AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

WYDZIAŁ NAUK INŻYNIERYJNYCH

BADANIA OPERACYJNE

PRACA LABORATORYJNA

APLIKACJA DO ROZWIĄZYWANIA ZADAGNIEŃ PROGRAMOWANIA LINIOWEGO

Autor: Filip Rzepiela
Grupa: L2
Kierunek: Informatyka II
Prowadzący: dr Aldona Wota

NOWY SĄCZ 2024

Spis treści

1. Słownik pojęć

Programowanie liniowe to technika matematyczna służąca do rozwiązywania problemów optymalizacyjnych, gdzie zarówno funkcja celu, jak i ograniczenia są liniowe. W praktyce oznacza to, że chcemy znaleźć najlepsze (największe lub najmniejsze) wartości dla pewnych zmiennych, które spełniają określone warunki (ograniczenia) i są przedstawione jako równania lub nierówności liniowe. Używa się jej do planowania i podejmowania decyzji tam, gdzie trzeba podzielić ograniczone zasoby (jak pieniądze, materiały czy czas) w taki sposób, aby osiągnąć jak najlepszy rezultat, np. maksymalny zysk czy minimalny koszt. Narzędziem, które często pomaga znaleźć rozwiązanie takiego problemu, jest metoda sympleks.

Solver to narzędzie do optymalizacji wykorzystywane w różnych programach do tworzenia arkuszy kalkulacyjnych, takich jak Microsoft Excel, które pozwala użytkownikom definiować i rozwiązywać problemy optymalizacyjne, gdzie cel można zdefiniować jako maksymalizację, minimalizację lub osiągnięcie konkretnego poziomu danej zmiennej. Solver automatycznie dostosowuje wartości zmiennych decyzyjnych w modelu, aby znaleźć najlepsze rozwiązanie problemu, biorąc pod uwagę określone ograniczenia. Zaletą Solvera jest jego zdolność do pracy z nieliniowymi funkcjami oraz różnorodnymi typami ograniczeń, co sprawia, że jest narzędziem uniwersalnym i elastycznym w rozwiązywaniu skomplikowanych problemów optymalizacyjnych w różnych dziedzinach, takich jak logistyka, finanse, inżynieria czy zarządzanie operacjami. Solver wykorzystuje różne algorytmy optymalizacyjne, w zależności od charakteru problemu, włączając w to metody sympleks dla problemów liniowych oraz różne techniki dla zadań nieliniowych.

Python to wysokopoziomowy, interpretowany język programowania, który został stworzony przez Guido van Rossuma i po raz pierwszy wydany w 1991 roku. Charakteryzuje się składnią, która jest zarówno czytelna, jak i zwięzła, co ułatwia naukę i efektywną pracę z kodem. Python wspiera różne paradygmaty programowania, w tym proceduralne, obiektowe i funkcyjne. Jego wszechstronność sprawia, że jest używany w wielu różnych dziedzinach, od tworzenia stron internetowych, przez analizę danych, sztuczną inteligencję, naukę o danych, automatykę do testowania, aż po tworzenie aplikacji sieciowych i desktopowych. Python posiada rozbudowaną bibliotekę standardową oraz ogromne społeczeństwo, które wspiera jego rozwój poprzez tworzenie i udostępnianie otwartoźródłowych pakietów i modułów, co dodatkowo rozszerza jego funkcjonalność. Jest to język, który jest szczególnie ceniony za swoją elastyczność i łatwość integracji z innymi technologiami, co sprawia, że jest chętnie wybierany zarówno przez początkujących, jak i doświadczonych programistów.

Jupyter Notebook to interaktywne środowisko programistyczne, które umożliwia tworzenie i udostępnianie dokumentów zawierających żywy kod, równania, wizualizacje oraz tekst narracyjny. Jego nazwa pochodzi od języków programowania Julia, Python i R, choć obsługuje wiele innych języków za pomocą tzw. "kerneli". Dokumenty Jupyter Notebook są używane w szczególności do analizy danych, wizualizacji danych, uczenia maszynowego i nauki o danych, gdzie pozwalają na eksperymentalne programowanie i łatwe dzielenie się wynikami pracy. Dokumenty te składają się z sekwencji komórek, które mogą zawierać kod, tekst sformatowany (Markdown), równania (LaTeX) czy grafiki i są bardzo popularne w edukacji i badaniach naukowych ze względu na swoją interaktywność i wszechstronność. Jupyter Notebook jest aplikacją internetową, co sprawia, że może być uruchamiana lokalnie na komputerze użytkownika lub na zdalnym serwerze, co ułatwia współpracę i dostępność na różnych platformach.

Microsoft Excel to zaawansowany program do tworzenia arkuszy kalkulacyjnych, rozwijany i dystrybuowany przez firmę Microsoft. Jest częścią pakietu biurowego Microsoft Office i umożliwia użytkownikom przetwarzanie danych, prowadzenie zaawansowanych analiz numerycznych, tworzenie różnorodnych wykresów oraz modelowanie finansowe. Excel oferuje szeroki zakres narzędzi do zarządzania danymi, w tym funkcje do obliczeń matematycznych i statystycznych, narzędzia do analizy danych (takie jak Tabele przestawne), a także możliwość pisania makr w języku VBA (Visual Basic for Applications), co pozwala na automatyzację zadań i dostosowywanie funkcjonalności programu do indywidualnych potrzeb użytkownika.

2. Zakres pracy

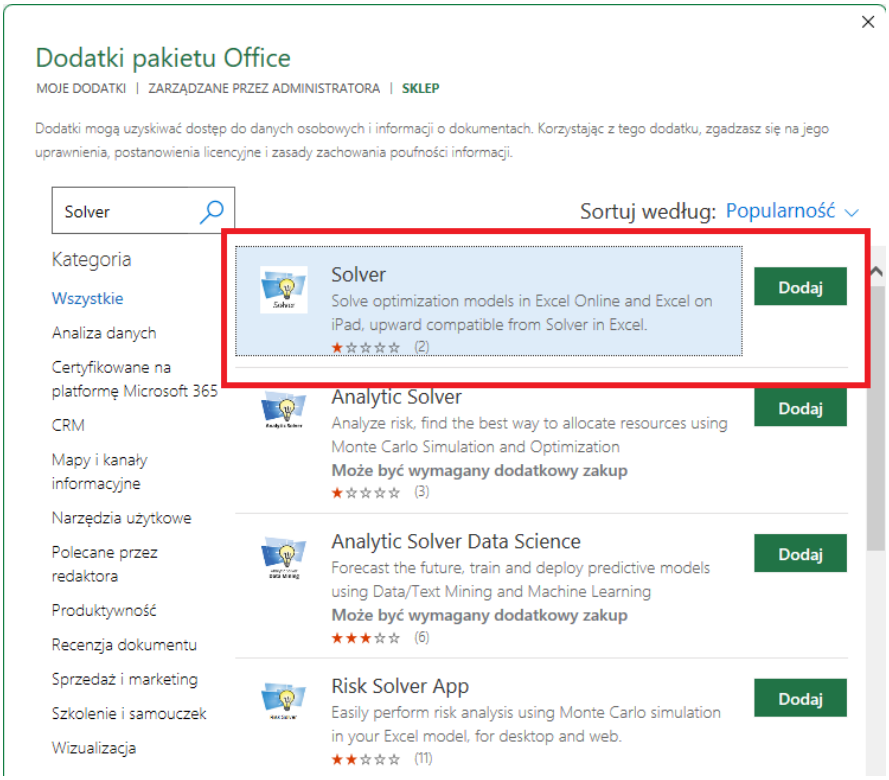
Zakres pracy laboratoryjnej objął opracowanie aplikacji do rozwiązywania programowania liniowego w dowolnym języku programowania. W przypadku tej pracy aplikacja zrealizowana zostanie w języku Python z wykorzystaniem narzędzia Jupyter Notebook. Program będzie się składać z trzech zakładek odpowiedzialnych za: Wprowadzenie zmiennych, ustawienie ograniczeń oraz ustawienie funkcji celu. Będzie posiadał funkcjonalność rozwiązywania problemu programowania liniowego oraz generował wykresy tj. nierówności, liniowy, kołowy oraz słupkowy. Wszystko, co użytkownik wykona podczas użytkowania programu, będzie można zapisać do formatu PDF.

3. Wykonanie programowania liniowego w MS Excel.

W tym rozdziale przedstawione zostanie przygotowanie środowiska Microsoft Excel do wykonania przykładowego zadania programowania liniowego.

3.1. Instalacja dodatku Solver

Na początku należy utworzyć nowy arkusz w programie Microsoft Excel. Następnie przechodzimy do opcji „Pobierz dodatki" i wyszukujemy dodatku Solver, za pomocą którego będziemy wykonywać zadanie z tematyki programowania liniowego.



Na liście dostępnych dodatków wyświetli się kilka wersji Solvera, wybieramy jednak pierwszą, zaznaczoną na powyższym obrazku kolorem czerwonym. Następnie przygotowujemy arkusz Excel pod wykonywanie obliczeń z treści zadania.

3.2. Treść zadania

Przykład 4. Spółdzielnia produkcyjna sporządza mieszankę paszową dla trzody chlewnej z dwóch produktów: P1 i P2. Mieszanka paszowa ma dostarczyć trzodzie chlewnej pewnych składników odżywczych: S1, S2, i S3, w ilościach nie mniejszych niż określone minima. Zawartość składników odżywczych w jednostce poszczególnych produktów, ceny produktów, a także minimalne ilości składników podano w tabl. 15.

Składniki	Zawartość składnika w 1 kg produktu		Minimalna ilość składnika
	P_1	P_2	
S_1	3	9	27
S_2	8	4	32
S_3	12	3	36
Cena (w zł)	6	9	

Należy zakupić takie ilości produktów P1 i P2, aby dostarczyć trzodzie chlewnej składników odżywczych S1, S2, i S3, w ilościach nie mniejszych niż minima określone w tabl. 15 i aby koszt zakupu był minimalny. Zbudować model matematyczny tego zagadnienia i przedstawić rozwiązanie metodą geometryczną.

3.3. Rozwiązanie

Na podstawie danych z treści zadania uzupełniony został arkusz Excel, co zostało przedstawione na rysunku poniżej. W komórkach C7:D7 wpisane zostały ceny, natomiast w komórkach C11:D13 zawartość składnika w 1 kg produktu.

	A	B	C	D	E	F	G	H	I
1	Zmienne decyzyjne		P1	P2					
2	Rozwiązanie								
3									
4	FC-Funkcja celu		min	0					
5									
6	Cel współczynniki		C1	C2					
7			6	9					
8									
9	Ograniczenie								
10									
11	Pierwsze		3	9		0		27	
12	Drugie		8	4		0		32	
13	Trzecie		12	3		0		36	
14									
15									

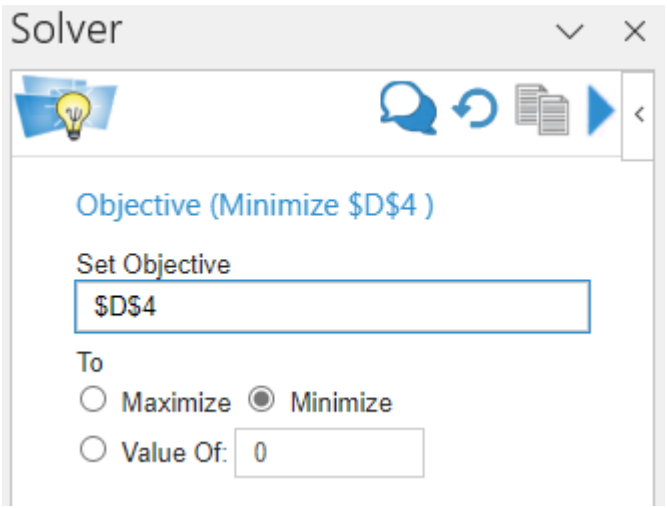
Komórka D4 ustawiona została jako funkcja celu, stanowiąca iloczyn komórek oznaczonych kolorem żółtym oraz oznaczonych kolorem jasnozielonym =SUMA.ILOCZYNÓW(C2:D2;C7:D7). W komórkach F11:F13 ustawione zostały następujące funkcje:

=SUMA.ILOCZYNÓW($C$2:$D$2;C11:D11),

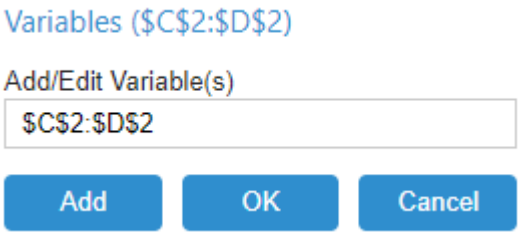
=SUMA.ILOCZYNÓW($C$2:$D$2;C12:D12)

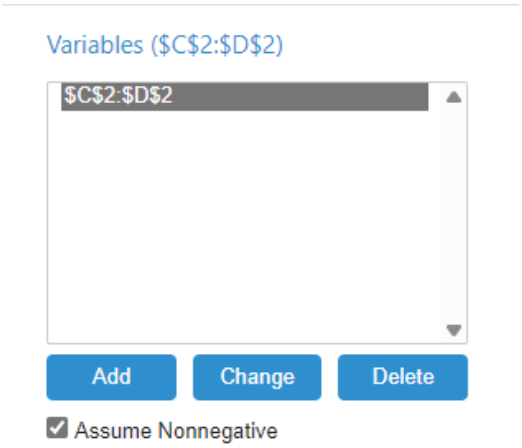
=SUMA.ILOCZYNÓW($C$2:$D$2;C13:D13)

Powyższe funkcje stanowią sumę iloczynów komórek oznaczonych kolorem jasnozielonym oraz komórek oznaczonych kolorem niebieskim. Następnie uruchomiony został dodatek do MS Excel Solver, gdzie dokonano ustawienia komórki $D$4, w której znajdować się będzie funkcja celu.

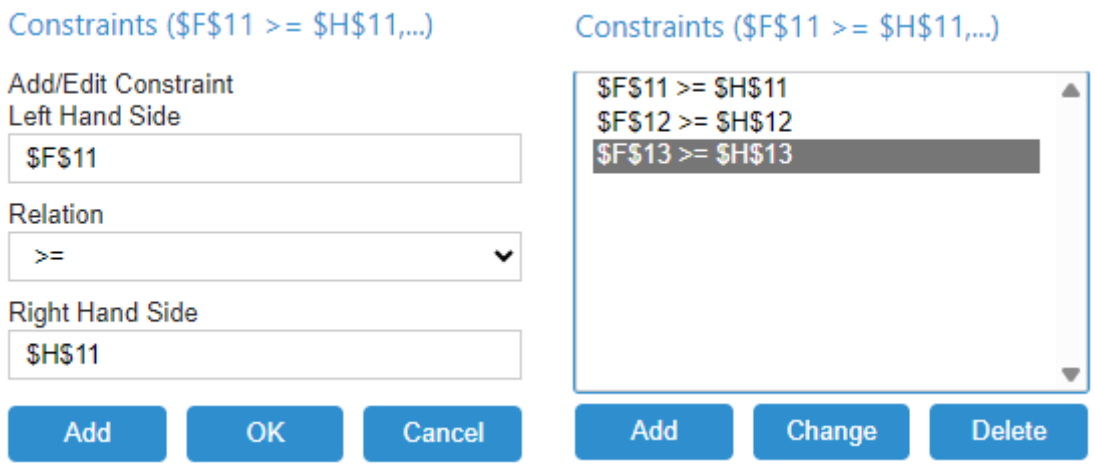


Następnie określono w programie komórki, w których znajdą się w po wykonaniu obliczeń wartości P1 oraz P2. Wyznaczone zostały komórki C2:D2.

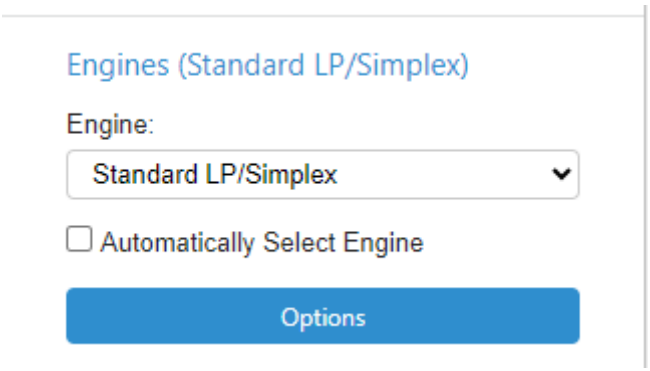




Kolejnym etapem było wprowadzenie do programu ograniczeń. Proces ten przedstawiony został poniżej.



W ostatnim kroku ustawiony został silnik na LP/Simplex.

