Optimization Suite - project report

Class: Python in the Enterprise

Piotr Żeberek	407663	gr. 6
Przemysław Ryś	405024	gr. 6

Contents

1 Project idea

The goal of this project was to create a graphical user interface application showcasing various optimization problems which we call 'scenarios'.

GitHub repository: `https://github.com/piotr-zeberek-agh/OptimizationSuite`

2 Project structure

The project consists of the following files and directories:

- **app.py** - the main file of the application, displays main window.
- **gui** - module containing the MainWindow class and help windows for each scenario.
- **default_scenarios** - module containing three main scenarios: Gradient Descent, Fullerene Structures, Portfolio Optimization.
- **utilities** - module containing abstract scenario class and utility functions.
- **results** - directory containing saved plots from scenarios.
- **config** - directory containing configuration for help windows.
- **resources** - directory containing images used in the application (icons).

Graphical User Interface (GUI) was implemented using the PyQt6 library. Other important modules used in the project are e.g. NumPy, Matplotlib, and Pandas (all available in the requirements.txt file).

Every Scenario class is derived from common abstract class named `Scenario` and sets up portion of the main window reserved for parameters input and displaying results. Separate classes for managing plots and charts are also defined e.g. `GradientDescentChartWidget`. More complex functionalities are sometimes moved to separate classes. Such case is present in the fullerene structure scenario where calculations are handles by the independent thread represented by `FullereneWorkerThread` class and are contained within `BrennerPotential` and `Fullerene` classes.

3 Running the app

The intended way to run the application is to use the virtual environment. To create it, run the following commands in the terminal:

```
python -m venv venv
source venv/bin/activate (for Linux systems)
pip install -r requirements.txt
```

After setting up the virtual environment, run the application with the following command:

```
python app.py
```

4 User interface

4.1 Main Window

The main launch window contains elements such as the window title, which changes when a scenario is selected, buttons, consecutively representing help about the current window, changing the window contrast (white/dark-gray), and (enabled by default) an autosave button for the scenario's graphs. There is also a drop-down selection of the scenario, which allows changing the window content.

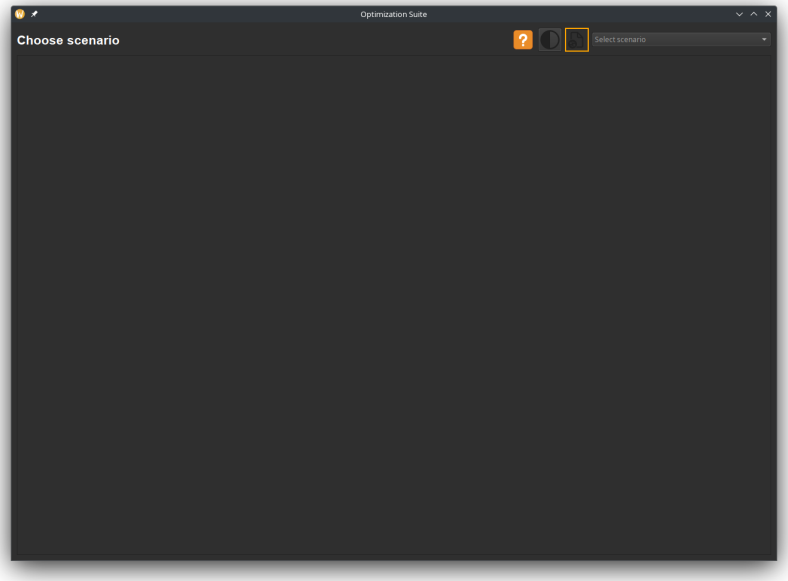

Figure 1: Main window of the application

After selecting a scenario, the blank space of the main application window is filled. Detailed description of algorithms used in each scenario can be found further in this report. The window title in the upper left corner changes to the selected option.

