 'margin-
right': '10px'}),
                dcc.Input(id='ub', type='number', value=5,
                          style={'backgroundColor': '#2A2A2A',
'color': '#FFFFFF', 'border': 'none', 'padding': '5px',
                                 'margin-right': '20px'}),
                html.Button('Render', id='render-function-
button', n_clicks=0,
                            style={'backgroundColor':
'#4CAF50', 'color': 'white', 'border': 'none', 'padding':
'10px 20px',
                                   'cursor': 'pointer', 'font-
size': '16px', 'margin-right': '20px'}),
                html.Button('Save', id='save-function-button',
n_clicks=0,
                            style={'backgroundColor':
'#4CAF50', 'color': 'white', 'border': 'none', 'padding':
'10px 20px',
                                   'cursor': 'pointer', 'font-
size': '16px'}),
                dcc.Graph(id='function-plot',
                          style={'margin-top': '20px',
'backgroundColor': '#2A2A2A', 'border-radius': '5px'}),
                html.H3('Current Functions', style={'color':
'#FFFFFF'}),
                html.Ul(id='function-list', children=[
                    html.Li([
                        html.Span(f"{func['name']} (code:
{func['code']})", style={'color': '#FFFFFF'}),
                        html.Button('Delete', id={'type':
'delete-function-button', 'index': i}, n_clicks=0,
                                    style={'backgroundColor':
'#FF4136', 'color': 'white', 'border': 'none',
                                           'padding': '5px
10px', 'cursor': 'pointer', 'margin-left': '10px'})
                    ], style={'display': 'flex', 'align-
items': 'center', 'margin-bottom': '10px'})
                    for i, func in
enumerate(algorithm_parameters['objective_functions']) if
'name' in func
                ], style={'backgroundColor': '#2A2A2A',
'color': '#FFFFFF', 'padding': '10px', 'border-radius':
'5px'})
            ])
        elif tab == 'tab-5':
            return html.Div([
                html.H3('History', style={'color':
'#FFFFFF'}),
                html.Ul(id='history-list', children=[
                    html.Li(
                        [
```

```
                            html.Div(entry['timestamp'],
style={'font-weight': 'bold'}),
                            html.Div(f"Alpha Score:
{entry['alpha_score']:.4f}"),
                            html.Div(f"Beta Score:
{entry['beta_score']:.4f}"),
                            html.Div(f"Delta Score:
{entry['delta_score']:.4f}"),
                            html.Div(f"Iterations:
{entry['iterations']}"),
                            html.Div(f"Wolves:
{entry['wolves']}"),
                            html.Div(f"Param Variation:
{entry['param_variation']}")
                        ],
                        style={'margin-bottom': '20px',
'padding': '10px', 'border': '1px solid #FFFFFF',
                               'border-radius': '5px'}
                    )
                    for entry in reversed(history)
                ], style={'backgroundColor': '#2A2A2A',
'color': '#FFFFFF', 'padding': '10px', 'border-radius':
'5px'})
            ])
        elif tab == 'tab-6':
            return html.Div([
                html.H3('Compare Results', style={'color':
'#FFFFFF'}),
                dcc.Dropdown(id='compare-dropdown',
                             options=[{'label':
f"{entry['timestamp']}", 'value': i} for i, entry in

enumerate(reversed(history))],
                             multi=True,
                             style={'backgroundColor':
'#2A2A2A', 'color': '#000000', 'border': 'none', 'padding':
'5px',
                                    'font-size': '16px',
'width': '300px'}),
                html.Button('Compare', id='compare-button',
n_clicks=0,
                            style={'backgroundColor':
'#4CAF50', 'color': 'white', 'border': 'none', 'padding':
'10px 20px',
                                   'cursor': 'pointer', 'font-
size': '16px', 'margin-top': '20px'}),
                dcc.Graph(id='compare-plot',
                          style={'margin-top': '20px',
'backgroundColor': '#2A2A2A', 'border-radius': '5px'})
            ])