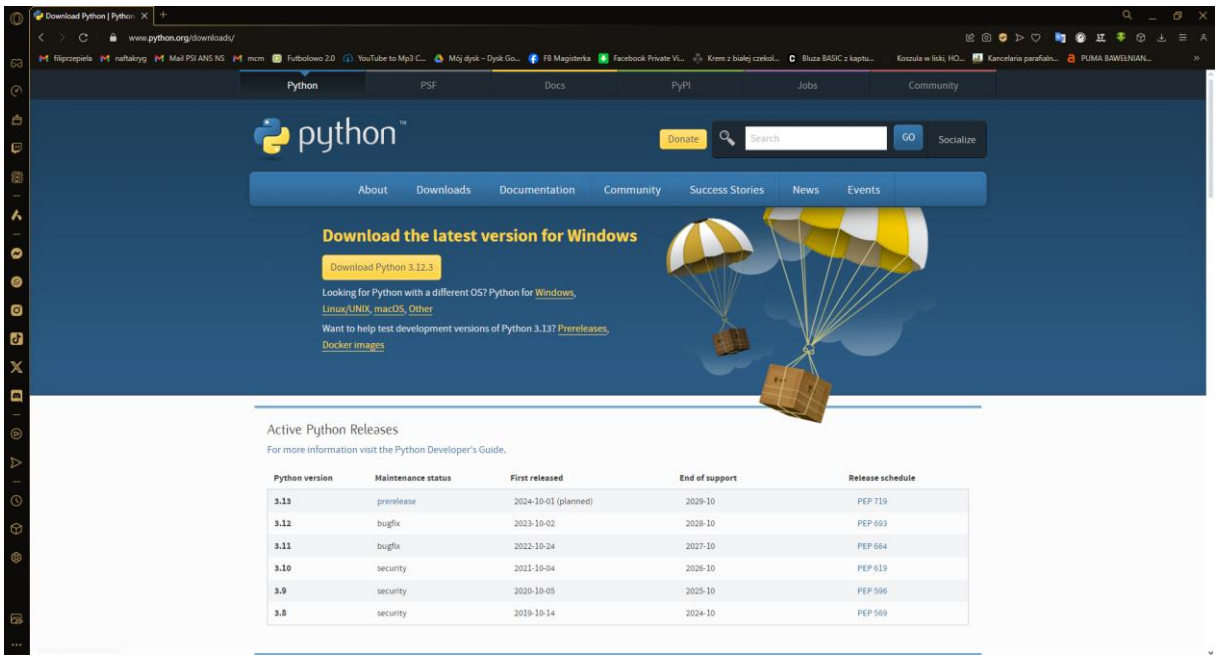


Następnie należy uruchomić program i czekać na wykonanie obliczeń. Efekt uzyskany po zakończeniu pracy programu przedstawiony został na poniższym rysunku. Funkcja celu wynosi 36, natomiast wartości P1 oraz P2 wynoszą 3 oraz 2.

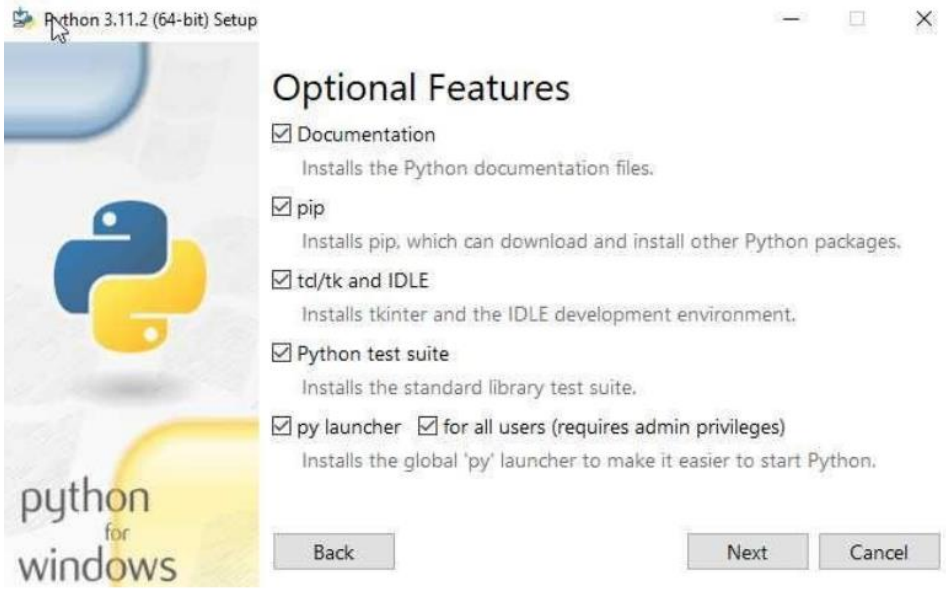
	A	B	C	D	E	F	G	H	I
1	Zmienne decyzyjne		P1	P2					
2	Rozwiązanie		3,00	2,00					
3									
4	FC-Funkcja celu		min	36					
5									
6	Cel współczynniki		C1	C2					
7			6,00	9,00					
8									
9	Ograniczenie								
10									
11	Pierwsze		3	9		27,00		27,00	
12	Drugie		8	4		32,00		32,00	
13	Trzecie		12	3		42,00		36,00	
14									
15									

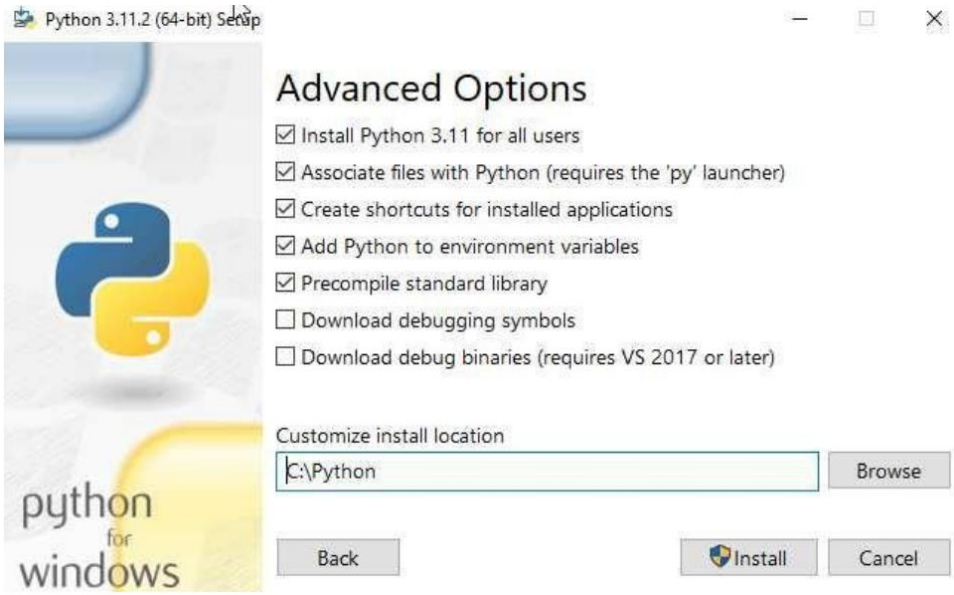
4. Przygotowanie środowiska do implementacji

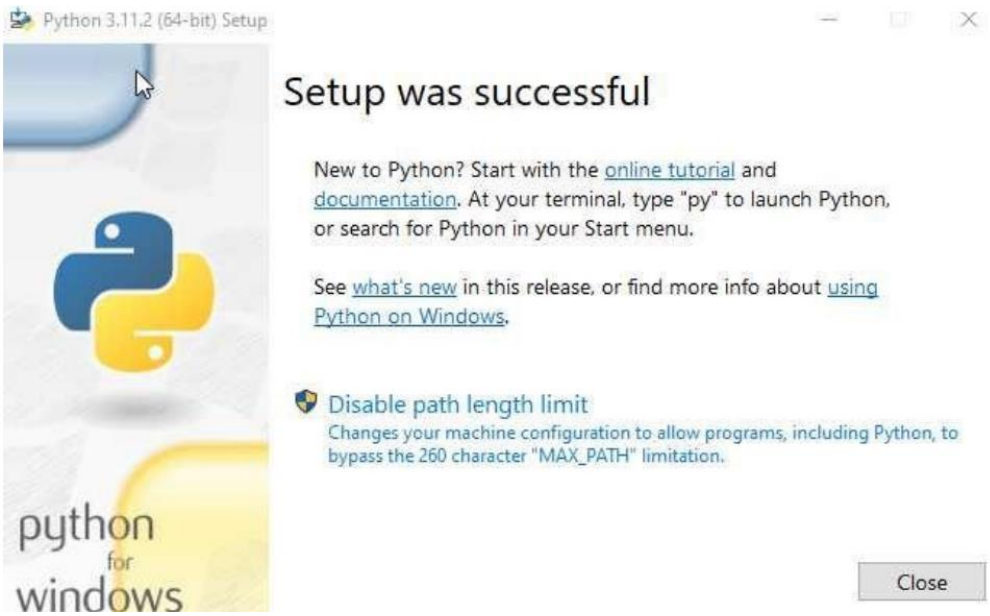
W ramach realizacji pracy laboratoryjnej opracowana została prosta aplikacja w języku Python, służąca do rozwiązywania różnego rodzaju zadań z programowania liniowego. Do wykonania implementacji wykorzystane zostało oprogramowanie Jupyter Notebook oraz różnego rodzaju biblioteki wymagane do pełnej funkcjonalności aplikacji. Przygotowując środowisko pracy należało rozpocząć od instalacji oprogramowania Python na komputerze. Wspomniane oprogramowanie należy pobrać ze strony: https://www.python.org/downloads/



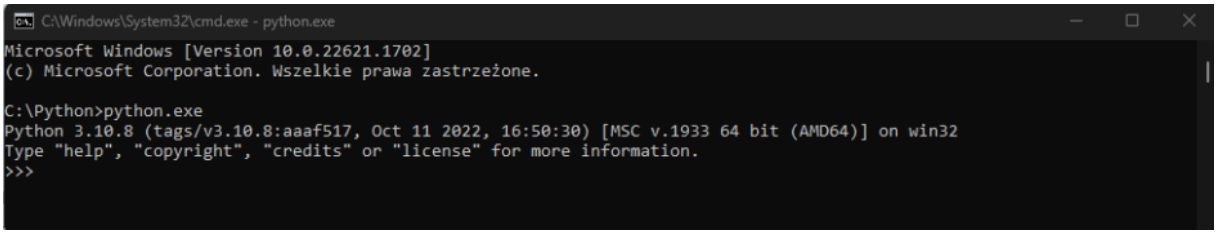
Następnie użytkownik musi przejść przez proces instalacyjny wybranej wersji oprogramowania.


Python 3.11.2 (64-bit) Setup
Optional Features
Documentation
Installs the Python documentation files.
pip
Installs pip, which can download and install other Python packages.
tcl/tk and IDLE
Installs tkinter and the IDLE development environment.
Python test suite
Installs the standard library test suite.
py launcher
for all users (requires admin privileges)
Installs the global 'py' launcher to make it easier to start Python.
python
for
windows
Back
Next
Cancel

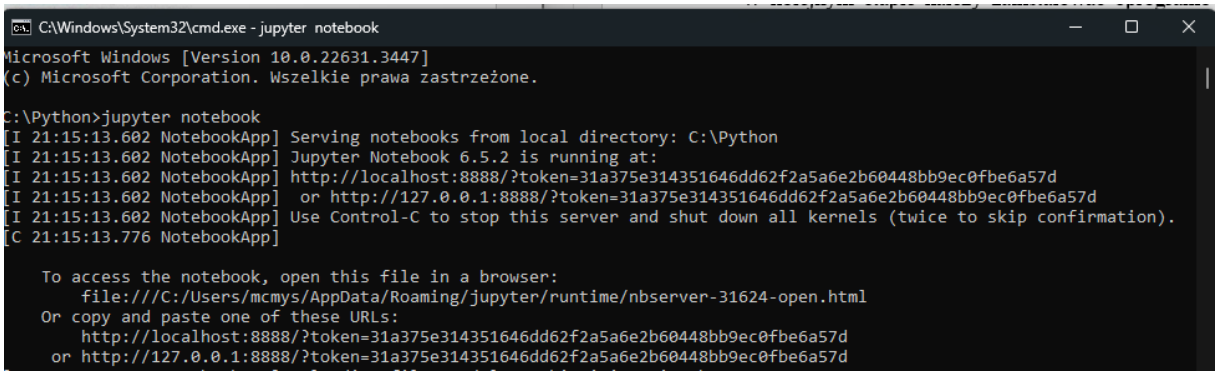

Python 3.11.2 (64-bit) Setup
Advanced Options
Install Python 3.11 for all users
Associate files with Python (requires the 'py' launcher)
Create shortcuts for installed applications
Add Python to environment variables
Precompile standard library
Download debugging symbols
Download debug binaries (requires VS 2017 or later)
Customize install location
C:\Python
Browse
python
for
windows
Back
Install
Cancel


Python 3.11.2 (64-bit) Setup
Setup was successful
New to Python? Start with the online tutorial and
documentation. At your terminal, type "py" to launch Python,
or search for Python in your Start menu.
See what's new in this release, or find more info about using
Python on Windows.
Disable path length limit
Changes your machine configuration to allow programs, including Python, to
bypass the 260 character "MAX_PATH" limitation.
python
for
windows
Close

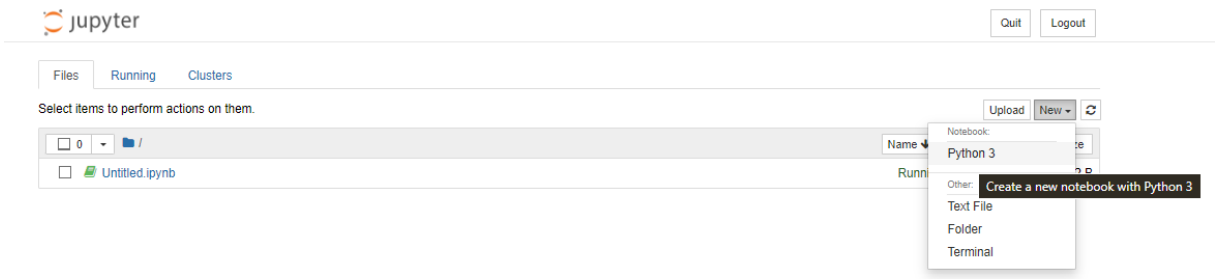
Kolejny etap to przejście pod wskazaną podczas instalacji ścieżkę z pomocą wiersza poleceń bądź PowerShell. Użytkownik wpisuje komendę python.exe.



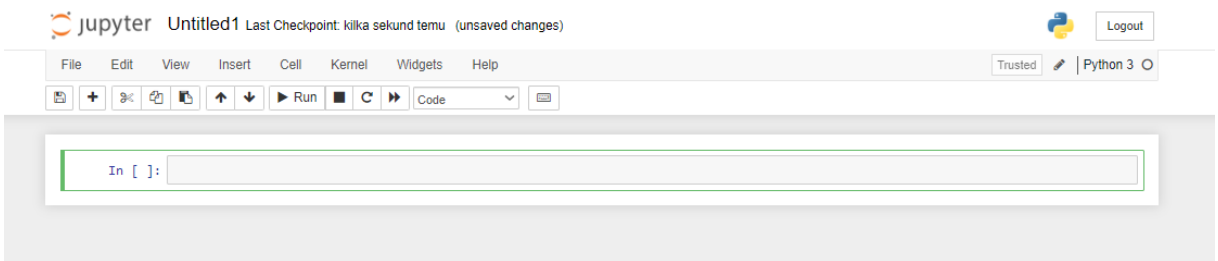
Następnie należy dokonać instalacji Jupyter Notebook za pomocą wiersza poleceń CMD, bądź PowerShell. W tym celu wpisujemy komendę pip install notebook. Kolejny krok to odpalenie zainstalowanego oprogramowania za pomocą komendy jupyter notebook oraz rozpoczęcie pracy z interfejsem oprogramowania lokalnego.



W domyślnie zainstalowanej przeglądarce uruchomiony zostanie interfejs Jupyter Notebook. Należy utworzyć plik, w którym w przyszłości opracowana zostanie aplikacja.



Po wybraniu opcji Python 3, przeglądarka uruchomi nową zakładkę, w której wyświetlony zostanie interfejs Jupyter Notebook odpowiedzialny za tworzenie programów.



Podczas implementacji wykorzystane zostały biblioteki, dzięki którym oprogramowanie jest w pełni funkcjonalne, a w ich skład weszły: biblioteka tkinter i jej moduły (ttk, messagebox, filedialog) – są używane do tworzenia interfejsu użytkownika, zarządzania zakładkami, oknami dialogowymi i obsługą plików, pulp – jest używany do definiowania problemów optymalizacyjnych liniowych, zmiennych i ograniczeń, matplotlib oraz matplotlib.pyplot – służą do generowania różnych rodzajów wykresów, zarządzania ich wyświetlaniem w interfejsie graficznym, numpy – wykorzystywany do operacji na tablicach, które są niezbędne przy tworzeniu wykresów, np. dla danych wejściowych ograniczeń w formie nierówności oraz matplotlib.colors, matplotlib.lines, matplotlib.patches – służą do niestandardowej stylizacji wykresów, zarządzania kolorami, liniami i kształtami na wykresach. Ostatania z bibliotek to biblioteka SymPy.

SymPy jest biblioteką języka Python przeznaczoną do symbolicznych obliczeń matematycznych. Celem projektu SymPy jest stworzenie pełnoprawnego systemu algebry komputerowej (ang. computer algebra system, CAS), przy jednoczesnym zachowaniu prostoty kodu, co ułatwia zrozumienie i rozbudowę. SymPy jest napisane całkowicie w Pythonie i nie wymaga zewnętrznych bibliotek, co czyni ją łatwą w instalacji i użyciu. SymPy pozwala na wykonywanie szerokiego zakresu operacji matematycznych, od prostych obliczeń symbolicznych po bardziej złożone zagadnienia, takie jak różniczkowanie, całkowanie, rozwiązywanie równań oraz manipulowanie wyrażeniami algebraicznymi. Możliwe jest także tworzenie własnych funkcji, rozszerzanie istniejących możliwości biblioteki oraz generowanie kodu w różnych językach programowania na podstawie symbolicznych wyrażeń.