4.2 Gradient Descent Window

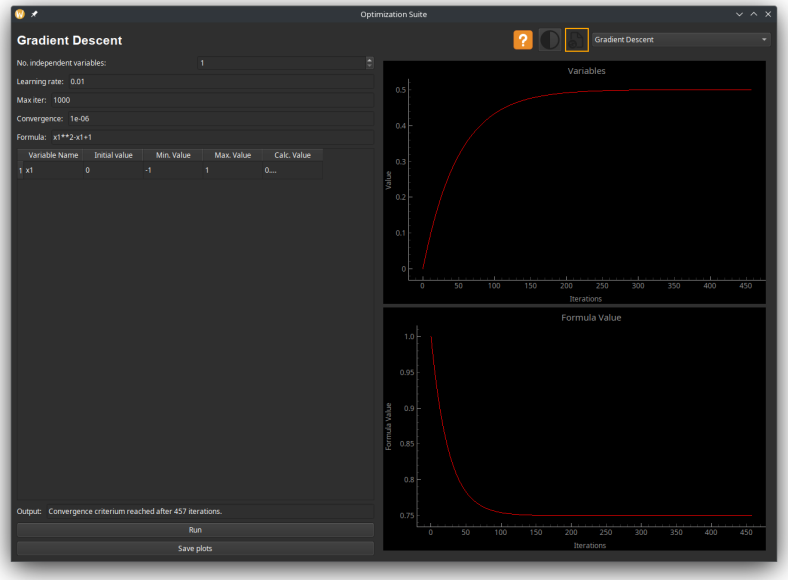

Figure 2: Gradient Descent window.

The GUI workspace contains the following elements:

- **No. independent variables** - An input field (Integer type) for specifying the number of independent variables, typically denoted as "x{i}", where {i} represents consecutive natural numbers (1, 2, 3, ...).
- **Learning rate** (Float) - The learning rate controls the step size during the optimization process. It determines how much the model adjusts its parameters in response to the gradient. A high learning rate may cause the algorithm to "overshoot" the optimal solution, while a low learning rate can result in slower convergence.
- **Max iter** (Integer) - This field specifies the maximum number of iterations the algorithm will run for. If the algorithm does not converge before reaching this limit, it will stop. This serves as a safeguard to prevent the algorithm from running indefinitely in case of convergence issues.
- **Convergence** - Refers to the difference between values in two consecutive iterations that is used as a criterion to determine whether the optimal solution has been reached.
- **Formula** - This field represents a mathematical expression or function used in the algorithm.

Below is a table containing the columns:

- **Variable Name** - The name of the variable.
- **Initial value** - The initial value for the independent variables.
- **Min. Value** - The minimum value that the independent variables can take.
- **Max. Value** - The maximum value that the independent variables can take.

- **Calc. Value** - Current calculated value, which is filled in as a result of the algorithm's operation (the minimum found).

The second, third, and fourth columns allow us to specify initial values for the independent variables, while the fifth column is filled with the result of the algorithm's operation (the found minimum).

Below the table, the **output** field provides information on how many iterations it took to achieve the goal or whether the algorithm stopped e.g. due to reaching variables limits.

On the right side, there are two plots:

- The first plot, titled **Variables**, shows the change in the values of the variables that define the optimal solution.
- The second plot, titled **Formula Value**, shows the change in the value of the function being optimized over the course of the algorithm.

4.3 Structure Of Fullerenes Window

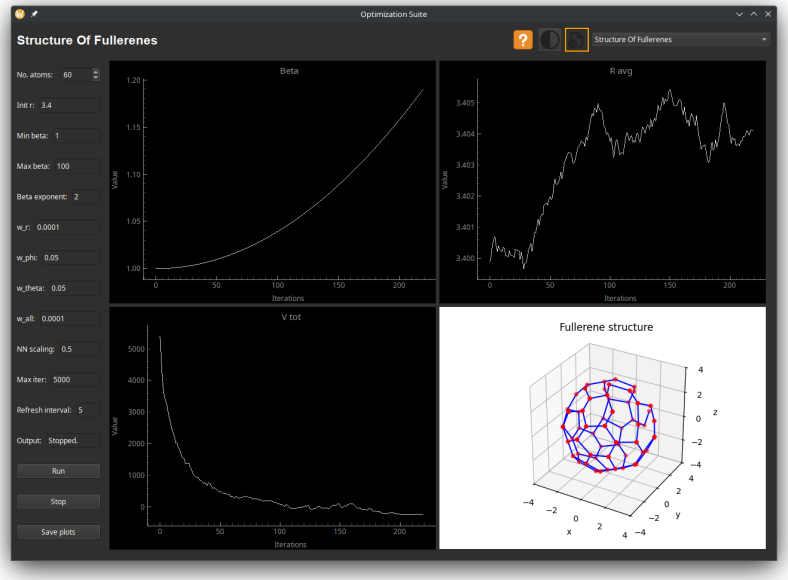

Figure 3: Structure of Fullerenes window.