    @app.callback(
```

```
        Output('optimization-plot', 'figure'),
        Output('wolf-list', 'children'),
        Output('stats', 'children'),
        Output('selected-param-details', 'children'),
        Input('run-button', 'n_clicks'),
        Input('param-variation-dropdown', 'value'),
        Input('function-dropdown', 'value'),
        State('iterations', 'value'),
        State('wolves', 'value')
    )
    def update_graph_and_display_details(n_clicks,
param_variation, selected_function, n_iterations, n_wolves):
        ctx = dash.callback_context
        if not ctx.triggered:
            return go.Figure(), [], "", ""

        triggered_id =
ctx.triggered[0]['prop_id'].split('.')[0]

        # Get the selected function details from MongoDB
        selected_function_details = next(
            (func for func in
algorithm_parameters['objective_functions'] if func['name'] ==
selected_function),
            None)

        if not selected_function_details:
            return go.Figure(), [], "", "No valid function
selected."

        selected_function_code =
selected_function_details['code']
        lb = selected_function_details['lb']
        ub = selected_function_details['ub']

        function_code = f"""
    import numpy as np

    def objective_function(x, y):
        return {selected_function_code}
    """
        local_env = {}
        exec(function_code, globals(), local_env)
        if 'objective_function' not in local_env:
            raise ValueError("The function
'objective_function' is not defined in the provided code.")

        objective = ObjectiveFunction(lb, ub,
func=local_env['objective_function'])
        x, y = objective.get_meshgrid()
        z = objective.objective_function(x, y)

        fig = go.Figure(data=[go.Surface(z=z, x=x, y=y,
```

```
colorscale='Viridis')])

        if triggered_id == 'param-variation-dropdown' or
triggered_id == 'function-dropdown':
            selected_param = next(
                (param for param in
algorithm_parameters['param_variations'] if param['name'] ==
param_variation), None)

            param_details = (
                f"Selected: {selected_param['name']} with y =
{selected_param['y']}, d = {selected_param['d']}, sigma2 =
{selected_param['sigma2']}"
                if selected_param else "No parameter variation
selected.")
            return fig, [], "", param_details

        if triggered_id == 'run-button':
            if n_clicks == 0:
                return go.Figure(), [], "", ""

            algorithm =
CGWOAlgorithm(objective.objective_function,
fitness_parameter=-1)

            for param in
algorithm_parameters['param_variations']:
                algorithm.add_params_variation(param['name'],
param['y'], param['d'], param['sigma2'])

            all_positions, all_stats, alpha_score, beta_score,
delta_score = algorithm.optimize(n_iterations, n_wolves, 2,

param_variation)

            fig = create_figure(all_positions, objective, x,
y)

            wolf_coords = [html.Li(f"Wolf {i + 1}: {pos}") for
i, pos in enumerate(all_positions[-1])]
            stats_text = f"Best Alpha Score:
{alpha_score:.4f}, Best Beta Score: {beta_score:.4f}, Best
Delta Score: {delta_score:.4f}"

            # Convert numpy arrays to lists before inserting
into MongoDB
            history_entry = {
                'timestamp': datetime.now().strftime('%Y-%m-%d
%H:%M:%S'),
                'alpha_score': alpha_score,
                'beta_score': beta_score,
                'delta_score': delta_score,
                'iterations': n_iterations,
```

```
                'wolves': n_wolves,
                'param_variation': param_variation,
                'positions': [pos.tolist() for pos in
all_positions]
            }
            history.append(history_entry)
            history_collection.insert_one(history_entry)

            return fig, wolf_coords, stats_text, ""


    @app.callback(
        Output('history-list', 'children'),
        Input('tabs', 'value')
    )
    def update_history(tab):
        if tab == 'tab-5':
            return [
                html.Li(
                    [
                        html.Div(entry['timestamp'],
style={'font-weight': 'bold'}),
                        html.Div(f"Alpha Score:
{entry['alpha_score']:.4f}"),
                        html.Div(f"Beta Score:
{entry['beta_score']:.4f}"),
                        html.Div(f"Delta Score:
{entry['delta_score']:.4f}"),
                        html.Div(f"Iterations:
{entry['iterations']}"),
                        html.Div(f"Wolves:
{entry['wolves']}"),
                        html.Div(f"Param Variation:
{entry['param_variation']}")
                    ],
                    style={'margin-bottom': '20px', 'padding':
'10px', 'border': '1px solid #FFFFFF',
                           'border-radius': '5px'}
                )
                for entry in reversed(history)
            ]
        return []