Aby oprogramowanie mogło w pełni funkcjonować należy zainstalować wspomniane biblioteki za pomocą wiersza polecań wpisując następujące polecenia:

```
pip install pulp
pip install matplotlib
pip install numpy
pip install sympy
```

Biblioteka tkinter zazwyczaj jest dostarczana wraz z instalacją Pythona, jednak w przypadku systemu operacyjnego Linux należy zainstalować go osobno za pomocą polecenia sudo apt-get install python3-tk.

W późniejszym etapie do uruchomienia opracowanej aplikacji wykorzystany zostanie wiersz poleceń oraz polecenie python aplikacja.py

5. Omówienie kodu programu

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter import filedialog
from pulp import LpVariable, LpProblem, lpSum, LpMinimize, LpMaximize
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.backends.backend_pdf import PdfPages
import numpy as np
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.lines import Line2D
import matplotlib.colors as mcolors
from matplotlib.patches import Polygon
```

Przedstawiony powyżej kod korzysta z różnych bibliotek w Pythonie, aby umożliwić tworzenie graficznego interfejsu użytkownika oraz przetwarzanie i wizualizację danych. Zacznijmy od Tkinter, który jest standardową biblioteką GUI w Pythonie. Importowane moduły z tej biblioteki, takie jak tk, ttk, messagebox oraz filedialog, pozwalają na budowanie okien aplikacji, wykorzystywanie zaawansowanych widgetów, wyświetlanie komunikatów oraz obsługę plików przez użytkownika. Kolejna importowana biblioteka to pulp, która jest używana do modelowania liniowego programowania. Umożliwia definicję problemów minimalizacji lub maksymalizacji z ograniczeniami i zmiennymi decyzyjnymi. Do wizualizacji danych służą matplotlib i jego moduły, takie jak pyplot, FigureCanvasTkAgg czy PdfPages, które pozwalają na tworzenie zaawansowanych wykresów, integrację wykresów z interfejsem Tkinter oraz zapis wykresów do plików PDF. Moduły numpy oraz dodatkowe elementy matplotlib, takie jak LinearSegmentedColormap czy Polygon, wspierają manipulację danymi numerycznymi oraz dostosowywanie wyglądu wykresów, na przykład poprzez definiowanie kolorów czy tworzenie złożonych kształtów na wykresach. Wszystkie te narzędzia razem tworzą potężną platformę do tworzenia aplikacji, które nie tylko przetwarzają dane, ale również prezentują je w przejrzysty i interaktywny sposób.

```
root = tk.Tk()
root.title("Uniwersalne narzędzie do programowania liniowego")
root.geometry("800x600")
```

```
zmienne = {}
ograniczenia = []
problem = None
kierunek_celu = tk.StringVar(value="Minimize")

zeszyt = ttk.Notebook(root)
zeszyt.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

zakladka_zmienne = tk.Frame(zeszyt)
zakladka_ograniczenia = tk.Frame(zeszyt)
zakladka_funkcja_celu = tk.Frame(zeszyt)

zeszyt.add(zakladka_zmienne, text="Zmienne")
zeszyt.add(zakladka_ograniczenia, text="Ograniczenia")
zeszyt.add(zakladka_funkcja_celu, text="Funkcja Celu")
```

Przedstawiony powyżej fragment kodu inicjuje aplikację graficzną w Pythonie przy użyciu biblioteki Tkinter, która służy jako narzędzie do programowania liniowego. Rozpoczyna od utworzenia głównego okna aplikacji, ustawiając jego tytuł na "Uniwersalne narzędzie do programowania liniowego" oraz określając jego wymiary na 800x600 pikseli. W dalszej kolejności, kod tworzy struktury danych do przechowywania zmiennych, ograniczeń i problemu programowania liniowego. Zmienna kierunek_celu, przechowująca informacje o celu optymalizacji (minimalizacja lub maksymalizacja), jest inicjowana jako obiekt StringVar z domyślną wartością "Minimize". Dodatkowo, kod implementuje kontrolki zakładek dzięki widżetowi Notebook z modułu ttk, które są dodane do głównego okna. Widżet Notebook jest upakowany z opcjami wypełnienia i rozwijania, aby zajmował dostępną przestrzeń, z dodatkowymi marginesami wewnątrz okna. Dla każdej z funkcji programu: zarządzania zmiennymi, ograniczeniami oraz funkcją celu, utworzone są osobne ramki (Frame), które są następnie dodawane jako oddzielne zakładki o odpowiednio oznaczonych tytułach: "Zmienne", "Ograniczenia" oraz "Funkcja Celu". Takie rozmieszczenie elementów umożliwia użytkownikowi łatwą nawigację i interakcję z poszczególnymi segmentami funkcjonalności aplikacji.

```
def inicjalizuj_problem():
    global problem
    kierunek = LpMinimize if kierunek_celu.get() == "Minimize" else
LpMaximize
    problem = LpProblem("Problem Optymalizacyjny", kierunek)

def inicjalizuj_zakladki():
```

```
    inicjalizuj_zakladke_zmienne()
    inicjalizuj_zakladke_ograniczenia()
    inicjalizuj_zakladke_funkcja_celu()

def inicjalizuj_zakladke_zmienne():
    wyczysc_ramke(zakladka_zmienne)
    tk.Label(zakladka_zmienne, text="Ilość zmiennych:").pack(side=tk.TOP,
fill=tk.X)
    global liczba_zmiennych_entry
    liczba_zmiennych_entry = tk.Entry(zakladka_zmienne)
    liczba_zmiennych_entry.pack(side=tk.TOP, fill=tk.X)
    tk.Button(zakladka_zmienne, text="Generuj pola",
command=utworz_pola_zmiennych).pack(side=tk.TOP)

def inicjalizuj_zakladke_ograniczenia():
    wyczysc_ramke(zakladka_ograniczenia)
    tk.Label(zakladka_ograniczenia, text="Ilość
ograniczeń:").pack(side=tk.TOP, fill=tk.X)
    global liczba_ograniczen_entry
    liczba_ograniczen_entry = tk.Entry(zakladka_ograniczenia)
    liczba_ograniczen_entry.pack(side=tk.TOP, fill=tk.X)
    tk.Button(zakladka_ograniczenia, text="Generuj pola",
command=utworz_pola_ograniczen).pack(side=tk.TOP)

def inicjalizuj_zakladke_funkcja_celu():
    wyczysc_ramke(zakladka_funkcja_celu)
    tk.Label(zakladka_funkcja_celu, text="Wyrażenie funkcji
celu:").pack(side=tk.TOP, fill=tk.X)
    global pole_funkcji_celu
    pole_funkcji_celu = tk.Entry(zakladka_funkcja_celu)
    pole_funkcji_celu.pack(side=tk.TOP, fill=tk.X)
    global kierunek_celu
    kierunek_celu = ttk.Combobox(zakladka_funkcja_celu, values=["Minimize",
"Maximize"], state="readonly")
    kierunek_celu.set("Minimize")
    kierunek_celu.pack(side=tk.TOP, fill=tk.X)
    tk.Button(zakladka_funkcja_celu, text="Ustaw funkcję celu i kierunek",
command=ustaw_funkcje_celu).pack(side=tk.TOP)
```

Przedstawiony fragment kodu realizuje funkcjonalność inicjalizacji i konfiguracji różnych komponentów interfejsu użytkownika w aplikacji do programowania liniowego przy użyciu Tkintera. Proces rozpoczyna się od definicji funkcji inicjalizuj_problem, która tworzy nowy obiekt problemu optymalizacyjnego z biblioteki PuLP, określając czy problem ma być zminimalizowany czy zmaksymalizowany w zależności od wyboru użytkownika. Kierunek

optymalizacji jest pobierany z kontrolki kierunek_celu. Następnie, funkcja inicjalizuj_zakladki odpowiada za wywołanie funkcji inicjujących dla każdej z trzech zakładek:

- inicjalizuj_zakladke_zmienne,
- inicjalizuj_zakladke_ograniczenia,
- inicjalizuj_zakladke_funkcja_celu.

Każda z tych funkcji konfiguruje odpowiednią ramkę dla swojej zakładki, czyści poprzednią zawartość ramki, dodaje odpowiednie elementy GUI takie jak etykiety, pola tekstowe i przyciski. Na przykład w zakładce zmiennych dodawane są pola umożliwiające wprowadzenie liczby zmiennych i przycisk do generowania odpowiedniej liczby pól do wprowadzania. Podobne działania są przeprowadzane dla zakładek ograniczeń i funkcji celu, gdzie użytkownik może wprowadzić odpowiednio liczby ograniczeń oraz formułę matematyczną określającą funkcję celu. W zakładce funkcji celu dodatkowo umieszczony jest rozwijany wybór (combobox) umożliwiający wybór czy cel ma być zminimalizowany, czy zmaksymalizowany, z możliwością ustawienia domyślnego wyboru na 'Minimize'. Wprowadzone wartości mogą być następnie użyte do zdefiniowania i rozwiązania problemu optymalizacyjnego. Wszystkie te komponenty i ich interakcje są kluczowe dla funkcjonalności aplikacji, umożliwiając użytkownikowi definiowanie i manipulowanie problemem programowania liniowego w sposób interaktywny.

```
def utworz_pola_zmiennych():
    try:
        liczba_zmiennych = int(liczba_zmiennych_entry.get())
    except ValueError:
        messagebox.showerror("Błąd", "Wprowadź poprawną liczbę zmiennych.")
        return

    wyczysc_ramke(zakladka_zmienne)
    inicjalizuj_zakladke_zmienne()
    for i in range(liczba_zmiennych):
        utworz_pole_zmiennej(zakladka_zmienne, i)

def utworz_pole_zmiennej(rodzic, indeks):
    ramka_zmiennej = tk.Frame(rodzic)
    ramka_zmiennej.pack(fill=tk.X, padx=5, pady=5)
    tk.Label(ramka_zmiennej, text=f"Zmienna {indeks+1}:").pack(side=tk.LEFT)

    nazwa_entry = tk.Entry(ramka_zmiennej, width=5)
    nazwa_entry.pack(side=tk.LEFT, padx=2)

    dolna_skala = tk.Scale(ramka_zmiennej, from_=0, to=999999,
orient=tk.HORIZONTAL)
```

```
    dolna_skala.pack(side=tk.LEFT, padx=2)

    gorna_skala = tk.Scale(ramka_zmiennej, from_=0, to=999999,
orient=tk.HORIZONTAL)
    gorna_skala.pack(side=tk.LEFT, padx=2)

    combo_kategoria = ttk.Combobox(ramka_zmiennej, values=["Continuous",
"Integer"], state="readonly")
    combo_kategoria.set("Continuous")
    combo_kategoria.pack(side=tk.LEFT, padx=2)

    przycisk_zapisz = tk.Button(ramka_zmiennej, text="Zapisz",
command=lambda ne=nazwa_entry, le=dolna_skala, ue=gorna_skala,
cc=combo_kategoria: zapisz_zmienna(ne, le, ue, cc))
    przycisk_zapisz.pack(side=tk.LEFT, padx=2)

    def zapisz_zmienna(nazwa_entry, dolna_skala, gorna_skala,
combo_kategoria):
    nazwa = nazwa_entry.get()
    dolna = dolna_skala.get()
    gorna = gorna_skala.get()
    kategoria = combo_kategoria.get()

    if nazwa == "":
        messagebox.showerror("Błąd", "Nazwa zmiennej nie może być pusta.")
        return
    if nazwa in zmienne:
        messagebox.showerror("Błąd", f"Zmienna '{nazwa}' już istnieje.")
        return

    zmienna = LpVariable(nazwa, lowBound=dolna, upBound=gorna,
cat=kategoria)
    zmienne[nazwa] = zmienna
    aktualizuj_informacje_o_stanie(f"Zmienna dodana: {nazwa} [{dolna},
{gorna}] {kategoria}")
```

Przedstawiony kod skupia się na dynamicznym tworzeniu interfejsu użytkownika dla definiowania zmiennych w problemie programowania liniowego, zaimplementowanym w Tkinter. Proces rozpoczyna się od definicji funkcji utworz_pola_zmiennych, która próbuje pobrać liczbę zmiennych z wprowadzonego przez użytkownika tekstu. Jeżeli wprowadzona wartość nie jest liczbą, wyświetlany jest komunikat o błędzie. W przypadku poprawnej wartości, ramka zakładki dla zmiennych jest czyszczona i ponownie inicjalizowana, a następnie dla każdej zmiennej tworzone są specyficzne pola. Każde pole zmiennej jest tworzone przez funkcję utworz_pole_zmiennej, która przyjmuje jako argumenty rodzica (kontener, gdzie elementy będą umieszczane) oraz indeks zmiennej. W ramce dla zmiennej umieszczane są: etykieta z nazwą zmiennej, pole tekstowe do wprowadzenia nazwy, dwie suwaki (skale) do określenia dolnego i górnego ograniczenia wartości zmiennej, rozwijalne menu do wyboru typu

zmiennej (ciągła lub całkowita), oraz przycisk do zapisu zdefiniowanej zmiennej. Funkcja zapisz_zmienna, przypisana do przycisku 'Zapisz', odpowiada za weryfikację i zapisanie każdej zmiennej. Sprawdza, czy nazwa zmiennej nie jest pusta oraz czy zmienna o tej samej nazwie już nie istnieje w słowniku zmienne. Jeśli te warunki są spełnione, tworzy nową zmienną za pomocą klasy LpVariable z biblioteki PuLP, określając jej nazwę, dolne i górne ograniczenie oraz kategorię, a następnie zapisuje ją w słowniku. W przypadku dodania zmiennej, aktualizuje informacje o stanie, informując użytkownika o dodanej zmiennej. Kod ten łączy funkcjonalność programowania liniowego z interaktywnym interfejsem użytkownika, co pozwala na łatwe definiowanie i zarządzanie zmiennymi w modelu matematycznym.

```
def utworz_pola_ograniczen():
    try:
        liczba_ograniczen = int(liczba_ograniczen_entry.get())
    except ValueError:
        messagebox.showerror("Błąd", "Wprowadź poprawną liczbę ograniczeń.")
        return

    wyczysc_ramke(zakladka_ograniczenia)
    inicjalizuj_zakladke_ograniczenia()
    for i in range(liczba_ograniczen):
        utworz_pole_ograniczenia(zakladka_ograniczenia, i)

def utworz_pole_ograniczenia(rodzic, indeks):
    ramka_ograniczenia = tk.Frame(rodzic)
    ramka_ograniczenia.pack(fill=tk.X, padx=5, pady=2)

    etykieta_ograniczenia = tk.Label(ramka_ograniczenia, text=f"Ograniczenie
{indeks+1}:")
    etykieta_ograniczenia.pack(side=tk.LEFT)

    pole_ograniczenia = tk.Entry(ramka_ograniczenia)
    pole_ograniczenia.pack(side=tk.LEFT, fill=tk.X, expand=True)

    przycisk_zapisz = tk.Button(ramka_ograniczenia, text="Zapisz",
command=lambda ce=pole_ograniczenia: zapisz_ograniczenie(ce))
    przycisk_zapisz.pack(side=tk.LEFT, padx=5)

def zapisz_ograniczenie(pole_ograniczenia):
    ograniczenie = pole_ograniczenia.get()
    if ograniczenie:
        ograniczenia.append(ograniczenie)
        aktualizuj_informacje_o_stanie(f"Ograniczenie dodane:
{ograniczenie}")
```

Przedstawiony powyżej kod, służy do zarządzania ograniczeniami w interfejsie użytkownika aplikacji do programowania liniowego, zrealizowanej przy użyciu Tkinter. Proces rozpoczyna się od definicji funkcji utworz_pola_ograniczen, która próbuje pobrać liczbę ograniczeń wprowadzoną przez użytkownika. Jeśli wartość ta nie jest prawidłową liczbą, aplikacja informuje użytkownika o błędzie za pomocą okna dialogowego. Po poprawnym wprowadzeniu, funkcja czyści ramkę przeznaczoną dla zakładki ograniczeń, inicjuje ją na nowo i dla każdej wprowadzonej liczby ograniczeń tworzy odpowiednie pola. Każde pole ograniczenia jest tworzone przez funkcję utworz_pole_ograniczenia, która jako argumenty przyjmuje rodzica (kontener, w którym elementy będą umieszczone) oraz indeks ograniczenia. W ramce dla każdego ograniczenia umieszczane są następujące elementy GUI: etykieta z numerem ograniczenia, pole tekstowe umożliwiające wprowadzenie wyrażenia ograniczenia, oraz przycisk "Zapisz", który jest przypisany do funkcji zapisującej. Funkcja zapisz_ograniczenie jest wywoływana przez przycisk i służy do zapisania wprowadzonego ograniczenia w globalnej liście ograniczenia, pod warunkiem, że pole nie jest puste. Po zapisaniu, stan aplikacji jest aktualizowany, informując użytkownika o dodanym ograniczeniu. Ten mechanizm pozwala na dynamiczne dodawanie ograniczeń do modelu optymalizacyjnego, co jest kluczowe dla efektywnego zarządzania i manipulowania problemami programowania liniowego w aplikacji.