The GUI workspace contains the following elements:

- **No. atoms**: Specifies the number of atoms in the fullerene system.
- **Init r**: The initial radius fo a sphere on which the atoms are randomly placed at the beginning of the optimization process.
- **Min beta** and **Max beta**: Define the range of the parameter β representing the inverse temperature in the simulated annealing algorithm.
- **Beta exponent**: Adjusts the growth or decay of the β parameter.
- **w_r**, **w_phi**, **w_theta**, **w_all**: Scaling factor for changes applied to the system during optimization.

- **NN scaling**: Specifies the scaling factor for determining the nearest neighbors (used for plotting). Nearest neighbors distance is calculated as *NN distance* = *NN scaling* · *avg distance of atoms from the center*.
- **Max iter**: The maximum number of iterations for the optimization algorithm.
- **Refresh interval**: Sets how often the GUI updates the plots during the optimization process.
- **Output**: Displays information about the progress of the algorithm in the form of the current number of iterations.

The GUI also includes the following control buttons:

- **Run**: Starts the optimization algorithm.
- **Stop**: Stops the algorithm's execution.
- **Save plots**: Saves the current state of the plots to files for further analysis or documentation.

On the right side, there are four dynamically updating plots, each running in a separate thread. These plots display:

- **Plot 1**: The value of the parameter β as a function of the number of iterations.
- **Plot 2**: The average distance of atoms from the center of the sphere over time.
- **Plot 3**: The total potential energy of the system that is minimized.
- **Plot 4 (3D visualization)**: A 3D representation of the atomic configuration. As the algorithm progresses, this plot updates to show the realistic shape of the fullerene structure being formed.

4.4 Portfolio Optimization Window

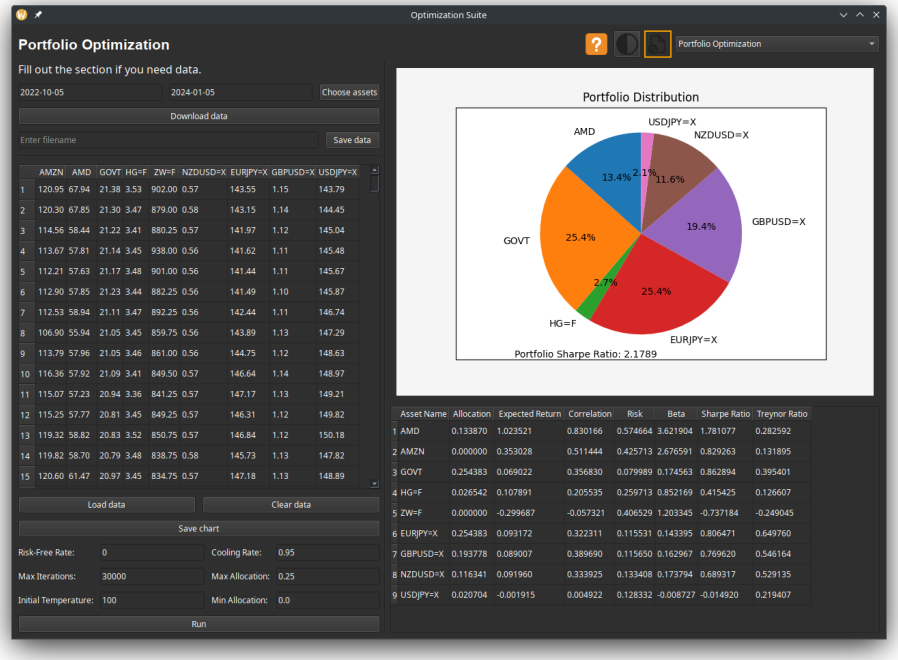

Figure 4: Portfolio Optimization window.

If you need data for specific assets, you can download them as follows:

- **Input interval (in year-month-day format)**.
- **Choose Assets**: A button that opens a window enabling you to select assets from various categories such as Stocks, Bonds, Commodities, ETFs, Currencies, and Indexes. These categories can be modified in the "config/tickers.json" file.
- **Download Data button**: This button is enabled after filling in the date fields and selecting assets. When pressed, it downloads the data.
- **Filename input text**: A field to enter the filename. If the data has already been downloaded and a filename is provided, the save button will become enabled. If the filename does not have the ".csv" extension, it will be added automatically.