    @app.callback(
        Output('compare-plot', 'figure'),
        Input('compare-button', 'n_clicks'),
        State('compare-dropdown', 'value')
    )
    def compare_results(n_clicks, selected_indices):
        if n_clicks == 0 or not selected_indices:
            return go.Figure()
```

```
        selected_entries = [history[i] for i in
selected_indices]
        fig = go.Figure()

        for entry in selected_entries:
            fig.add_trace(go.Scatter(
                x=[i for i in range(len(entry['positions']))],
                y=[pos[0][0] for pos in entry['positions']],
                mode='lines',
                name=f"Alpha {entry['timestamp']}"
            ))

        fig.update_layout(title='Comparison of Alpha Wolf
Positions Over Iterations')

        return fig


    @app.callback(
        Output('param-name', 'value'),
        Output('y', 'value'),
        Output('d', 'value'),
        Output('sigma2', 'value'),
        Input('save-algorithm-button', 'n_clicks'),
        State('param-name', 'value'),
        State('y', 'value'),
        State('d', 'value'),
        State('sigma2', 'value')
    )
    def save_algorithm_params(n_clicks, name, y, d, sigma2):
        if n_clicks > 0:
            param_entry = {'name': name, 'y': y, 'd': d,
'sigma2': sigma2}

algorithm_parameters['param_variations'].append(param_entry)
            params_collection.insert_one(param_entry)
        return name, y, d, sigma2


    @app.callback(
        Output('function-plot', 'figure'),
        Input('render-function-button', 'n_clicks'),
        State('objective-function', 'value'),
        State('lb', 'value'),
        State('ub', 'value')
    )
    def render_custom_function(n_clicks, return_expression,
lb, ub):
        if n_clicks == 0 or not return_expression:
            return go.Figure()
        try:
            # Wrap the return expression in a full function
definition
```

```
            function_code = f"""
    import numpy as np

    def objective_function(x, y):
        return {return_expression}
    """
            # Define a dictionary to safely execute the code
and retrieve the objective_function
            local_env = {}
            exec(function_code, globals(), local_env)

            # Ensure the objective_function is defined in the
code
            if 'objective_function' not in local_env:
                raise ValueError("The function
'objective_function' is not defined in the provided code.")

            # Create an instance of the ObjectiveFunction
class with the custom function
            objective = ObjectiveFunction(lb, ub,
func=local_env['objective_function'])
            x, y = objective.get_meshgrid()
            z = objective.objective_function(x, y)

            fig = go.Figure(data=[go.Surface(z=z, x=x, y=y,
colorscale='Viridis')])
            return fig
        except Exception as e:
            return go.Figure(data=[go.Scatter(x=[0], y=[0],
text=[str(e)], mode='text')])


    @app.callback(
        Output('function-list', 'children'),
        Input('save-function-button', 'n_clicks'),
        Input({'type': 'delete-function-button', 'index':
ALL}, 'n_clicks'),
        State('function-name', 'value'),
        State('objective-function', 'value'),
        State('lb', 'value'),
        State('ub', 'value')
    )
    def update_function_list(save_n_clicks,
delete_n_clicks_list, name, code, lb, ub):
        ctx = dash.callback_context
        triggered = ctx.triggered[0] if ctx.triggered else
None

        # Handle save function logic
        if triggered and 'save-function-button' in
triggered['prop_id']:
            if save_n_clicks > 0 and name and code:
                try:
```

```
                    # Test the function before saving
                    function_code = f"""
    import numpy as np

    def objective_function(x, y):
        return {code}
    """
                    local_env = {}
                    exec(function_code, globals(), local_env)
                    if 'objective_function' not in local_env:
                        raise ValueError("The function
'objective_function' is not defined in the provided code.")