```
def ustaw_funkcje_celu():
    global problem
    try:
        # Wyciągnięcie wyrażenia z pola funkcji celu
        wyrazenie = pole_funkcji_celu.get()
        if not wyrazenie:
            raise ValueError("Funkcja celu nie może być pusta.")

        for nazwa_zmiennej in zmienne.keys():
            wyrazenie = wyrazenie.replace(nazwa_zmiennej,
f'zmienne["{nazwa_zmiennej}"]')

        inicjalizuj_problem()

        problem += eval(wyrazenie), "Funkcja Celu"

        aktualizuj_informacje_o_stanie(f"Funkcja celu ustawiona:
{pole_funkcji_celu.get()} [{kierunek_celu.get()}]")
    except Exception as e:
        messagebox.showerror("Błąd", f"Niepoprawne wyrażenie funkcji celu:
{e}")
```

Powyższy kod opisuje funkcję ustaw_funkcje_celu w aplikacji wykorzystującej Tkinter do zarządzania problemem optymalizacji liniowej. Funkcja ta służy do konfiguracji funkcji celu w modelu programowania liniowego. Początkowo funkcja deklaruje zmienną problem jako globalną, co umożliwia modyfikację tego obiektu zdefiniowanego poza zakresem lokalnym funkcji. Następnie próbuje pobrać wyrażenie matematyczne wpisane przez użytkownika do pola tekstowego pole_funkcji_celu. Jeżeli pole to jest puste, generowany jest błąd, informujący, że funkcja celu nie może być pusta. W kolejnym kroku funkcja przeszukuje wyrażenie w celu zastąpienia każdej nazwy zmiennej występującej w słowniku zmienne odpowiednim odwołaniem do tej zmiennej w słowniku, co jest kluczowe dla prawidłowego przetworzenia wyrażenia przez funkcję eval. To przygotowanie wyrażenia jest niezbędne, aby eval mogło poprawnie zinterpretować i przeliczyć wyrażenie matematyczne funkcji celu z wykorzystaniem aktualnych wartości zmiennych. Po przygotowaniu wyrażenia, funkcja inicjalizuj_problem jest wywoływana w celu ponownej inicjalizacji problemu optymalizacji z aktualnym kierunkiem celu (minimalizacja lub maksymalizacja). Następnie dodawane jest wyrażenie funkcji celu do problemu optymalizacyjnego, a jego nazwa jest określana jako "Funkcja Celu". W razie wystąpienia błędów podczas procesu, np. błędnych odwołań lub składni w wyrażeniu funkcji celu, użytkownik zostaje o tym poinformowany poprzez okno dialogowe z odpowiednim komunikatem błędu. Jeśli wszystko przebiegnie pomyślnie, informacja o ustawieniu funkcji celu wraz z wybranym kierunkiem celu jest wyświetlana, co pozwala na śledzenie stanu i postępów konfiguracji problemu optymalizacyjnego. Funkcja ta stanowi kluczowy element interfejsu użytkownika aplikacji do programowania liniowego, umożliwiając użytkownikowi definiowanie i modyfikowanie funkcji celu, co jest centralnym aspektem rozwiązywania problemów optymalizacyjnych.

```
def rozwiaz_problem():
    try:
        global problem, ograniczenia, zmienne
        inicjalizuj_problem()

        wyrazenie = pole_funkcji_celu.get()
        for nazwa_zmiennej in zmienne.keys():
            wyrazenie = wyrazenie.replace(nazwa_zmiennej,
f'zmienne["{nazwa_zmiennej}"]')
        problem += eval(wyrazenie), "Funkcja Celu"

        for ograniczenie_str in ograniczenia:
            wyrazenie_ograniczenia = ograniczenie_str
            for nazwa_zmiennej in zmienne.keys():
```

```
                wyrazenie_ograniczenia =
wyrazenie_ograniczenia.replace(nazwa_zmiennej,
f'zmienne["{nazwa_zmiennej}"]')
            problem += eval(wyrazenie_ograniczenia), ""

        problem.solve()
        wynik = f"Status: {problem.status}, Wartość:
{problem.objective.value()}\n"
        for v in zmienne.values():
            wynik += f"{v.name}: {v.varValue}\n"
        aktualizuj_informacje_o_stanie(wynik)
    except Exception as e:
        messagebox.showerror("Błąd", f"Problem z rozwiązaniem: {e}")
```

Powyższy kod przedstawia funkcję rozwiaz_problem, która jest kluczowym elementem aplikacji do programowania liniowego, służącej do rozwiązywania zdefiniowanych matematycznie problemów optymalizacyjnych z wykorzystaniem interfejsu użytkownika stworzonego w Tkinter. Funkcja ta odpowiada za pełny proces rozwiązania problemu, od jego inicjalizacji, przez konfigurację funkcji celu, aktualizację ograniczeń, aż do obliczenia rozwiązania i prezentacji wyników. Na początku funkcja zapewnia, że zmienne problem, ograniczenia i zmienne są traktowane jako globalne, co umożliwia ich modyfikację wewnątrz funkcji. Następnie następuje ponowna inicjalizacja problemu optymalizacyjnego, co jest konieczne do usunięcia wszelkich poprzednio ustawionych funkcji celu lub ograniczeń. Po reinicjalizacji, funkcja celu jest ponownie konfigurowana. Wyrażenie funkcji celu jest pobierane z interfejsu użytkownika, a następnie przetwarzane tak, aby odwołania do zmiennych były odpowiednio zaktualizowane i mogły być prawidłowo przetworzone przez funkcję eval, która interpretuje wyrażenie jako kod Pythona. Po skonfigurowaniu funkcji celu, podobny proces jest przeprowadzany dla ograniczeń. Każde ograniczenie jest aktualizowane o odniesienia do zmiennych i dodawane do problemu. Kiedy wszystkie elementy problemu są skonfigurowane, funkcja solve() z biblioteki PuLP jest wywoływana w celu znalezienia optymalnego rozwiązania problemu. Po rozwiązaniu problemu, funkcja generuje string zawierający status rozwiązania, wartość funkcji celu oraz wartości poszczególnych zmiennych decyzyjnych. Wyniki te są następnie wyświetlane w interfejsie użytkownika za pomocą funkcji aktualizuj_informacje_o_stanie, co umożliwia użytkownikowi szybką weryfikację wyników. W przypadku wystąpienia błędów w procesie rozwiązywania, są one prezentowane użytkownikowi przez okno dialogowe z odpowiednim komunikatem, co umożliwia identyfikację i rozwiązanie problemu. Ta funkcja jest zatem centralnym elementem aplikacji,

umożliwiającym efektywne zarządzanie i rozwiązywanie problemów optymalizacyjnych w sposób interaktywny.

```
def aktualizuj_informacje_o_stanie(wiadomosc):
    info_stanu.config(state='normal')
    info_stanu.insert(tk.END, wiadomosc + "\n")
    info_stanu.config(state='disabled')
```

Powyższy kod przedstawia funkcję aktualizuj_informacje_o_stanie, która jest używana do aktualizacji informacji o stanie procesu w aplikacji GUI stworzonej za pomocą Tkinter. Funkcja ta ma za zadanie dostarczać użytkownikowi bieżące informacje o działaniach wykonywanych przez aplikację, w tym wynikach obliczeń czy ewentualnych błędach. Funkcja przyjmuje jeden argument, wiadomosc, który jest tekstem przeznaczonym do wyświetlenia w komponencie interfejsu użytkownika. W praktyce, info_stanu jest prawdopodobnie kontrolką typu Text lub podobnym elementem Tkinter służącym do prezentacji tekstowej, gdzie użytkownicy mogą obserwować logi czy statusy operacji. Na początku działania funkcji, stan widgetu info_stanu jest ustawiany na 'normal', co umożliwia modyfikację jego zawartości. Następnie, nowa wiadomość jest dodawana do istniejącej zawartości widgetu. Działanie to realizowane jest przez metodę insert, która dodaje przekazany tekst wiadomosc na końcu (określone przez tk.END) obecnej treści widgetu. Po dodaniu wiadomości, funkcja dodaje również znak nowej linii, co zapewnia, że każda wiadomość jest wyraźnie oddzielona od poprzedniej. Po zaktualizowaniu treści, stan widgetu info_stanu jest ponownie ustawiany na 'disabled'. Ten krok zapobiega dalszej edycji treści przez użytkownika, co jest standardową praktyką w przypadku komponentów wyświetlających tylko logi lub informacje systemowe – użytkownik może je przeglądać, ale nie modyfikować.

```
def generuj_wykres_kolowy():
    etykiety = [v.name for v in zmienne.values()]
    wielkosci = [v.varValue for v in zmienne.values()]
    if wielkosci:
        fig, ax = plt.subplots()
        ax.pie(wielkosci, labels=etykiety, autopct='%1.1f%%',
startangle=140)
        ax.axis('equal')
        pokaz_wykres(fig)
    else:
        messagebox.showinfo("Informacja", "Brak danych do wykresu
kołowego.")
```

```
def generuj_wykres_liniowy():
    etykiety = [v.name for v in zmienne.values()]
    wartosci = [v.varValue for v in zmienne.values()]
    if wartosci:
        fig, ax = plt.subplots()
        ax.plot(etykiety, wartosci, marker='o')
        ax.set_xlabel('Zmienne')
        ax.set_ylabel('Wartości')
        ax.set_title('Wykres wartości zmiennych')
        pokaz_wykres(fig)
    else:
        messagebox.showinfo("Informacja", "Brak danych do wykresu
liniowego.")

def generuj_wykres_slupkowy(zapisz_pdf=None):
    etykiety = [v.name for v in zmienne.values()]
    wartosci = [v.varValue for v in zmienne.values()]
    if wartosci:
        fig, ax = plt.subplots()
        ax.bar(etykiety, wartosci, color='blue')
        ax.set_xlabel('Zmienne')
        ax.set_ylabel('Wartości')
        ax.set_title('Wykres słupkowy wartości zmiennych')
        if zapisz_pdf:
            zapisz_pdf.savefig(fig)
            plt.close(fig)
        else:
            pokaz_wykres(fig)
    else:
        messagebox.showinfo("Informacja", "Brak danych do wykresu
słupkowego.")


def pokaz_wykres(fig):
    okno_wykresu = tk.Toplevel(root)
    okno_wykresu.title("Wykres")
    plotno = FigureCanvasTkAgg(fig, master=okno_wykresu)
    plotno.draw()
    plotno.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)
```

Powyższy kod przedstawia zestaw funkcji do generowania różnych rodzajów wykresów w aplikacji wykorzystującej Python i biblioteki matplotlib oraz Tkinter, które służą do wizualizacji danych z problemów optymalizacyjnych rozwiązanych za pomocą programowania liniowego. Zdefiniowane funkcje odpowiadają za tworzenie wykresów kołowych, liniowych oraz słupkowych, umożliwiając użytkownikom graficzne przedstawienie rozwiązanych

wartości zmiennych. Funkcja generuj_wykres_kolowy generuje wykres kołowy, który ilustruje procentowy udział każdej zmiennej w całości, wykorzystując etykiety i wartości zmiennej. Wykres jest tworzony tylko w przypadku, gdy istnieją dane do wyświetlenia. W przeciwnym razie użytkownik otrzymuje komunikat o braku danych. Funkcja generuj_wykres_liniowy tworzy wykres liniowy, pokazujący wartości poszczególnych zmiennych na osi, z etykietami odpowiadającymi nazwom zmiennych. Podobnie jak w przypadku wykresu kołowego, gdy brakuje danych, użytkownik jest informowany o tej sytuacji. Funkcja generuj_wykres_slupkowy tworzy wykres słupkowy, gdzie każda zmienna jest reprezentowana przez słupek, którego wysokość odpowiada wartości zmiennej. Dodatkowo funkcja ta oferuje możliwość zapisania wykresu do pliku PDF, jeśli jest to wymagane przez użytkownika. Ostatnia funkcja, pokaz_wykres, jest odpowiedzialna za wyświetlanie dowolnego wygenerowanego wykresu w nowym oknie interfejsu użytkownika. Wykres jest umieszczany w oknie za pomocą widgetu FigureCanvasTkAgg, co umożliwia integrację grafiki matplotlib z interfejsem Tkinter. Okno to jest tworzone jako podrzędne względem głównego okna aplikacji, co pozwala na wygodne zarządzanie prezentacją wyników analizy. Wszystkie te funkcje razem tworzą kompleksowe narzędzie do wizualizacji danych, które jest integralną częścią aplikacji do programowania liniowego, pozwalając użytkownikom na łatwe i efektywne analizowanie wyników ich modeli optymalizacyjnych.

```
def dodaj_przyciski_wykresow():
    ramka_przyciskow = tk.Frame(root)
    ramka_przyciskow.pack(side=tk.BOTTOM, fill=tk.X, pady=15)

    przycisk_rozwiaz = tk.Button(ramka_przyciskow, text="Rozwiąż problem",
command=rozwiaz_problem)
    przycisk_rozwiaz.pack(side=tk.LEFT, padx=15)

    przycisk_wykres_nierownosci = tk.Button(ramka_przyciskow, text="Wykres
nierówności", command=lambda: generuj_wykres_nierownosci(ograniczenia))
    przycisk_wykres_nierownosci.pack(side=tk.LEFT, padx=15)

    przycisk_wykres_kolowy = tk.Button(ramka_przyciskow, text="Wykres
kołowy", command=generuj_wykres_kolowy)
    przycisk_wykres_kolowy.pack(side=tk.LEFT, padx=15)

    przycisk_wykres_liniowy = tk.Button(ramka_przyciskow, text="Wykres
liniowy", command=generuj_wykres_liniowy)
    przycisk_wykres_liniowy.pack(side=tk.LEFT, padx=15)

    przycisk_wykres_slupkowy = tk.Button(ramka_przyciskow, text="Wykres
słupkowy", command=generuj_wykres_slupkowy)
    przycisk_wykres_slupkowy.pack(side=tk.LEFT, padx=15)
```

```
    przycisk_zapisz_pdf = tk.Button(ramka_przyciskow, text="Zapisz do PDF",
command=zapisz_do_pdf)
    przycisk_zapisz_pdf.pack(side=tk.LEFT, padx=15)

def zapisz_do_pdf():
    nazwa_pliku_pdf = filedialog.asksaveasfilename(defaultextension=".pdf",
filetypes=[("Pliki PDF", "*.pdf")])
    if nazwa_pliku_pdf:
        with PdfPages(nazwa_pliku_pdf) as pdf:
            # Zapisz wykres kołowy
            if any(v.varValue for v in zmienne.values()):
                generuj_wykres_kolowy(zapisz_pdf=pdf)
            # Zapisz wykres liniowy
            if any(v.varValue for v in zmienne.values()):
                generuj_wykres_liniowy(zapisz_pdf=pdf)
            # Zapisz wykres słupkowy
            if any(v.varValue for v in zmienne.values()):
                generuj_wykres_slupkowy(zapisz_pdf=pdf)
            # Zapisz wykres nierówności
            if ograniczenia:
                generuj_wykres_nierownosci(ograniczenia, pokaz_okno=False,
zapisz_pdf=pdf)
            # Zapisz tekst z dolnej części okna
            tekst_info_stanu = info_stanu.get("1.0", tk.END)
            fig = plt.figure(figsize=(8, 0.5))
            plt.text(0, 0, tekst_info_stanu, fontsize=10, wrap=True)
            plt.axis('off')
            pdf.savefig(fig, bbox_inches='tight')
            plt.close()

            messagebox.showinfo("Sukces", "Dane i wykresy zostały zapisane
do PDF.")
```

Powyższy kod przedstawia funkcję dodaj_przyciski_wykresow, która dodaje serię przycisków do interfejsu użytkownika aplikacji wykorzystującej Tkinter, oraz funkcję zapisz_do_pdf, która umożliwia zapisanie wyników i wykresów do pliku PDF. Obie funkcje integrują narzędzia wizualizacji danych i zarządzania plikami z główną logiką aplikacji do programowania liniowego. Funkcja dodaj_przyciski_wykresow tworzy ramkę przycisków umieszczoną na dole głównego okna aplikacji. Zawiera ona przyciski do rozwiązania problemu optymalizacyjnego, generowania różnych rodzajów wykresów (kołowego, liniowego, słupkowego oraz nierówności), a także przycisk do zapisu danych do pliku PDF. Każdy przycisk jest skonfigurowany z odpowiednią komendą, która wywołuje określoną funkcję przy jego naciśnięciu. Funkcja zapisz_do_pdf pozwala na zapisanie danych do pliku PDF,

korzystając z okna dialogowego do wyboru ścieżki i nazwy pliku. Używa biblioteki PdfPages z matplotlib do zarządzania stronami PDF. W funkcji tej, zależnie od dostępności danych (wartości zmiennych i ograniczeń), generowane są wykresy za pomocą odpowiednich funkcji i zapisywane do PDF. Dodatkowo, zawartość tekstowa okna stanu aplikacji jest również zapisywana jako strona w PDF, co pozwala na kompleksowe dokumentowanie procesu i wyników analizy. Wykorzystanie takiej funkcjonalności umożliwia użytkownikowi nie tylko interaktywne zarządzanie i analizę problemów optymalizacyjnych, ale także efektywne dokumentowanie i dzielenie się wynikami pracy, co jest szczególnie wartościowe w kontekście naukowym, edukacyjnym czy profesjonalnym.