Below this is an empty field where the selected resources will be displayed in the form of a table. The columns of the table are filled in when you select resources in the "Select resources" window, and when you click the "Download data" button fills in the data for the set interval. It also fills in after loading a data file, which serves as an alternative method of getting data. The table is filled with the closing prices of these assets for each day.

Below are the buttons:

- **Load data**: Loads asset data from a file, if available. By default, it opens the "data/" subdirectory.
- **Clear data**: Removes the asset data from memory, clearing the chart and the table of calculated asset parameters.
- **Save chart**: Saves the optimal portfolio chart to the "results/portfolio_optimization/" directory. If the autosave option is unchecked, the user selects the save directory manually.
- **Run**: Runs the algorithm, updating the results on the right side of the window.

The **Run** button has input parameters above it. Every 0.5 seconds, which includes monitoring the values of the set parameters of the annealing algorithm. We enter the following data

- **Risk-Free Rate**: Return on a risk-free asset.
- **Max Iterations**: Limit on algorithm steps.
- **Initial Temperature**: Initial temperature of the algorithm.
- **Cooling Rate**: Degree of cooling, acts as a multiplier of temperature in successive steps.
- **Max Allocation**: Upper limit on asset allocation.
- **Min Allocation**: Lower limit to asset allocation.

On the right side, there is a chart representing the percentage distribution of the optimal portfolio. This is shown as a pie chart. If a particular asset has a negative or near-zero return, it will not be displayed. Below the chart, the optimal Sharpe ratio is shown.

Below the chart, there is a table that, after running the algorithm, will be populated with the appropriate parameters that are essential for assessing the return and risk of the resulting portfolio. It may turn out that, for the selected assets, even though a distribution has been calculated, it is not statistically attractive in terms of return.

Note, the distributions of successive portfolios may differ in the percentage composition of assets, some less, some more. This is because different states of the portfolio meet the optimization of the index, at a similar level.

5 Algorithms description

5.1 Gradient Descent

Gradient Descent is an optimization algorithm used to minimize a function by iteratively moving towards the direction of the steepest descent, which is the negative gradient of the function. The method is based on the idea of finding the local minimum of a differentiable function $f(\mathbf{x})$, where $\mathbf{x}$ is a vector of variables.

The update rule for Gradient Descent is given by:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)$$

Where:

- $\mathbf{x}_k$ is the current value of the variables.
- $\nabla f(\mathbf{x}_k)$ is the gradient of the function at $\mathbf{x}_k$.
- α is the learning rate, controlling the size of the steps taken towards the minimum.

The process continues until convergence is reached or other limiting conditions are met.

5.2 Simulated Annealing

The Simulated Annealing algorithm is used in the next two scenarios therefore it is a good thing describe it generally. Simulated Annealing is a probabilistic method for solving optimization problems. It is inspired by the annealing process in metallurgy, where a material is heated and then slowly cooled to reach a stable state. The algorithm explores the solution space to find a global optimum by accepting both improving and, with some probability, non-improving solutions.

5.2.1 Algorithm Steps

1. **Initialization:**
   - Start with an initial solution $x_{current}$.
   - Set the initial temperature T_0.
2. **Iterative Process:**
   - Generate a new solution x_{new} by perturbing $x_{current}$.
   - Calculate the objective function $f(x)$ for both $x_{current}$ and x_{new}.
   - Accept x_{new} if $f(x_{new}) < f(x_{current})$, otherwise accept it with probability:
   $$P = \exp\left(-\beta(f(x_{new}) - f(x_{current}))\right).$$
   - Decrease the temperature (increase β).
3. **Stopping Condition:**
   - Stop after a maximum number of iterations or when convergence criteria are met.
   - Return the best solution found.

5.3 Fullerene Structures

This scenario aims to find the optimal structure of a fullerene molecule starting from a random configuration of atoms on a sphere. The optimization process is based on the simulated annealing algorithm.

5.3.1 Brenner's potential

Brenner's potential was chosen to calculate the energy a system consisting of n atoms. This energy is minimized during the optimization process.