                    # Save the function to the database
                    function_entry = {'name': name, 'code':
code, 'lb': lb, 'ub': ub}

algorithm_parameters['objective_functions'].append(function_en
try)

functions_collection.insert_one(function_entry)
                    print("Function saved successfully.")
                except Exception as e:
                    print(f"Error: {str(e)}")


        # Handle delete function logic
        elif triggered and 'delete-function-button' in
triggered['prop_id']:
            index =
json.loads(triggered['prop_id'].split('.')[0])['index']
            if index <
len(algorithm_parameters['objective_functions']):
                functions_collection.delete_one(
                    {'_id':
ObjectId(algorithm_parameters['objective_functions'][index]['_
id'])})
                del
algorithm_parameters['objective_functions'][index]

        function_list = [
            html.Li([
                html.Span(f"{func['name']} (code:
{func['code']})", style={'color': '#FFFFFF'}),
                html.Button('Delete', id={'type': 'delete-
function-button', 'index': i}, n_clicks=0,
                            style={'backgroundColor':
'#FF4136', 'color': 'white', 'border': 'none', 'padding': '5px
10px',
                                   'cursor': 'pointer',
'margin-left': '10px'})
            ], style={'display': 'flex', 'align-items':
'center', 'margin-bottom': '10px'})
```

```
            for i, func in
enumerate(algorithm_parameters['objective_functions'])
            if 'name' in func and 'code' in func
        ]

        return function_list


    def evolutionary_algorithm(objective_function, pop_size,
n_iterations, n_evolution_iterations, n_populations,
                               sigma_limits, y_limits,
d_limits):
        sigma_min, sigma_max = map(float,
sigma_limits.split(','))
        y_min, y_max = map(float, y_limits.split(','))
        d_min, d_max = map(float, d_limits.split(','))
        progress = []

        def mutate_params(params):
            return {
                'sigma2': np.clip(params['sigma2'] +
np.random.uniform(-0.1, 0.1), sigma_min, sigma_max),
                'y': np.clip(params['y'] + np.random.uniform(-
0.1, 0.1), y_min, y_max),
                'd': np.clip(params['d'] + np.random.uniform(-
0.1, 0.1), d_min, d_max)
            }

        def create_random_params():
            return {
                'sigma2': np.random.uniform(sigma_min,
sigma_max),
                'y': np.random.uniform(y_min, y_max),
                'd': np.random.uniform(d_min, d_max)
            }

        best_populations = []
        for evolution_iteration in
range(n_evolution_iterations):
            populations = []
            for _ in range(n_populations):
                algorithm = CGWOAlgorithm(objective_function,
fitness_parameter=-1)
                algorithm.add_params_variation('evolution',
**create_random_params())
                positions, _, alpha_score, _, _ =
algorithm.optimize(n_iterations, pop_size, 2, 'evolution')
                populations.append((abs(alpha_score),
algorithm))

            populations.sort(key=lambda p: p[0])
            best_populations = populations[:3]
            for i in range(n_populations):
```

```
                _, algorithm = populations[i %
len(best_populations)]
                new_params =
mutate_params(algorithm.params_variations['evolution'])
                algorithm.add_params_variation('evolution',
**new_params)
                positions, _, alpha_score, _, _ =
algorithm.optimize(n_iterations, pop_size, 2, 'evolution')
                populations[i] = (abs(alpha_score), algorithm)

            progress.append(f"Iteration {evolution_iteration +
1}: Best Alpha Score: {best_populations[0][0]:.4f}")

        return best_populations, progress


    @app.callback(
        Output('evolution-progress', 'children'),
        Output('evolution-result-list', 'children'),
        Output('param-variation-list', 'children',
allow_duplicate=True),
        Input('evolute-button', 'n_clicks'),
        State('evolution-population-size', 'value'),
        State('evolution-iterations', 'value'),
        State('evolution-evolution-iterations', 'value'),
        State('evolution-wolf-populations', 'value'),
        State('evolution-function-dropdown', 'value'),
        State('sigma-limits', 'value'),
        State('y-limits', 'value'),
        State('d-limits', 'value'),
        prevent_initial_call=True
    )
    def perform_evolution(n_clicks, pop_size, n_iterations,
n_evolution_iterations, n_populations, selected_function,
sigma_limits, y_limits, d_limits):
        if n_clicks == 0:
            return "", [], [], []