```
def generuj_wykres_kolowy(zapisz_pdf=None):
    etykiety = [v.name for v in zmienne.values()]
    wielkosci = [v.varValue for v in zmienne.values()]
    fig, ax = plt.subplots()
    ax.pie(wielkosci, labels=etykiety, autopct='%1.1f%%')
    ax.axis('equal')
    if zapisz_pdf:
        zapisz_pdf.savefig(fig)
    else:
        pokaz_wykres(fig)

def generuj_wykres_liniowy(zapisz_pdf=None):
    etykiety = [v.name for v in zmienne.values()]
    wartosci = [v.varValue for v in zmienne.values()]
    fig, ax = plt.subplots()
    ax.plot(etykiety, wartosci, marker='o')
    ax.set_xlabel('Zmienne')
    ax.set_ylabel('Wartości')
    ax.set_title('Wykres wartości zmiennych')
    if zapisz_pdf:
        zapisz_pdf.savefig(fig)
    else:
        pokaz_wykres(fig)

def parse_constraint(constraint_str):
    if '<=' in constraint_str:
        lhs, rhs = constraint_str.split('<=')
        nierownosc = '<='
    elif '>=' in constraint_str:
        lhs, rhs = constraint_str.split('>=')
        nierownosc = '>='
    else:
        raise ValueError("Nieprawidłowy format ograniczenia. Musi zawierać
'<=' lub '>='.")

    warunki_lhs = lhs.strip().replace(' ', '').split('+')
```

```
    wartosc_rhs = float(rhs.strip())
    return warunki_lhs, wartosc_rhs, nierownosc

def stworz_funkcje_ograniczenia(warunki_lhs, wartosc_rhs):
    wspolczynniki = {'P1': 0, 'P2': 0}
    for warunek in warunki_lhs:
        if 'P1' in warunek:
            wsp = warunek.split('*P1')[0]
            wspolczynniki['P1'] = float(wsp) if wsp else 1.0
        elif 'P2' in warunek:
            wsp = warunek.split('*P2')[0]
            wspolczynniki['P2'] = float(wsp) if wsp else 1.0
    return lambda p1, p2: wspolczynniki['P1'] * p1 + wspolczynniki['P2'] *
p2 - wartosc_rhs

def znajdz_punkt_przeciecia(funkcje, lista_ograniczen):
    for i in range(len(funkcje)):
        for j in range(i + 1, len(funkcje)):
            p1, p2 = sp.symbols('P1 P2')
            eq1 = sp.Eq(funkcje[i](p1, p2), 0)
            eq2 = sp.Eq(funkcje[j](p1, p2), 0)
            solution = sp.solve((eq1, eq2), (p1, p2))
            if solution:
                if all([funkcje[k](solution[p1], solution[p2]) <= 0 if '<='
in lista_ograniczen[k] else funkcje[k](solution[p1], solution[p2]) >= 0 for
k in range(len(funkcje)) if k != i and k != j]):
                    return solution[p1], solution[p2]
    return None, None

def generuj_wykres_nierownosci(lista_ograniczen, pokaz_okno=True,
zapisz_pdf=None):
    zakres_p1 = np.linspace(0, 300, 400)
    zakres_p2 = np.linspace(0, 300, 400)
    P1, P2 = np.meshgrid(zakres_p1, zakres_p2)

    fig, ax = plt.subplots(figsize=(10, 8))
    legendy = []

    kolory_linii = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w', 'tab:blue',
'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink',
'tab:gray', 'tab:olive', 'tab:cyan', 'lime', 'teal']
    kolory_wypelnienia = ['salmon', 'lightgreen', 'lightblue', 'lavender',
'beige', 'wheat', 'tan', 'orchid', 'lightcoral', 'palegreen',
'paleturquoise', 'lightsteelblue', 'powderblue', 'thistle', 'lightgrey',
'rosybrown', 'mediumaquamarine', 'peachpuff', 'khaki', 'bisque']

    funkcje_ograniczen = []
    for i, c in enumerate(lista_ograniczen):
        warunki_lhs, wartosc_rhs, nierownosc = parse_constraint(c)
        funkcja = stworz_funkcje_ograniczenia(warunki_lhs, wartosc_rhs)
```

```
        funkcje_ograniczen.append(funkcja)

        ograniczenie = funkcja(P1, P2)
        if nierownosc == '<=':
            ax.contour(P1, P2, ograniczenie, levels=[0],
colors=kolory_linii[i], linestyles='-')
            ax.contourf(P1, P2, ograniczenie, levels=[ograniczenie.min(),
0], colors=[kolory_wypelnienia[i]], alpha=0.3)
        elif nierownosc == '>=':
            ax.contour(P1, P2, ograniczenie, levels=[0],
colors=kolory_linii[i], linestyles='-')
            ax.contourf(P1, P2, ograniczenie, levels=[0,
ograniczenie.max()], colors=[kolory_wypelnienia[i]], alpha=0.3)

        legendy.append(Line2D([0], [0], color=kolory_linii[i], lw=2,
label=f"Ograniczenie {i+1}: {c}"))

    # Znajdź punkt przecięcia ograniczeń
    p1_przeciecie, p2_przeciecie =
znajdz_punkt_przeciecia(funkcje_ograniczen, lista_ograniczen)
    if p1_przeciecie is not None and p2_przeciecie is not None:
        ax.plot(p1_przeciecie, p2_przeciecie, 'yo', markersize=10)  # Rysuj
żółtą kropkę
        # Dodanie informacji o punkcie przecięcia do legendy
        legendy.append(Line2D([0], [0], marker='o', color='yellow',
label=f'Punkt przecięcia: ({p1_przeciecie:.2f}, {p2_przeciecie:.2f})',
markersize=10))

    ax.legend(handles=legendy, loc='upper right')
    ax.set_xlim((0, 20))
    ax.set_ylim((0, 20))
    ax.set_xlabel('P1')
    ax.set_ylabel('P2')
    ax.set_title('Diagram nierówności')

    if zapisz_pdf:
        zapisz_pdf.savefig(fig)
    if pokaz_okno:
        pokaz_wykres(fig)
    plt.close(fig)


def pokaz_wykres(fig):
    okno_wykresu = tk.Toplevel(root)
    okno_wykresu.title("Wykres")
    plotno = FigureCanvasTkAgg(fig, master=okno_wykresu)
    plotno.draw()
    plotno.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)
```

Powyższy kod przedstawia zaawansowane funkcje do generowania i prezentacji wykresów w aplikacji wykorzystującej Pythona z bibliotekami matplotlib, SymPy i Tkinter. Każda funkcja jest przystosowana do specyficznego typu wizualizacji danych z problemów optymalizacyjnych. Funkcja generuj_wykres_kolowy przyjmuje opcjonalny argument zapisz_pdf. Gdy argument ten nie jest podany, funkcja tworzy wykres kołowy, który przedstawia wartości zmiennych jako procenty całkowitej wartości, gdzie etykiety i odpowiednie segmenty wykresu odpowiadają poszczególnym zmiennym. Gdy jest dostarczony plik PDF, wykres jest zapisywany bezpośrednio do niego. Podobnie, generuj_wykres_liniowy rysuje wykres liniowy, który łączy punkty odpowiadające wartościom zmiennych, z etykietami dla zmiennych na osi X i ich wartościami na osi Y. Ta funkcja również obsługuje opcję zapisu wykresu do pliku PDF. Funkcja parse_constraint służy do analizowania stringów reprezentujących ograniczenia matematyczne, oddzielając lewą i prawą stronę nierówności oraz identyfikując typ nierówności (większe lub równe, mniejsze lub równe). Z kolei stworz_funkcje_ograniczenia tworzy funkcję matematyczną reprezentującą ograniczenie na podstawie przetworzonych wcześniej warunków i współczynników. znajdz_punkt_przeciecia wykorzystuje funkcje ograniczeń do znalezienia punktu, w którym linie reprezentujące te ograniczenia się przecinają, co jest kluczowym elementem przy rozwiązywaniu problemów programowania liniowego. W tym celu korzysta z biblioteki SymPy do symbolicznego rozwiązywania równań. generuj_wykres_nierownosci jest funkcją, która wizualizuje ograniczenia na dwuwymiarowej siatce. Linie ograniczeń są rysowane i wypełniane różnymi kolorami w zależności od ich typu, tworząc diagram, który pokazuje dopuszczalny obszar rozwiązań. Dodatkowo, funkcja ta może oznaczyć punkt przecięcia ograniczeń i, jeśli jest podane, zapisuje wykres do pliku PDF. Na koniec, pokaz_wykres jest funkcją, która umożliwia wyświetlenie wygenerowanego wykresu w nowym oknie GUI aplikacji, korzystając z widgetu FigureCanvasTkAgg do integracji grafiki z matplotlib z interfejsem użytkownika Tkinter. Te funkcje łącznie oferują bogaty zestaw narzędzi do analizy i prezentacji danych optymalizacyjnych, co czyni je nieocenionymi w narzędziach badawczych, edukacyjnych, czy profesjonalnych analizach inżynierskich i naukowych.

```
def wyczysc_ramke(ramka):
    for widget in ramka.winfo_children():
        widget.destroy()

def wyczysc_wszystko():
    global zmienne, ograniczenia, problem
    zmienne.clear()
    ograniczenia.clear()
    problem = None
```

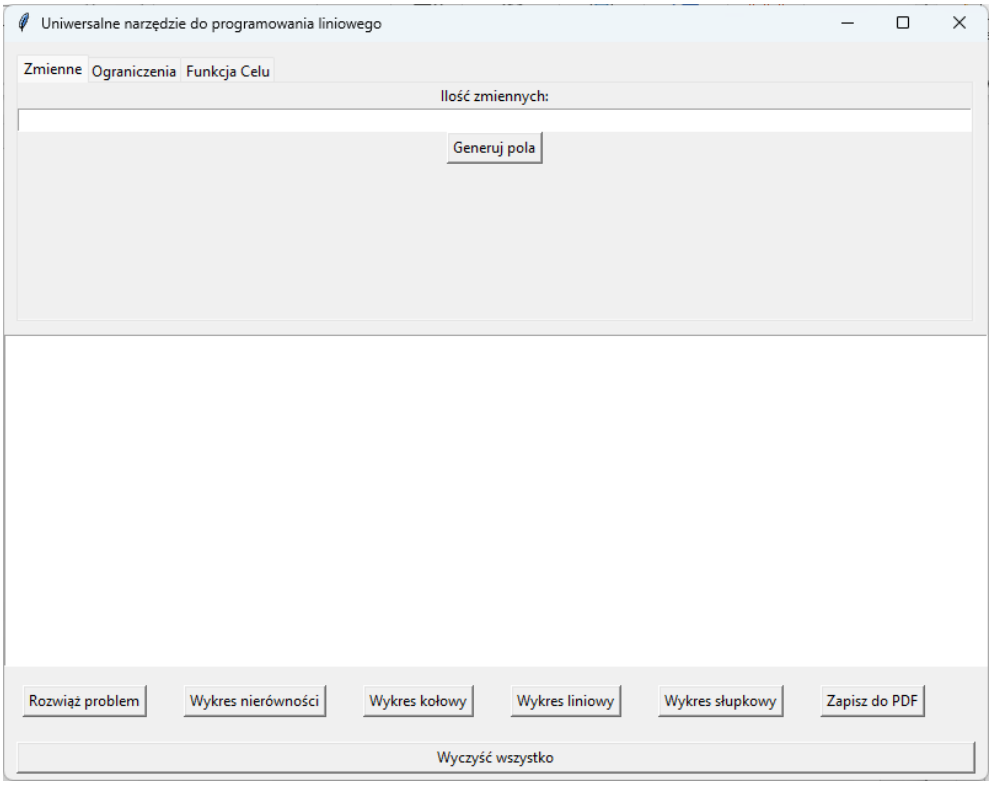
```
    wyczysc_ramke(zakladka_zmienne)
    wyczysc_ramke(zakladka_ograniczenia)
    inicjalizuj_zakladke_zmienne()
    inicjalizuj_zakladke_ograniczenia()
    pole_funkcji_celu.delete(0, tk.END)
    info_stanu.config(state='normal')
    info_stanu.delete('1.0', tk.END)
    info_stanu.config(state='disabled')
    messagebox.showinfo("Informacja", "Dane zostały wyczyszczone.")

def dodaj_przycisk_czyszczenia():
    przycisk_czyszczenia = tk.Button(root, text="Wyczyść wszystko",
command=wyczysc_wszystko)
    przycisk_czyszczenia.pack(side=tk.BOTTOM, fill=tk.X, padx=10, pady=5)

dodaj_przycisk_czyszczenia()

inicjalizuj_problem()
inicjalizuj_zakladki()
dodaj_przyciski_wykresow()

info_stanu = tk.Text(root, height=10, state='disabled')
info_stanu.pack(fill=tk.BOTH, expand=True)