Subsequent equations describe how to calculate the energy:

$$E_{\text{brenner}} = V_{tot} = \frac{1}{2}\sum_{i=1}^{n} V_i \tag{1}$$

i-th atom's interaction energy with its neighbors:

$$V_i = \sum_{j=1,\ j\neq i}^{n} V_{ij} = \sum_{j=1,\ j\neq i}^{n} f_{cut}(r_{ij})\left[V_R(r_{ij}) - \overline{B}_{ij}V_A(r_{ij})\right] \tag{2}$$

Cutoff function (to limit the range of interactions):

$$f_{cut}(r_{ij}) = \begin{cases} 1 & \text{dla } r_{ij} < R_1 \\ \frac{1}{2}\left[1 + \cos\left(\frac{r_{ij}-R_1}{R_2-R_1}\pi\right)\right] & \text{dla } R_1 \leq r_{ij} \leq R_2 \\ 0 & \text{dla } r_{ij} > R_2 \end{cases} \tag{3}$$

Repulsion potential:

$$V_R(r_{ij}) = \frac{D_e}{S-1}\exp\left[-\sqrt{2S}\lambda(r_{ij} - R_0)\right] \tag{4}$$

Attraction potential:

$$V_A(r_{ij}) = \frac{D_e S}{S-1}\exp\left[-\sqrt{\frac{2}{S}}\lambda(r_{ij} - R_0)\right] \tag{5}$$

Scaling factor for attraction potential that takes into account formed bonds:

$$\overline{B}_{ij} = \frac{B_{ij} + B_{ji}}{2} \tag{6}$$

$$B_{ij} = (1 + \xi_{ij})^{-\delta} \tag{7}$$

$$\xi_{ij} = \sum_{k=1,\ k\neq i,j}^{n} f_{cut}(r_{ik})g(\theta_{ijk}) \tag{8}$$

$$g(\theta_{ijk}) = a_0\left[1 + \frac{c_0^2}{d_0^2} - \frac{c_0^2}{d_0^2 + (1+\cos(\theta_{ijk}))^2)}\right] \tag{9}$$

θ_{ijk} - angle between vectors $\vec{r}_{ij} = \vec{r}_j - \vec{r}_i$ and $\vec{r}_{ik} = \vec{r}_k - \vec{r}_i$

Parameters of the potential used in the simulation:

D_e	S	λ	R_0	R_1	R_2	δ	a_0	c_0	d_0
6.325 eV	1.29	1.5 Å^{-1}	1.315 Å	1.7 Å	2.0 Å	0.80469	0.011304	19	2.5

Avoiding four bonds per atom

Considering the most popular fullerene C60 (buckminsterfullerene) as a reference, we know that each carbon atom forms exactly three bonds. Brenner potential allows for the formation of four bonds. Therefore, an additional condition is introduced to avoid such behavior.

For four bonds and symmetric atom distribution, the angles between vectors $\vec{r}_{ij}$ and $\vec{r}_{ik}$ are 90° (approximating by points on a plane), and for three bonds, 120°. Limiting the attractive

potential when the angle is 90° is one way to avoid the formation of four bonds. The following condition is introduced:

$$\cos(\theta_{ijk}) > 0 \rightarrow \xi_{ij} = 10 \tag{10}$$

Giving a large value (arbitrarily chosen 10) to ξ_{ij} effectively reduces B_{ij} and consequently $V_A(r_{ij})$.

5.3.2 Simulated Annealing for fullerenes

One step of the implemented simulated annealing algorithm consists of two main parts: shiting all atoms individually and collectively scaling the distance of all atoms from the center.

Shifting individual atoms
Iterating over all atoms (ideally, they should be chosen randomly but the difference is not significant), we change their coordinates (spherical in this case) by scaling them according to parameters w_r, w_ϕ, and w_θ.

$$r' = r \cdot [1 + (2 \cdot U(0,1) - 1)w_r] \tag{11}$$

$$\phi' = \phi \cdot [1 + (2 \cdot U(0,1) - 1)w_\phi] \tag{12}$$

$$\theta' = \theta \cdot [1 + (2 \cdot U(0,1) - 1)w_\theta] \tag{13}$$

In the above equations, $U(0,1)$ is a random number taken from a uniform distribution between 0 and 1.