        selected_function_details = next(
            (func for func in
algorithm_parameters['objective_functions'] if func['name'] ==
selected_function),
            None)

        if not selected_function_details:
            return "No valid function selected.", [], [], []

        selected_function_code =
selected_function_details['code']
        lb = selected_function_details['lb']
        ub = selected_function_details['ub']

        function_code = f"""
```

```
    import numpy as np

    def objective_function(x, y):
        return {selected_function_code}
    """
        local_env = {}
        exec(function_code, globals(), local_env)
        if 'objective_function' not in local_env:
            return "Error in function code.", [], [], []

        objective = ObjectiveFunction(lb, ub,
func=local_env['objective_function'])

        best_populations, progress =
evolutionary_algorithm(objective.objective_function, pop_size,
n_iterations, n_evolution_iterations, n_populations,
sigma_limits, y_limits, d_limits)

        result_list = []
        for i, (_, pop) in enumerate(best_populations):
            params = pop.params_variations['evolution']
            result_list.append(html.Li(f"Population {i + 1}:
Alpha Score = {pop.get_alpha_score():.4f}, Iterations =
{n_iterations}, y = {params['y']}, d = {params['d']}, sigma2 =
{params['sigma2']}",
                                       style={'color':
'#FFFFFF'}))

        best_populations_data = [
            {
                'alpha_score': pop.get_alpha_score(),
                'params': pop.params_variations['evolution']
            }
            for _, pop in best_populations
        ]

        # Save the best parameters automatically
        if best_populations_data:
            best_population = min(best_populations_data,
key=lambda p: p['alpha_score'])
            params = best_population['params']
            param_entry = {'name': f"evolutionary_best", 'y':
params['y'], 'd': params['d'], 'sigma2': params['sigma2']}

algorithm_parameters['param_variations'].append(param_entry)
            params_collection.insert_one(param_entry)

        # Update the param variation list to reflect the new
saved params
        param_variation_list = [
            html.Li([
                html.Span(f"{param['name']} (y={param['y']},
d={param['d']}, sigma2={param['sigma2']})",
```

```
                            style={'color': '#FFFFFF'})
            ], style={'display': 'flex', 'align-items':
'center', 'margin-bottom': '10px'})
            for i, param in
enumerate(algorithm_parameters['param_variations'])
        ]

        return html.Ul([html.Li(p) for p in progress],
style={'color': '#FFFFFF'}), result_list, param_variation_list


    @app.callback(
        Output('param-variation-list', 'children'),
        Input({'type': 'delete-param-button', 'index': ALL},
'n_clicks'),
        prevent_initial_call=True
    )
    def update_param_variation_list(delete_n_clicks_list):
        ctx = dash.callback_context
        triggered = ctx.triggered[0] if ctx.triggered else
None

        if triggered and 'delete-param-button' in
triggered['prop_id']:
            index =
json.loads(triggered['prop_id'].split('.')[0])['index']
            if index <
len(algorithm_parameters['param_variations']):
                params_collection.delete_one({'_id':
ObjectId(algorithm_parameters['param_variations'][index]['_id'
])})
                del
algorithm_parameters['param_variations'][index]

        return [
            html.Li([
                html.Span(f"{param['name']} (y={param['y']},
d={param['d']}, sigma2={param['sigma2']})",
                          style={'color': '#FFFFFF'}),
                html.Button('Delete', id={'type': 'delete-
param-button', 'index': i}, n_clicks=0,
                            style={'backgroundColor':
'#FF4136', 'color': 'white', 'border': 'none', 'padding': '5px
10px',
                                   'cursor': 'pointer',
'margin-left': '10px'})
            ], style={'display': 'flex', 'align-items':
'center', 'margin-bottom': '10px'})
            for i, param in
enumerate(algorithm_parameters['param_variations'])
        ]
```

```
    if __name__ == '__main__':
        app.run_server(debug=True)
```

Filename: gray_wolf_algorithm.py