root.mainloop()
```

Powyższy fragment kodu przedstawia zaawansowaną implementację aplikacji GUI w Pythonie z wykorzystaniem biblioteki Tkinter, służącej do zarządzania problemami optymalizacyjnymi przy użyciu programowania liniowego. Kody te inicjalizują interfejs użytkownika, dodają funkcjonalności i zarządzają interakcjami z użytkownikiem. Na początku znajduje się funkcja wyczysc_ramke, która służy do usuwania wszystkich widżetów z określonej ramki Tkinter. Ta funkcja jest wykorzystywana w innych częściach kodu do czyszczenia danych z interfejsu użytkownika, co pozwala na resetowanie stanu aplikacji bez konieczności jej restartowania. Funkcja wyczysc_wszystko jest kluczowym elementem zarządzania stanem aplikacji. Resetuje wszystkie globalne zmienne przechowujące dane o zmiennych, ograniczeniach i samym problemie optymalizacyjnym. Opróżnia również zawartość odpowiednich ramek i pól tekstowych, zapewniając, że użytkownik zacznie pracę od czystego stanu po jej wykonaniu. Na koniec funkcja ta wyświetla komunikat informacyjny, potwierdzając czyszczenie danych. Kolejna część kodu dotyczy dodania przycisku „Wyczyść wszystko” do głównego okna aplikacji. Przycisk ten, gdy zostanie naciśnięty, wywołuje funkcję wyczysc_wszystko, co pozwala użytkownikowi łatwo i szybko zresetować wszystkie

wprowadzone informacje i przygotować interfejs do nowych zadań. Dodatkowo, kod inicjuje problem optymalizacyjny i dodaje zakładki za pomocą funkcji inicjalizuj_problem i inicjalizuj_zakladki. Zakładki te mogą zawierać kontrolki dla zmiennych, ograniczeń oraz funkcji celu, umożliwiając użytkownikowi strukturalne wprowadzanie danych do problemu optymalizacyjnego. Na końcu znajduje się inicjalizacja pól tekstowych informacyjnych, które są używane do wyświetlania bieżących informacji o stanie operacji lub wyników. Te pola są domyślnie ustawione na stan 'disabled' (nieedytowalne), co zapobiega ich modyfikacji przez użytkownika, ale pozwalają na programowe aktualizowanie ich treści. Całość jest zamknięta w pętli mainloop Tkintera, która uruchamia i utrzymuje główne okno aplikacji do momentu jego zamknięcia przez użytkownika.

6. Interfejs użytkownika

Interfejs użytkownika przedstawiony na zrzucie ekranu poniżej to okno aplikacji o nazwie "Uniwersalne narzędzie do programowania liniowego". Jest to aplikacja graficzna z zakładkami i przyciskami, umożliwiająca użytkownikowi interakcję z narzędziem programowania liniowego.

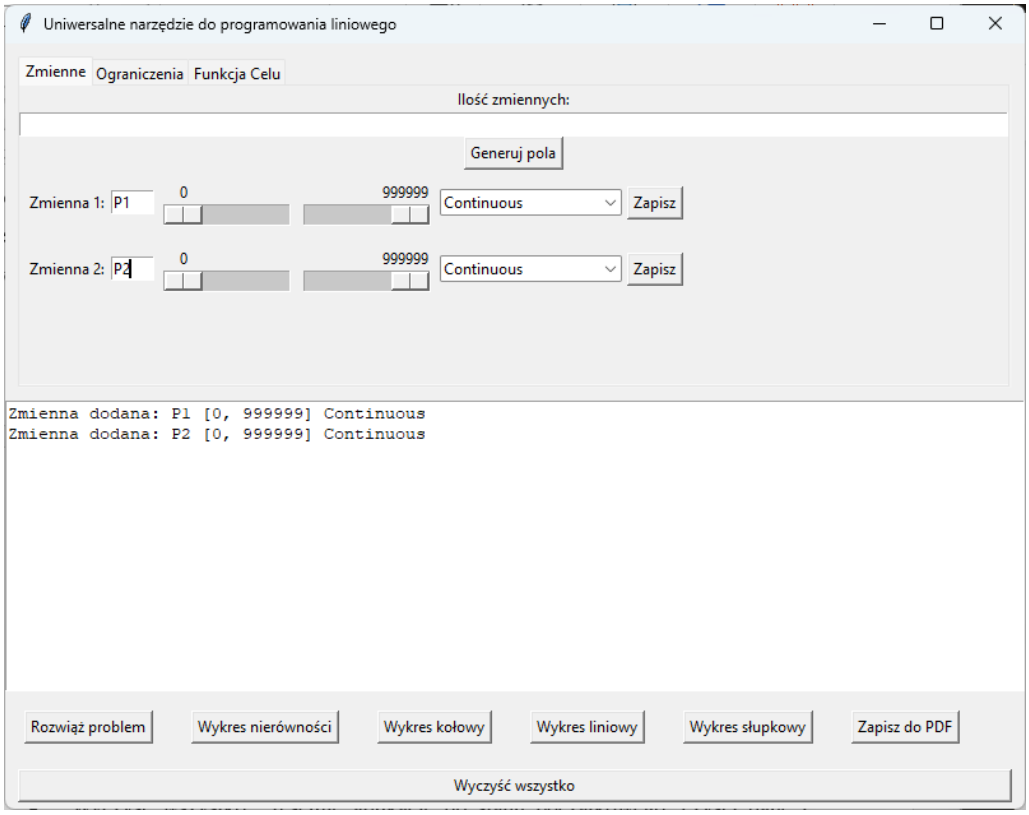


W górnej części okna znajduje się szereg zakładek o etykietach "Zmienne", "Ograniczenia" oraz "Funkcja Celu", sugerujące, że użytkownik może przełączać się między różnymi

aspektami definiowania problemu programowania liniowego. Poniżej zakładek, w zakładce "Zmienne", wyświetlany jest interfejs z polem tekstowym i przyciskiem "Generuj pola". Pole tekstowe służy do wprowadzenia liczby zmiennych, które mają być użyte w problemie optymalizacyjnym, a przycisk "Generuj pola" będzie generował odpowiednią liczbę pól do wprowadzenia danych dla tych zmiennych. W dolnej części interfejsu znajduje się rząd przycisków związanych z operacjami, które użytkownik może przeprowadzić:

- "Rozwiąż problem": uruchamia proces rozwiązywania zdefiniowanego problemu programowania liniowego.
- "Wykres nierówności", "Wykres kołowy", "Wykres liniowy", "Wykres słupkowy": służą do generowania różnych typów wykresów na podstawie danych wejściowych i wyników problemu optymalizacyjnego.
- "Zapisz do PDF": umożliwia eksport wyników i wykresów do pliku PDF.
- "Wyczyść wszystko": resetuje aplikację do stanu początkowego, czyści dane i przygotowuje interfejs do nowego zadania.

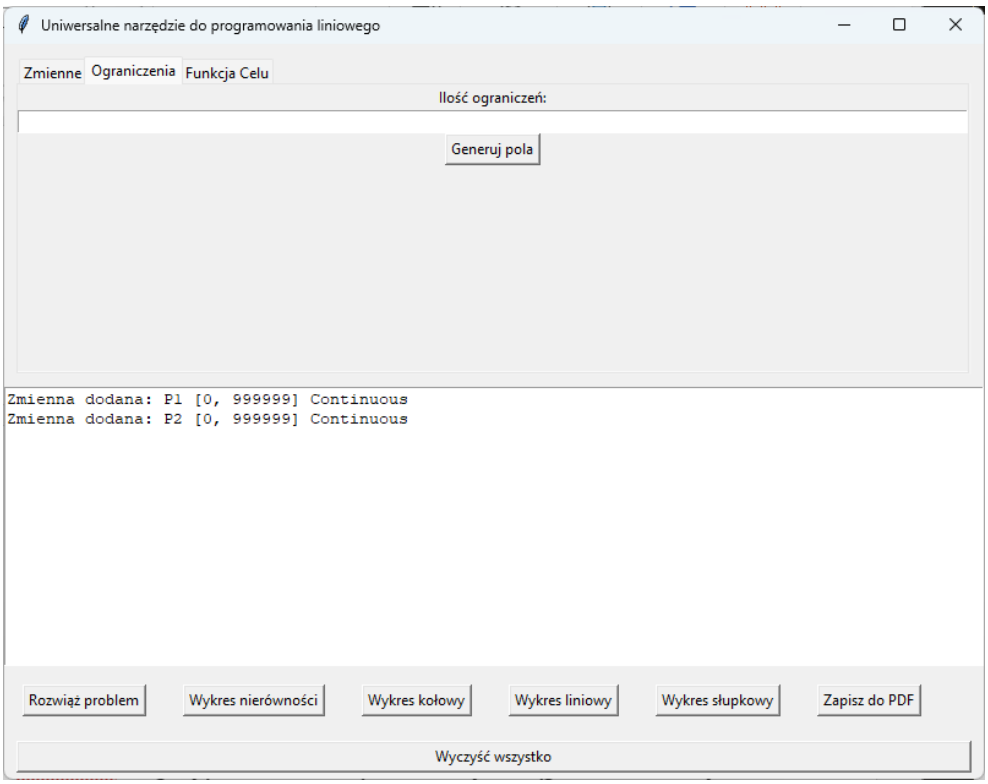
Poniżej przycisków w dolnej części okna znajduje się przestrzeń przeznaczona na dziennik aktywności użytkownika, w którym wyświetlany jest każdy krok wykonany przez użytkownika.



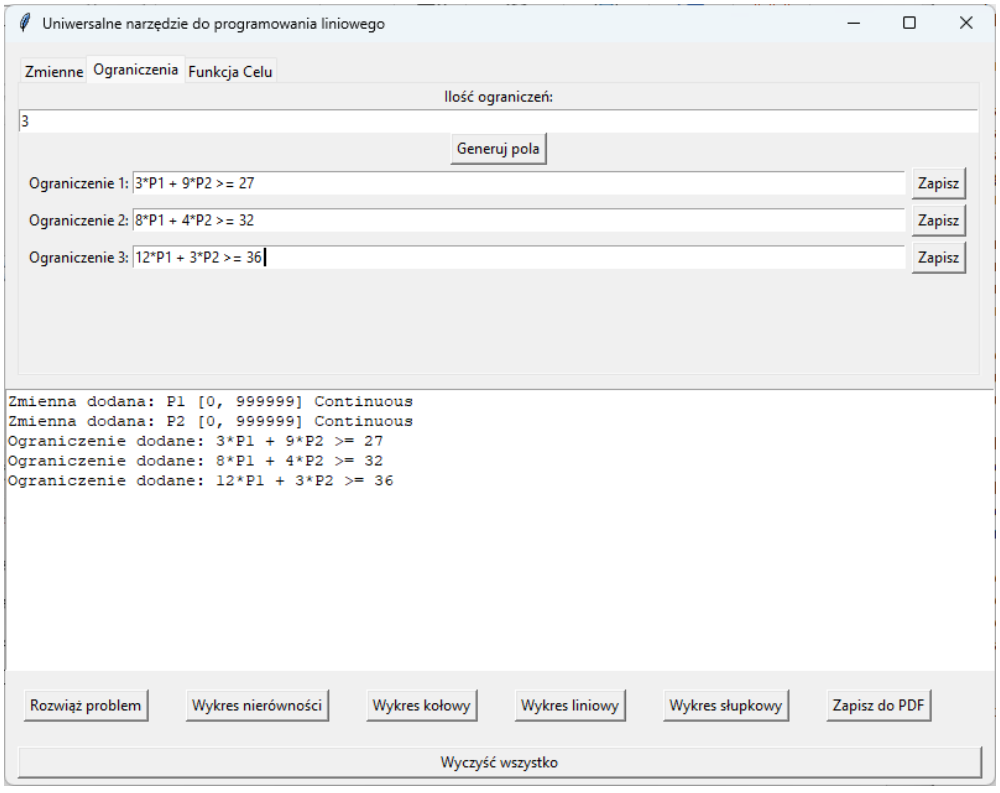
Na obrazku powyżej interfejs użytkownika wykazuje, że użytkownik zdecydował się wygenerować pola dla dwóch zmiennych. W zakładce "Zmienne" są teraz dwie sekcje, każda odpowiadająca za jedną zmienną. W każdej sekcji można zauważyć następujące elementy:

- Etykieta zmiennej - Pokazuje "Zmienna 1: P1" i "Zmienna 2: P2", co wskazuje na to, że zmienne zostały nazwane przez użytkownika oraz, że są automatycznie przypisywane w kolejności tworzenia.
- Suwaki ograniczeń wartości - Dla każdej zmiennej widoczne są dwa suwaki, które pozwolą użytkownikowi określić dolne i górne ograniczenie wartości zmiennej. Suwaki te są ustawione z zakresem od 0 do 999999, co wskazuje na możliwość definiowania szerokiego spektrum ograniczeń.
- Rozwijane menu typu zmiennej - Obok każdej etykiety zmiennej znajduje się rozwijane menu z opcjami "Continuous" oraz „Integer”, w którym obecnie wybrano "Continuous", sugerujące, że zmienne są traktowane jako ciągłe. Menu to oferuje również opcję wyboru innych typów zmiennych, jak np. całkowitoliczbowe.
- Przycisk zapisu - Każda sekcja posiada przycisk "Zapisz", który prawdopodobnie służy do zatwierdzania wprowadzonych danych dotyczących każdej zmiennej.

Poniżej sekcji dla zmiennych znajduje się obszar tekstowy, w którym wyświetlane są komunikaty informacyjne. Na przykład, pokazuje on informacje o dodanych zmiennych, co sugeruje, że akcje użytkownika były skuteczne: "Zmienna dodana: P1 [0, 999999] Continuous" i "Zmienna dodana: P2 [0, 999999] Continuous".