Adjusting the angles so that they are in the correct ranges:

$$\phi' = \begin{cases} \phi' + 2\pi & \text{dla } \phi' < 0 \\ \phi' - 2\pi & \text{dla } \phi' > 2\pi \end{cases} \tag{14}$$

$$\theta' = \theta \text{ dla } \theta' \notin [0, \pi] \tag{15}$$

New coordinates are accepted based on the Metropolis-Hastings algorithm:

$$p_{\text{accept}} = min\left(1, \exp\left(-\beta(E_{\text{new}} - E_{\text{old}})\right)\right), \tag{16}$$

where E_{new} and E_{old} are the energies of the system with new and old coordinates, respectively. E should be calculated using Brenner's potential but we can get away with calculating only the difference in atom's interaction energy with its neighbors V_i. This simplification greatly reduces the computational cost of the simulation.

$$U(0,1) < p_{\text{accept}} \rightarrow \text{accept new coordinates} \tag{17}$$

Collectively scaling atoms distance from the center
Scaling factor s is calculated using parameter w_{all}:

$$s = 1 + (2 \cdot U(0,1) - 1)W_{all} \tag{18}$$

Every atom's distance from the center is then scaled:

$$r_i' = r_i \cdot s \tag{19}$$

As before, acceptance probability is calculated and new coordinates are accepted or rejected.

Beta parameter

$$\beta = \frac{1}{k_b T} \tag{20}$$

β is changed during the simulation according to the following formula:

$$\beta = \beta_{min} + \left(\frac{it}{N_{iter}}\right)^p (\beta_{max} - \beta_{min}) \tag{21}$$

where it is the current iteration number, N_{iter} is the total number of iterations, and p is a parameter that controls the rate of change of β.

Changing β from low (high temperature) to high (low temperature) values with iteration number allows the system to escape local minima at the beginning of the simulation and converge at the end. Hence, 'simulated annealing'.

5.4 Portfolio optimization

This scenario aims to find the optimal percentage distribution for an investment portfolio based on different asset types such as: stocks, bonds, commodities, ETFs, currencies and indexes. The simulated annealing algorithm is also used here.

The objective of portfolio optimization is to select the weights of different assets in a portfolio to achieve the best balance between return and risk. The Simulated Annealing algorithm can be applied to find the optimal asset weights that maximize the Sharpe ratio.

5.4.1 Sharpe Ratio and Risk

The Sharpe ratio is a measure of the risk-adjusted return of a portfolio. It is calculated as:

$$S = \frac{R_p - R_f}{\sigma_p}$$

where:

- S - the Sharpe ratio,
- R_p - the expected return of the portfolio,
- R_f - the risk-free rate,
- σ_p - the standard deviation of the portfolio's returns (a measure of risk).

The higher the Sharpe ratio, the better the portfolio is at delivering return per unit of risk. In portfolio optimization, the goal is to **maximize the Sharpe ratio**, which corresponds to maximizing return while minimizing risk. This is achieved by adjusting the weights of the assets in the portfolio.

5.4.2 Simulated Annealing for portfolio optimization

The goal of portfolio optimization is to find the best combination of assets that maximizes the risk-return ratio, as measured by the Sharpe ratio, while minimizing portfolio risk. To achieve this, we use the **Simulated Annealing** algorithm, which explores the solution space probabilistically.

Algorithm Steps:

1. **Initialize the current solution** x_0 (portfolio weights) and the initial temperature T_0.
2. **Calculate the objective function** (Sharpe ratio) for the current solution.

3. **Repeat the following steps until the stopping criterion is met:**

   - Generate a new solution x_{new} by making a small random change to the current solution $x_{current}$.
   - Calculate the Sharpe ratio for the new solution S_{new}.
   - If the new solution has a higher Sharpe ratio, accept it as the current solution.
   - If the new solution has a lower Sharpe ratio, accept it with a probability of:

     $$P = \exp\left(\frac{S_{new} - S_{current}}{T}\right)$$

     where S_{new} is the Sharpe ratio of the new solution, $S_{current}$ is the Sharpe ratio of the current solution, and T is the current temperature. This allows for some exploration of less optimal solutions to avoid getting stuck in local optima.
   - Decrease the temperature: $T = \alpha T$ (where $0 < \alpha < 1$).

4. **When the temperature reaches a minimum threshold or the maximum number of iterations is reached, return the best solution found.**

6 Conclusion

The Optimization Suite project demonstrates the various optimization scenarios through an interactive graphical user interface. Selection of implemented scenarios may seem random at first but it shows the applicability of Python as a general purpose programming language. Although the authors of this project did not manage to implement every desired functionality, the final result is more than satisfactory for them.