```
    import numpy as np
    import plotly.graph_objs as go


    class ObjectiveFunction:
        def __init__(self, lb, ub, func=None):
            self.lb = lb
            self.ub = ub
            self.x_min = -5
            self.x_max = 5
            self.y_min = -5
            self.y_max = 5
            if func is None:
                self.func = self.default_objective_function
            else:
                self.func = func

        def default_objective_function(self, x, y):
            return -20 * np.exp(-0.2 * np.sqrt(0.5 * (x ** 2 +
y ** 2))) - np.exp(
                0.5 * (np.cos(2 * np.pi * x) + np.cos(2 *
np.pi * y))) + 20 + np.e

        def objective_function(self, x, y):
            return self.func(x, y)

        def get_bounds(self):
            return self.lb, self.ub

        def get_meshgrid(self, num_points=100):
            x = np.linspace(self.x_min, self.x_max,
num_points)
            y = np.linspace(self.y_min, self.y_max,
num_points)
            x, y = np.meshgrid(x, y)
            return x, y


    class CGWOAlgorithm:
        def __init__(self, objective_function,
fitness_parameter=-1):
            self.objective_function = objective_function
            self.fitness_parameter = fitness_parameter
            self.params_variations = {}
```

```
            self.alpha_score = np.inf

        def add_params_variation(self, name, y, d, sigma2):
            self.params_variations[name] = {'y': y, 'd': d,
'sigma2': sigma2}

        def initialize_positions(self, n_wolves, dim):
            return np.random.uniform(-5, 5, size=(n_wolves,
dim))

        def calculate_new_position(self, a, Xp_t, X_t):
            r1, r2 = np.random.rand(2)
            A = 2 * a * r1 - a
            B = 2 * r2
            C = abs(B * Xp_t - X_t)
            return Xp_t - A * C

        def update_wolves_positions_crazy(self, X_t, Xp_alpha,
Xp_beta, Xp_delta, n_iterations, t, y=0, d=0, sigma2=0):
            n_wolves, dim = X_t.shape
            a = 2.0 - t * (2.0 / n_iterations)
            E_r = np.zeros_like(X_t)

            for i in range(n_wolves):
                for j in range(dim):
                    X1 = self.calculate_new_position(a,
Xp_alpha[j], X_t[i][j])
                    X2 = self.calculate_new_position(a,
Xp_beta[j], X_t[i][j])
                    X3 = self.calculate_new_position(a,
Xp_delta[j], X_t[i][j])

                    Xp_new = (X1 + X2 + X3) / 3
                    H_k = np.random.randn()
                    E_r[i, j] = y * E_r[i, j] - d * (Xp_new -
X_t[i, j]) + sigma2 * H_k
                    X_t[i, j] = np.clip(Xp_new + E_r[i, j], -
5, 5)

            return X_t

        def optimize(self, n_iterations, n_wolves, dim,
variation_name):
            if variation_name not in self.params_variations:
                raise ValueError(f"Variation name
'{variation_name}' not found in parameters variations.")

            params = self.params_variations[variation_name]

            positions = self.initialize_positions(n_wolves,
dim)

            alpha_pos = np.zeros(dim)
```

```
            alpha_score = np.inf
            beta_pos = np.zeros(dim)
            beta_score = np.inf
            delta_pos = np.zeros(dim)
            delta_score = np.inf

            all_positions = []
            all_stats = []

            for t in range(n_iterations):
                for i in range(n_wolves):
                    fitness =
abs(self.objective_function(positions[i][0], positions[i][1]))

                    if fitness < alpha_score:
                        alpha_score = fitness
                        alpha_pos = positions[i].copy()
                    elif fitness < beta_score:
                        beta_score = fitness
                        beta_pos = positions[i].copy()
                    elif fitness < delta_score:
                        delta_score = fitness
                        delta_pos = positions[i].copy()

                positions =
self.update_wolves_positions_crazy(positions, alpha_pos,
beta_pos, delta_pos, n_iterations, t,

**params)
                all_positions.append(positions.copy())
                all_stats.append((t + 1, alpha_score,
beta_score, delta_score))

            self.alpha_score = alpha_score  # Update
alpha_score
            return all_positions, all_stats, alpha_score,
beta_score, delta_score

        def get_alpha_score(self):
            return self.alpha_score

        def get_params(self):
            return self.params_variations


    def create_animation_frames(all_positions,
objective_function, x, y, plot_bg):
        frames = []
        for t, positions in enumerate(all_positions):
            z_wolves = objective_function(positions[:, 0],
positions[:, 1])
            frames.append(go.Frame(data=[
                go.Surface(z=objective_function(x, y), x=x,
```