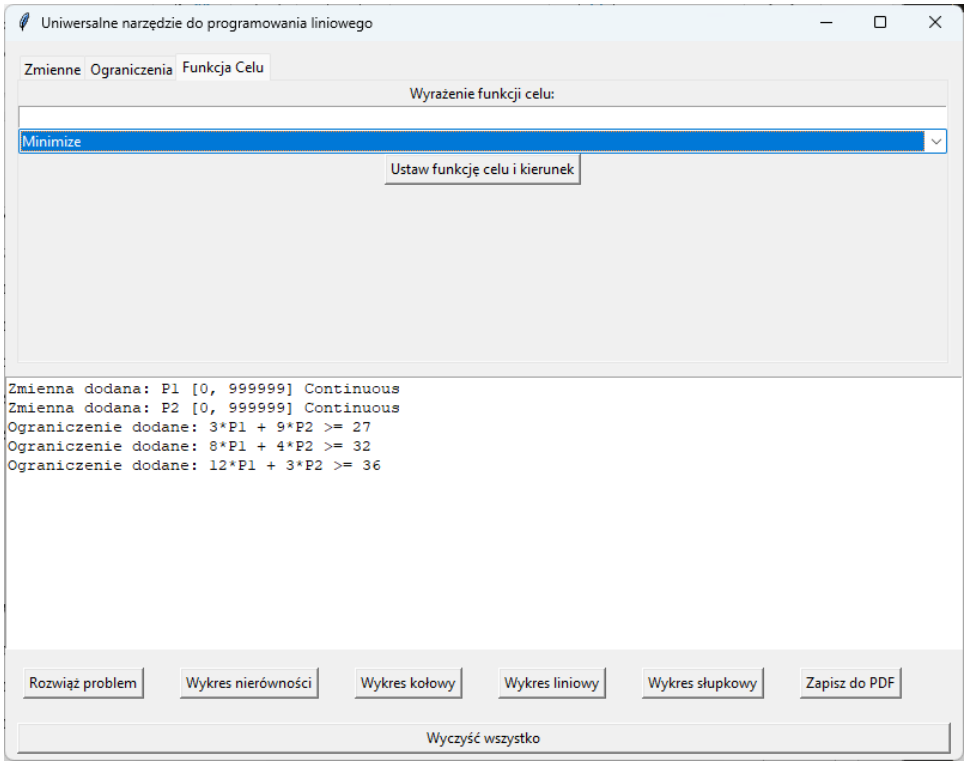
Powyższy zrzut ekranu przedstawia interfejs zakładki ograniczenia opracowanej aplikacji. W tej zakładce pojawia się pole tekstowe przeznaczone do wprowadzenia liczby ograniczeń, które użytkownik chce zdefiniować w modelu. Obok pola tekstowego umieszczony jest przycisk "Generuj pola", który służy do stworzenia odpowiedniej liczby pól wejściowych dla ograniczeń w zależności od wprowadzonej liczby.



Na kolejnym zrzucie ekranu w zakładce "Ograniczenia" aplikacji do programowania liniowego dodano trzy ograniczenia. Pole tekstowe na górze, gdzie użytkownik wpisuje liczbę ograniczeń, zawiera teraz liczbę "3", co wskazuje na to, że użytkownik chce zdefiniować trzy ograniczenia. Poniżej tego znajdują się trzy pola z wyrażeniami matematycznymi, które definiują ograniczenia liniowe dla zmiennych P1 i P2. Każde z ograniczeń ma przypisany przycisk "Zapisz", co sugeruje możliwość zapisu i zastosowania każdego z ograniczeń do modelu. Wprowadzono trzy ograniczenia:

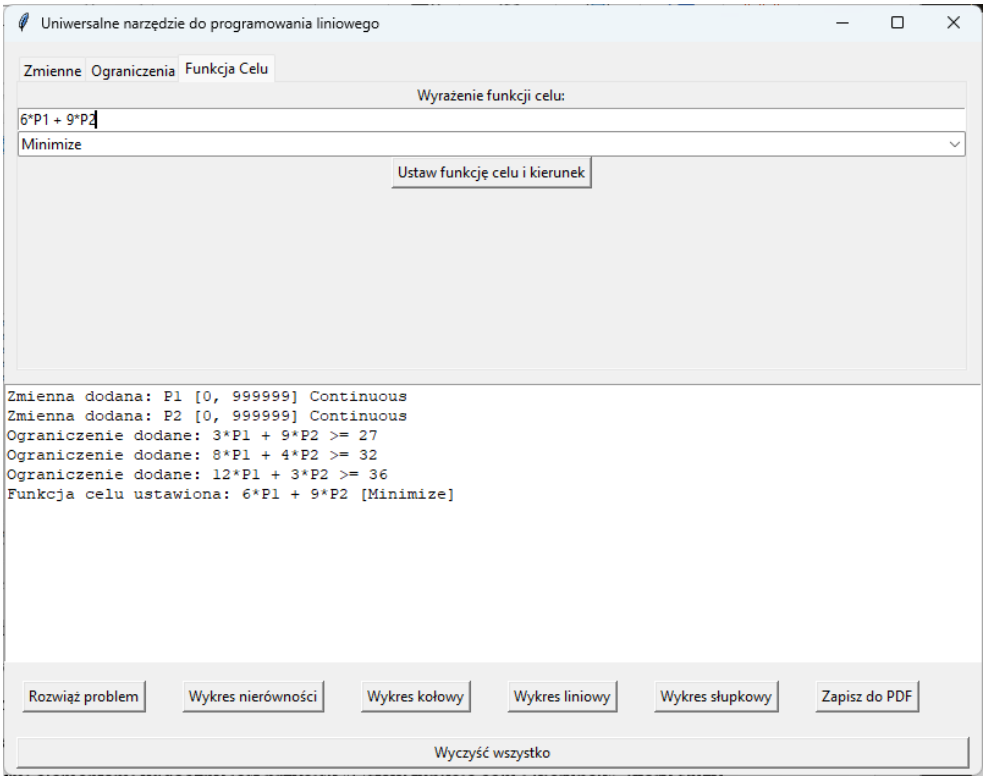
- $3 \cdot P1 + 9 \cdot P2 \geq 27$
- $8 \cdot P1 + 4 \cdot P2 \geq 32$
- $12 \cdot P1 + 3 \cdot P2 \geq 36$

W dolnej części interfejsu, w obszarze tekstowym z informacjami zwrotnymi, widoczne są komunikaty potwierdzające dodanie zarówno zmiennych, jak i nowo zdefiniowanych ograniczeń. Informacje te wskazują, że zmienne P1 i P2 zostały dodane z określonymi ograniczeniami wartości i typami, a także, że określone ograniczenia zostały dodane do modelu.

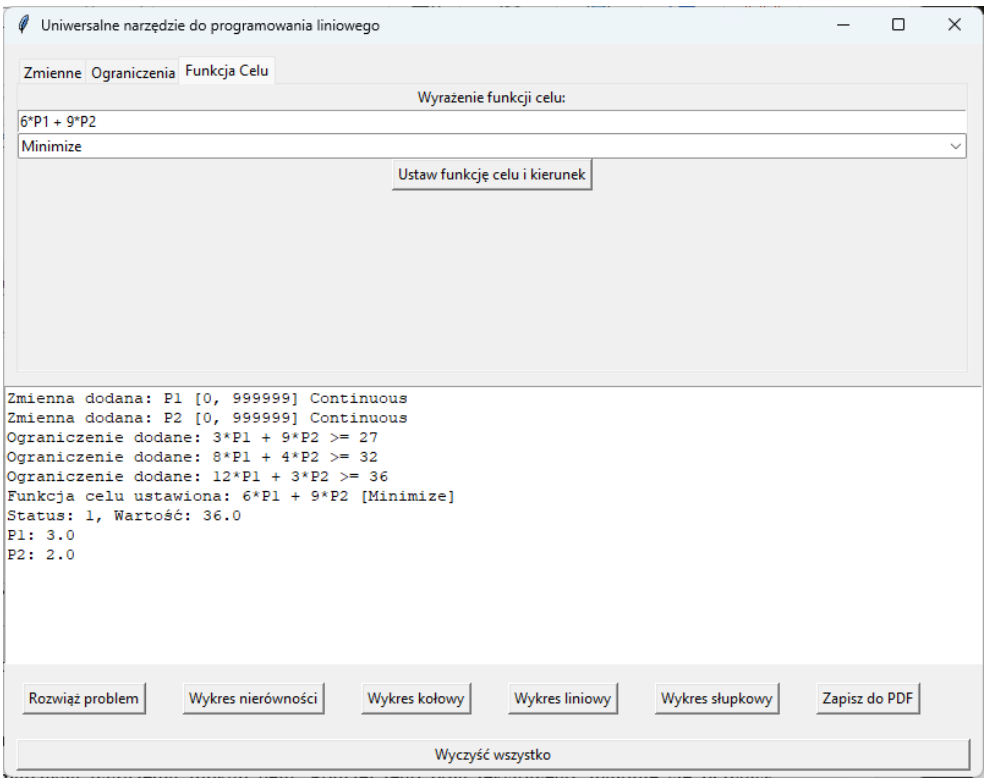


W zakładce "Funkcja Celu" zauważalne są pola do zdefiniowania funkcji celu problemu programowania liniowego. Na górze znajduje się rozwijane menu, w którym użytkownik może wybrać, czy cel ma być minimalizowany ("Minimize") czy maksymalizowany. Nad nim znajduje się duże pole tekstowe, które służy do wpisania wyrażenia matematycznego funkcji

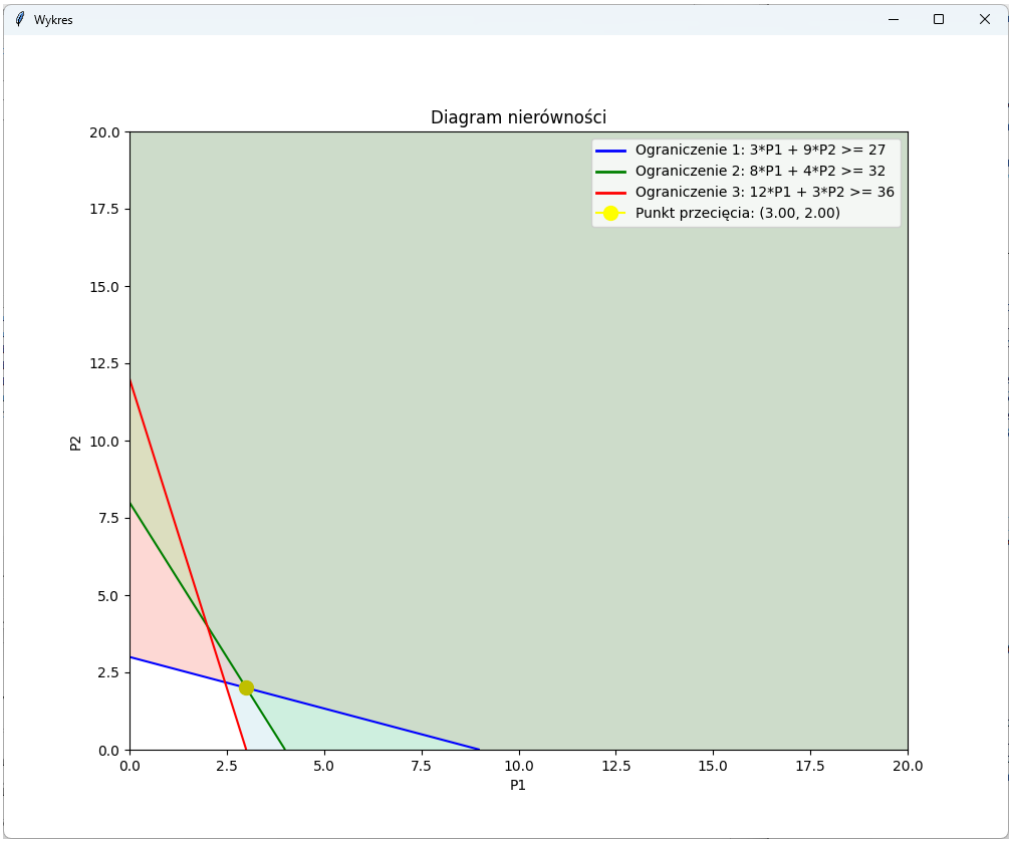
celu. Pod tymi elementami widoczny jest przycisk "Ustaw funkcję celu i kierunek", który służy do zatwierdzenia wybranej funkcji celu oraz określenia kierunku optymalizacji.



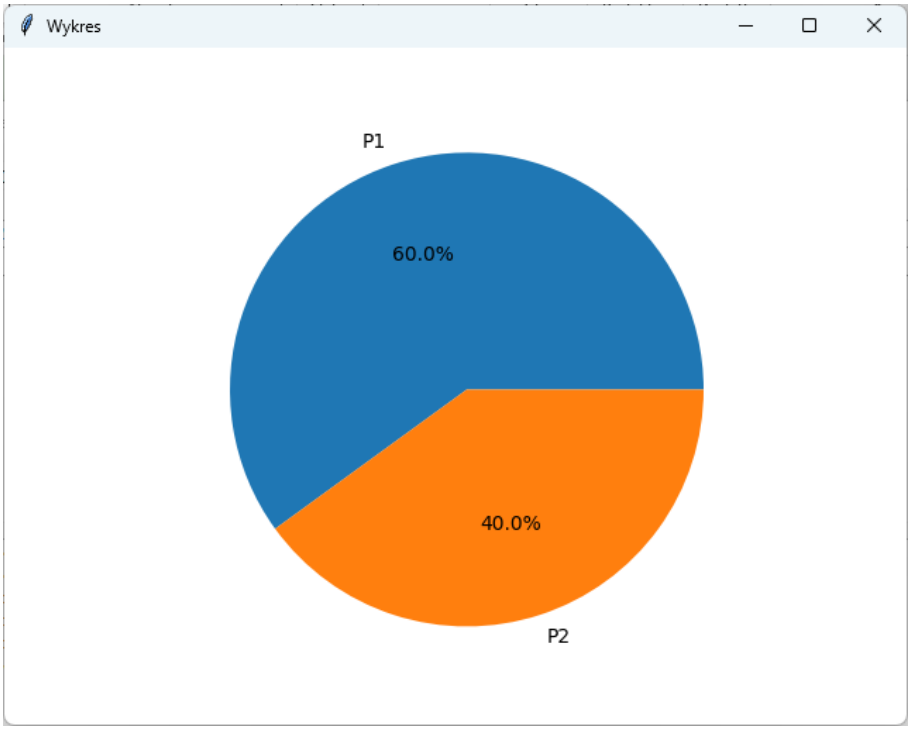
Na powyższym zrzucie ekranu interfejsu aplikacji do programowania liniowego zauważamy, że użytkownik uzupełnił informacje w zakładce "Funkcja Celu". W polu tekstowym na środku ekranu pojawiło się wyrażenie matematyczne funkcji celu „6P1 + 9P2”, które jest formułą, jaką aplikacja ma minimalizować. Obok formuły znajduje się rozwijane menu, w którym użytkownik wybrał opcję „Minimize”, wskazującą na to, że celem jest minimalizacja wyrażenia funkcji celu. Poniżej tego pola tekstowego znajduje się przycisk "Ustaw funkcję celu i kierunek", który służy do zatwierdzenia wprowadzonej funkcji celu i rozpoczęcia procesu rozwiązywania problemu optymalizacyjnego. Dodatkowo, na dole okna aplikacji pojawiła się nowa informacja "Funkcja celu ustawiona: 6P1 + 9P2 [Minimize]", która potwierdza, że funkcja celu została skonfigurowana zgodnie z zamiarami użytkownika. Pokazuje to, że użytkownik zakończył proces definiowania problemu programowania liniowego, obejmującego zmienne, ograniczenia oraz funkcję celu.



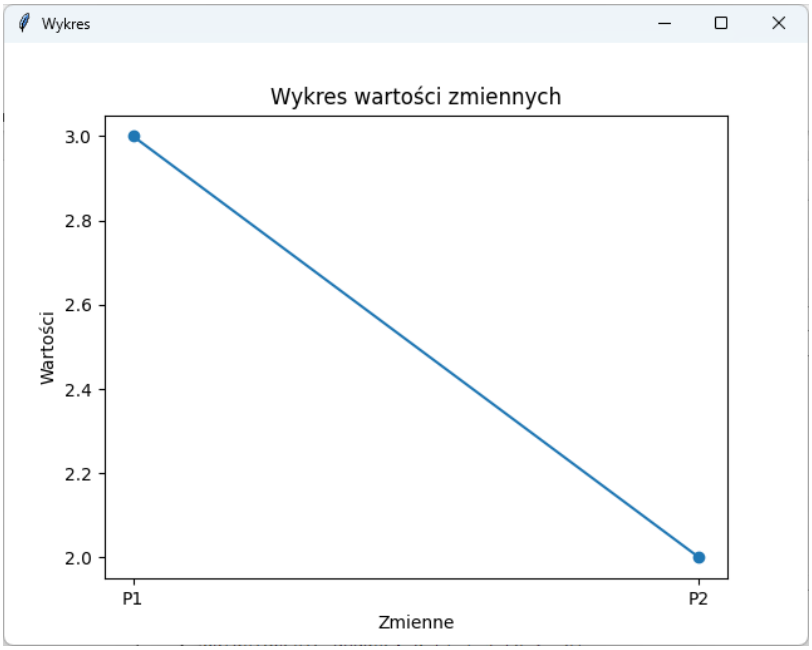
W polu tekstowym pojawiły się informacje o wynikach rozwiązania problemu optymalizacyjnego. Został wyświetlony status rozwiązania, wartość funkcji celu po optymalizacji oraz wartości przypisane do zmiennych P1 i P2. Status oznaczony liczbą "1" wskazuje na to, że problem został rozwiązany poprawnie (w bibliotece PuLP status "1" odpowiada "Optimal"). Wartość funkcji celu to "36.0", co oznacza wynik minimalizacji wyrażenia "6P1 + 9P2". Dalej podane są optymalne wartości zmiennych decyzyjnych, gdzie P1 przyjęło wartość "3.0", a P2 "2.0". Te wyniki wskazują na to, że po wprowadzeniu ograniczeń i zdefiniowaniu funkcji celu, program z sukcesem znalazł rozwiązanie, które minimalizuje funkcję celu w ramach określonych ograniczeń.



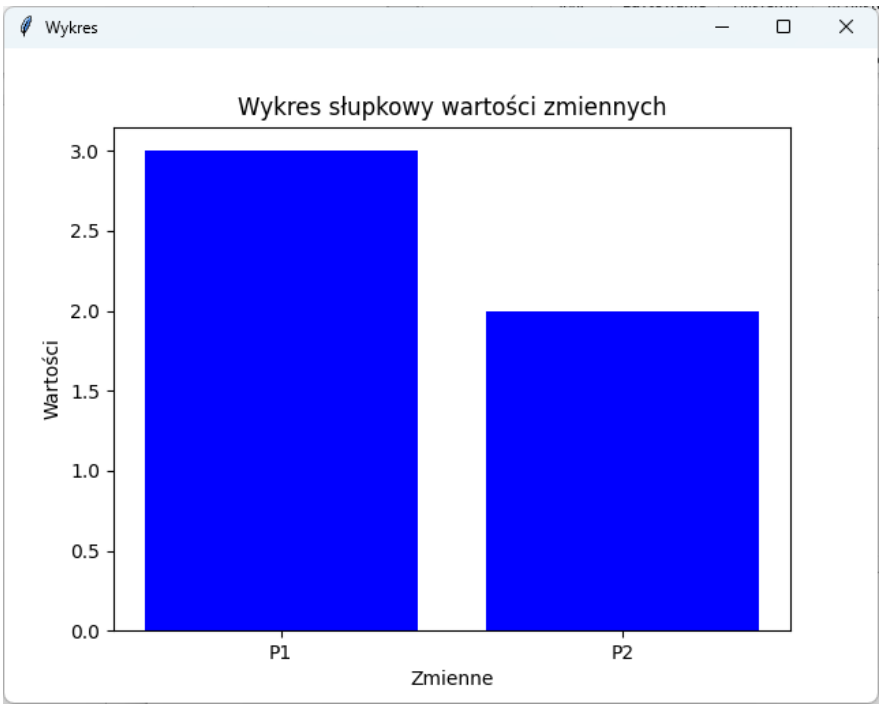
Powyższy rysunek przedstawia wykres nierówności z aplikacji do programowania liniowego, który graficznie pokazuje ograniczenia zdefiniowane przez użytkownika. Widzimy tu wykres, gdzie oś pozioma oznacza wartości zmiennej P1, a oś pionowa zmiennej P2. Różnokolorowe linie reprezentują trzy różne ograniczenia, które użytkownik wprowadził do systemu. Każda linia na wykresie jest etykietowana odpowiadającym jej ograniczeniem matematycznym, dając wizualne przedstawienie obszaru rozwiązań, które spełniają te ograniczenia. Obszar, w którym linie się przecinają oznaczony żółtym punktem, reprezentuje punkt optymalnego rozwiązania dla danych ograniczeń przy minimalizacji funkcji celu. Punkt przecięcia, jak widać, został dokładnie oznaczony jako (3.00, 2.00), co zgadza się z poprzednio podanymi wynikami rozwiązania problemu. Obszary po jednej stronie linii są zakolorowane, co wskazuje na spełnienie nierówności, definiując region, w którym wszystkie ograniczenia są spełnione. Wykres ten jest pomocny w wizualizacji złożoności problemu i pomaga zorientować się w przestrzeni rozwiązań, która spełnia wszystkie nałożone na problem ograniczenia.