```
y=y, colorscale='Viridis', opacity=0.6),
                go.Scatter3d(x=positions[:, 0], y=positions[:,
1], z=z_wolves, mode='markers',
                             marker=dict(size=5, color='red'),
name="Wolves"),
                go.Scatter3d(x=[positions[0][0]],
y=[positions[0][1]], z=[z_wolves[0]], mode='markers',
                             marker=dict(size=10,
color='gold', symbol='diamond'), name="Alpha"),
                go.Scatter3d(x=[positions[1][0]],
y=[positions[1][1]], z=[z_wolves[1]], mode='markers',
                             marker=dict(size=10,
color='blue', symbol='square'), name="Beta"),
                go.Scatter3d(x=[positions[2][0]],
y=[positions[2][1]], z=[z_wolves[2]], mode='markers',
                             marker=dict(size=10,
color='green', symbol='circle'), name="Delta")
            ], name=f'frame{t}',
layout=go.Layout(title=f"Iteration: {t + 1}",
paper_bgcolor=plot_bg)))
        return frames


    def create_figure(all_positions, objective, x, y,
plot_bg='rgba(240, 240, 240, 1)'):
        frames = create_animation_frames(all_positions,
objective.objective_function, x, y, plot_bg)
        initial_z_wolves =
objective.objective_function(all_positions[0][:, 0],
all_positions[0][:, 1])
        lb, ub = objective.get_bounds()
        fig = go.Figure(
            data=[
                go.Surface(z=objective.objective_function(x,
y), x=x, y=y, colorscale='Viridis', opacity=0.6),
                go.Scatter3d(x=all_positions[0][:, 0],
y=all_positions[0][:, 1], z=initial_z_wolves,
                             mode='markers',
marker=dict(size=5, color='red'), name="Wolves"),
                go.Scatter3d(x=[all_positions[0][0][0]],
y=[all_positions[0][0][1]], z=[initial_z_wolves[0]],
                             mode='markers',
                             marker=dict(size=10,
color='gold', symbol='diamond'), name="Alpha"),
                go.Scatter3d(x=[all_positions[0][1][0]],
y=[all_positions[0][1][1]], z=[initial_z_wolves[1]],
                             mode='markers',
                             marker=dict(size=10,
color='blue', symbol='square'), name="Beta"),
                go.Scatter3d(x=[all_positions[0][2][0]],
y=[all_positions[0][2][1]], z=[initial_z_wolves[2]],
                             mode='markers',
                             marker=dict(size=10,
```

```
color='green', symbol='circle'), name="Delta")
            ],
            layout=go.Layout(
                updatemenus=[dict(type="buttons",
showactive=False,
                                  buttons=[dict(label="Play",

method="animate",
                                                args=[None,

dict(frame=dict(duration=200, redraw=True),
fromcurrent=True)])],
                                  x=0, y=-0.2, xanchor="left",
yanchor="bottom",
                                  pad=dict(r=10, t=10, b=20,
l=20),
                                  font=dict(color="black"),
                                  bgcolor="yellow",
bordercolor="yellow")],
                title='Iteration: 1',
                scene=dict(zaxis=dict(range=[lb, ub]),
                           bgcolor=plot_bg),
                plot_bgcolor=plot_bg,
                paper_bgcolor=plot_bg,
                showlegend=True,
                legend=dict(x=0, y=1),
                margin=dict(l=0, r=0, b=0, t=50),
            ),
            frames=frames
        )

        fig.update_layout(
            scene_camera=dict(eye=dict(x=1.25, y=1.25,
z=1.25)),
            scene_dragmode='orbit'
        )

        return fig
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	96 с.
	Інші документи	
2.	Презентаційні матеріали	18 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Пелюшок Б.В.			Дослідження гібридного методу правдоподібної кластеризації на основі еволюційного алгоритму
Перевірив		Шафроненко А.Ю.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-22-3	122
ХНУРЕ кафедра ШІ	