Na powyższym rysunku przedstawiony został wykres kołowy, który ilustruje proporcjonalny udział dwóch zmiennych, P1 i P2, w kontekście rozwiązania problemu programowania liniowego. Wykres dzieli koło na dwa segmenty, gdzie każdy segment reprezentuje udział procentowy jednej ze zmiennych. Segment odpowiadający zmiennej P1 jest większy i zajmuje około 60% koła, pokazany w kolorze niebieskim, natomiast segment dla zmiennej P2 to około 40% koła, pokazany w kolorze pomarańczowym. Procentowe wartości dla obu zmiennych są wyraźnie oznaczone na wykresie. Wykres ten może być wykorzystywany do wizualizacji wkładu poszczególnych zmiennych w optymalne rozwiązanie problemu optymalizacyjnego lub do prezentacji, jak duży jest ich udział w osiągnięciu wartości funkcji celu w modelu optymalizacji. Jest to użyteczne narzędzie, które może pomóc w intuicyjnym zrozumieniu relacji między zmiennymi, zwłaszcza w edukacyjnych lub prezentacyjnych kontekstach, gdzie wizualizacja danych jest kluczowa.

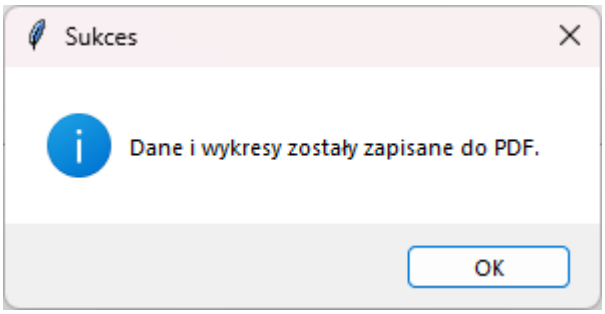


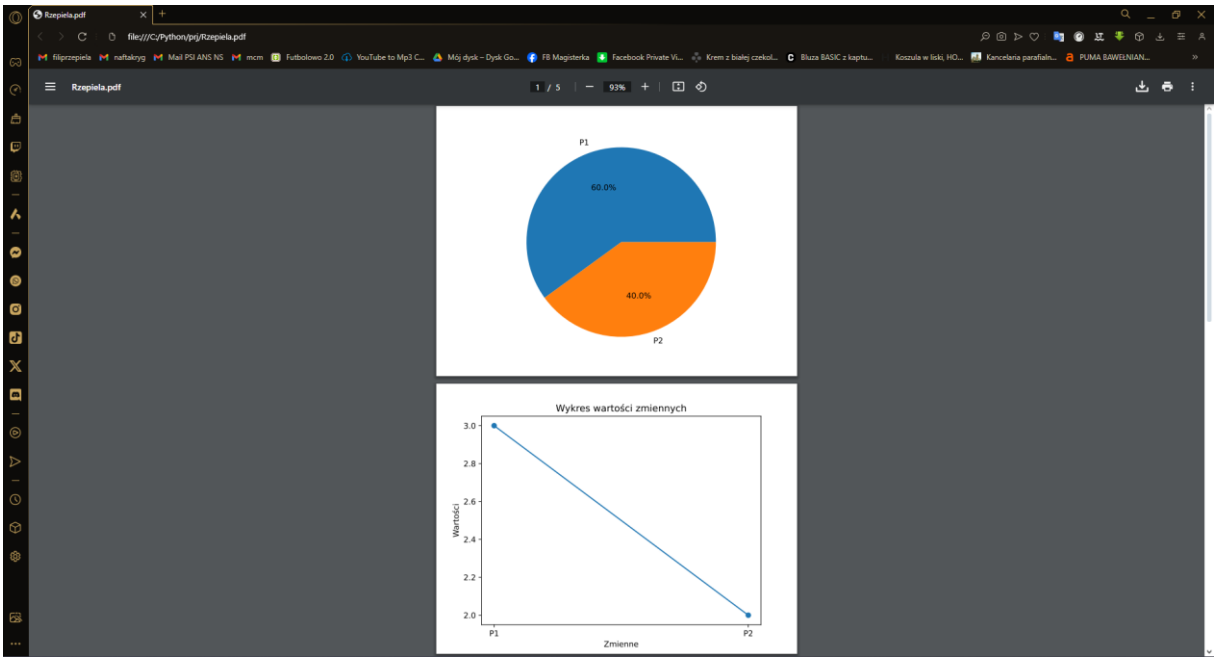
Na załączonym powyżej obrazie widnieje wykres liniowy oznaczony tytułem "Wykres wartości zmiennych". Wykres przedstawia dwie zmienne, P1 i P2, zaznaczone na osi poziomej oznaczonej jako "Zmienne". Na osi pionowej, oznaczonej jako "Wartości", wyraźnie zaznaczone są konkretne wartości dla obu zmiennych. Punkty reprezentujące każdą zmienną są połączone linią, tworząc prosty wykres liniowy. Punkt odpowiadający zmiennej P1 znajduje się wyżej na osi wartości, co sugeruje większą wartość zmiennej w porównaniu do P2. Linia łącząca te dwa punkty opada od P1 do P2, co wizualizuje różnicę wartości między tymi zmiennymi. Wykres zapewnia prostą, ale wyraźną ilustrację wartości tych dwóch zmiennych, co jest przydatne przy szybkiej wizualnej ocenie i porównaniu wartości tych zmiennych.

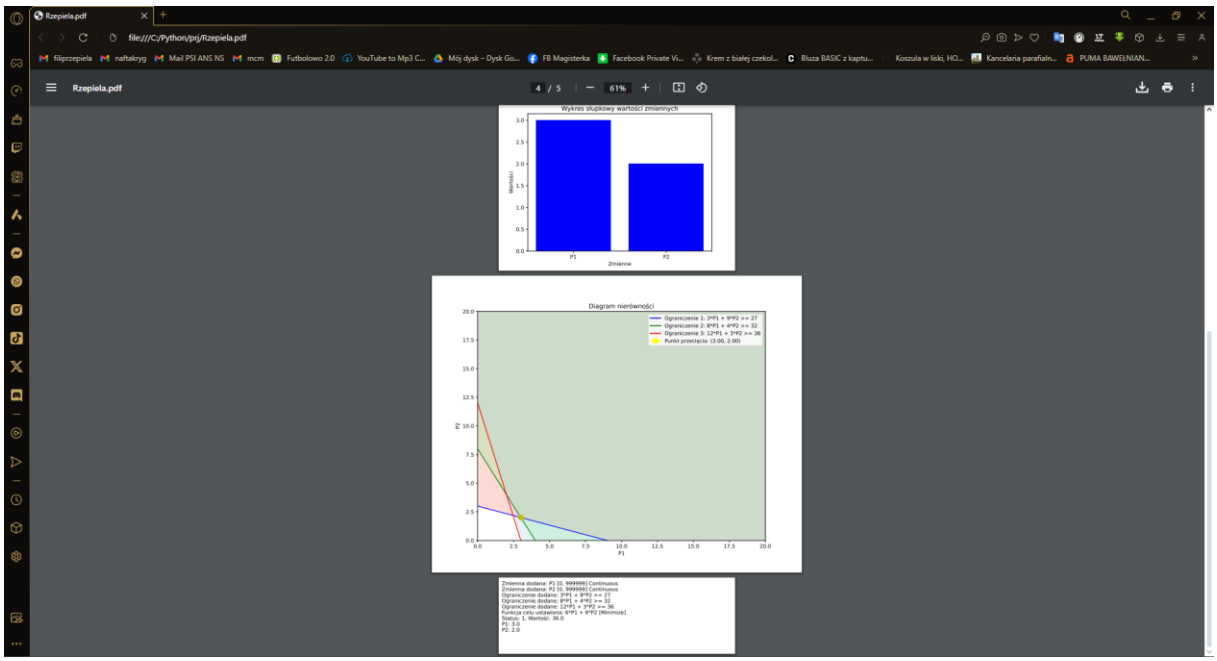


Powyżej przedstawiony obraz pokazuje wykres słupkowy o tytule „Wykres słupkowy wartości zmiennych”. Wykres przedstawia dwa słupki odpowiadające dwóm zmiennym, P1 i P2, zlokalizowanym na osi poziomej etykietowanej jako „Zmienne”. Na osi pionowej, oznaczonej jako „Wartości”, wysokość każdego słupka odpowiada wartości przypisanej do odpowiedniej zmiennej. Słupek dla zmiennej P1 jest wyraźnie wyższy niż słupek dla P2, co ilustruje, że zmienna P1 ma większą wartość. Oba słupki są koloru niebieskiego i wyraźnie oddzielone, co umożliwia łatwe porównanie między wartościami obu zmiennych. Wykres ten jest użytecznym narzędziem do wizualizacji i porównywania wartości, co może być pomocne przy prezentowaniu wyników lub w dalszej analizie problemu optymalizacyjnego.

Wciśnięcie przycisku „Zapisz do PDF” uruchamia procedurę zapisu wykresów oraz danych z dziennika aktywności użytkownika do pliku PDF. Po zakończeniu procedury uzyskujemy informację o ukończeniu zapisu. Podgląd pliku PDF znajduje się na obrazkach poniżej (pod oknem sukcesu).

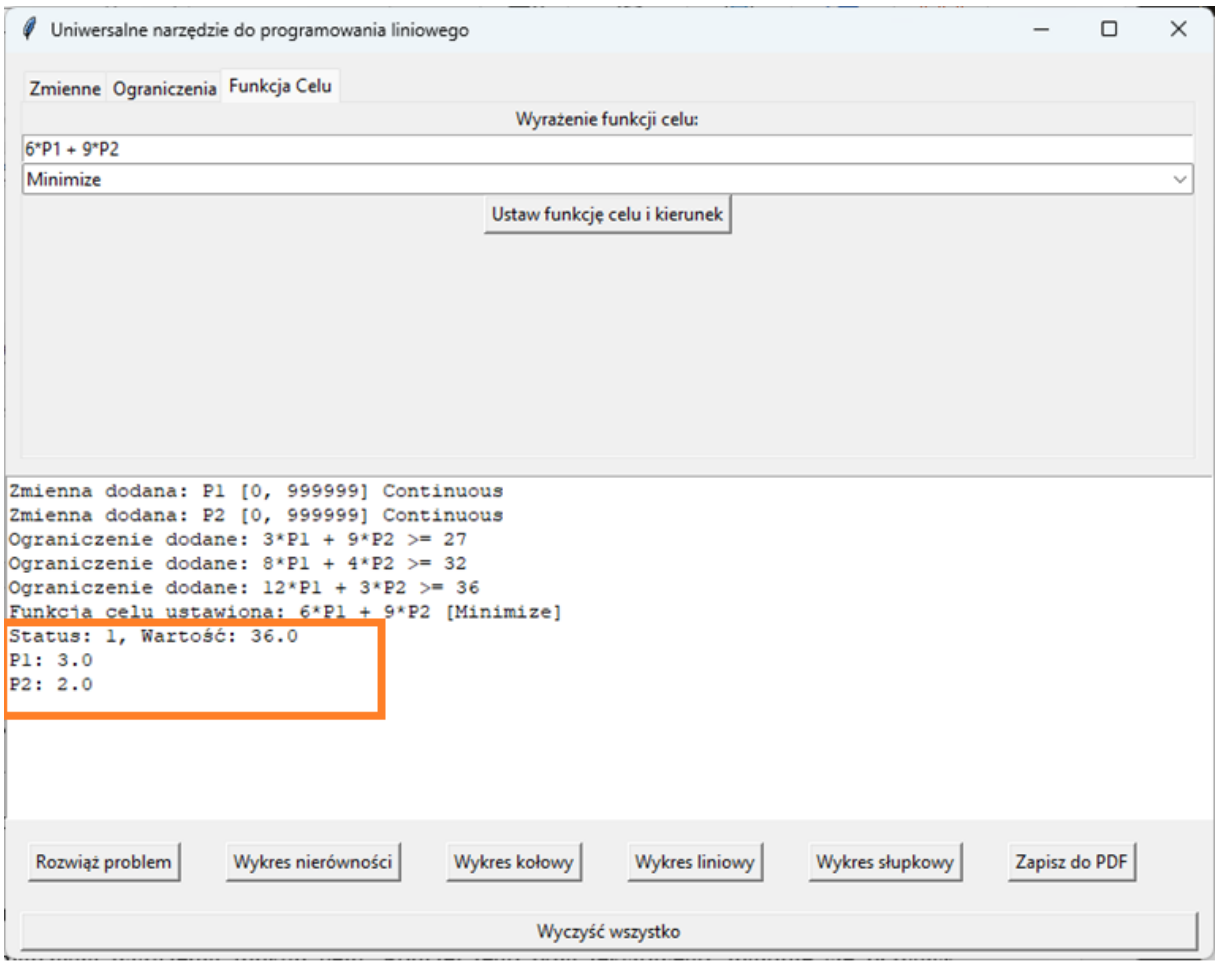






7. Wyniki

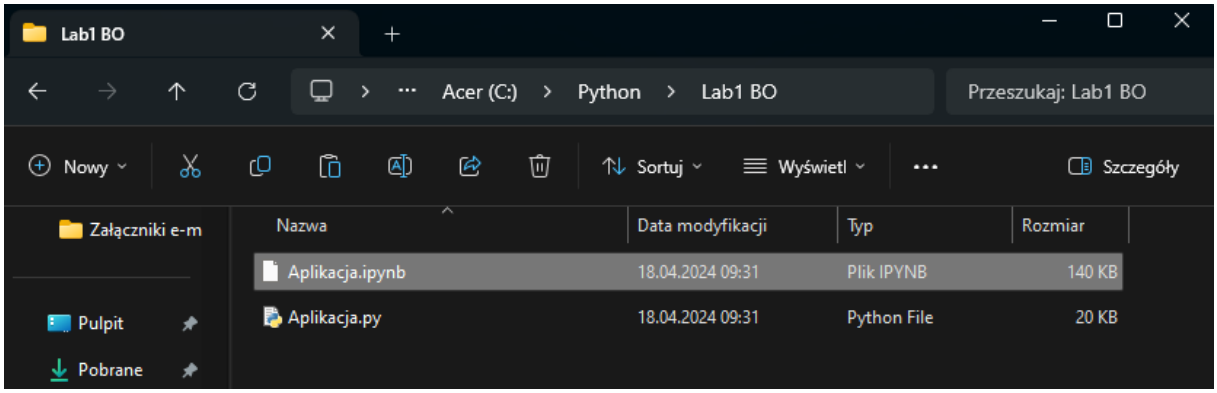
Podczas zajęć laboratoryjnych wykonano zadania zarówno z pomocą oprogramowania Microsoft Excel z dodatkiem Solver, jak i opracowano aplikację w języku Python, której nadrzędnym celem było wykonanie tego samego procesu w prostszy i przyjazny dla użytkownika sposób. Potwierdzeniem poprawności działania opracowanej aplikacji jest fakt, że wyniki uzyskane zarówno w arkuszu kalkulacyjnym Microsoft Excel, jak i w opracowanej aplikacji są takie same. Dodatkowym atutem opracowanej aplikacji jest fakt, że posiada ona reprezentację graficzną problemu poprzez generowane wykresy. Wygenerowany plik PDF może posłużyć jako sprawozdanie laboratoryjne, czy też raport przeprowadzonego badania. Podobieństwo wyników stwierdzające poprawność funkcjonowania aplikacji zostały przedstawione poniżej. Jak można zauważyć wyniki faktycznie są takie same.



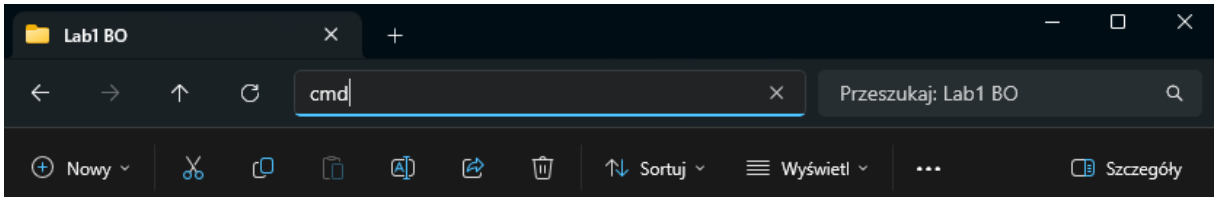
	A	B	C	D	E	F	G	H	I
1	Zmienne decyzyjne		P1	P2					
2	Rozwiązanie		3,00	2,00					
3									
4	FC-Funkcja celu		min	36					
5									
6	Cel współczynniki		C1	C2					
7			6,00	9,00					
8									
9	Ograniczenie								
10									
11	Pierwsze		3	9		27,00		27,00	
12	Drugie		8	4		32,00		32,00	
13	Trzecie		12	3		42,00		36,00	
14									

8. Uruchomienie aplikacji

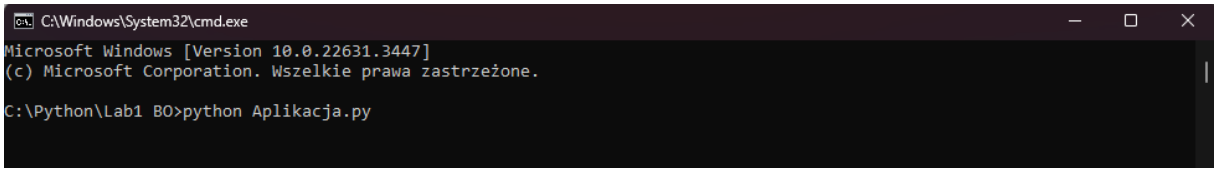
W celu uruchomienia aplikacji przechodzimy do dogodniej dla nas lokalizacji i wklejamy tam program Aplikacja.py.



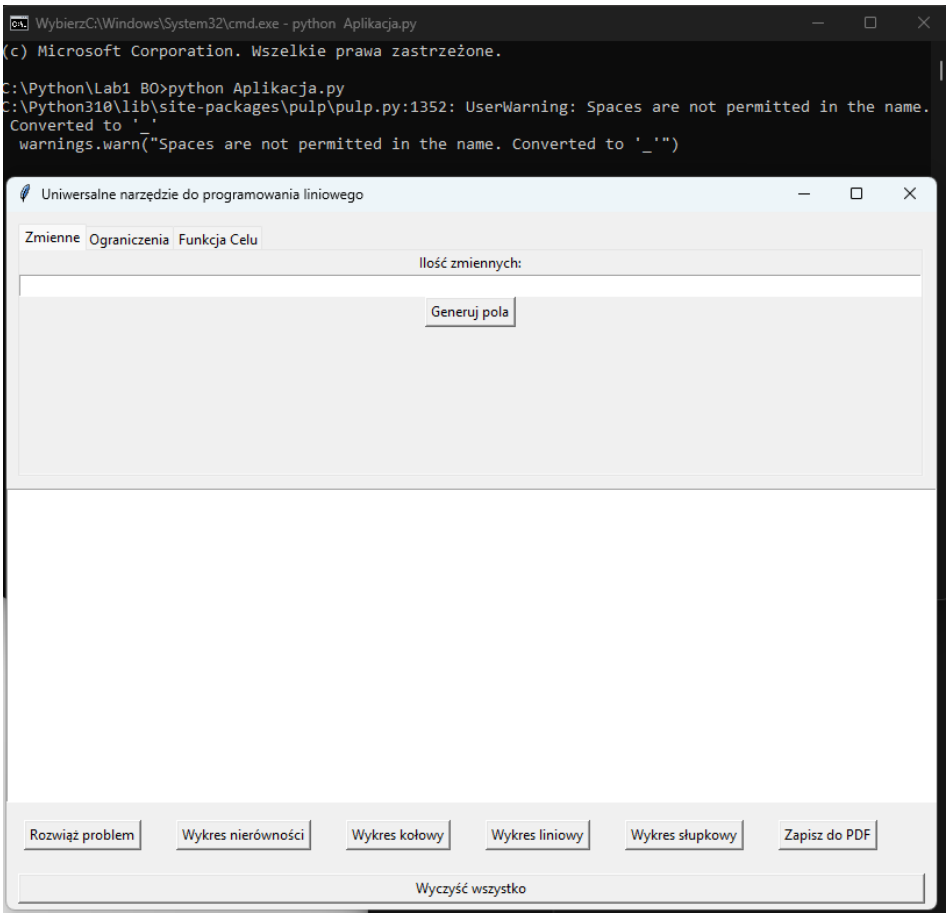
Następnie klikamy na ścieżkę powyżej i wpisujemy cmd



W wierszu poleceń wpisujemy polecenie python Aplikacja.py i klikamy enter



Aplikacja uruchamia się i możemy z niej w pełni korzystać



Aplikacje możemy uruchomić również za pomocą Jupyter Notebook (proces uruchomienia na początku). W lokalizacji wybieramy Aplikacja.ipynb

Następnie klikamy przycisk Run i aplikacja po chwili uruchomi się

jupyter Aplikacja (autosaved)

Logout

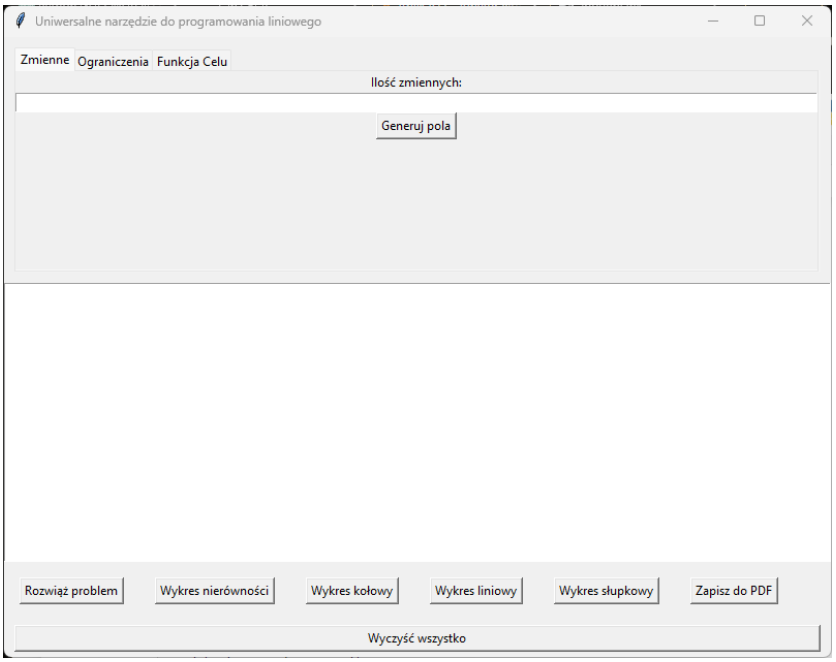
File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

run cell, select below

In [12]:
```
import sympy as sp
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter import filedialog
from pulp import LpVariable, LpProblem, lpSum, LpMinimize, LpMaximize
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.backends.backend_pdf import PdfPages
import numpy as np
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.lines import Line2D
import matplotlib.colors as mcolors
from matplotlib.patches import Polygon


root = tk.Tk()
root.title("Uniwersalne narzędzie do programowania liniowego")
root.geometry("800x600")

zmienne = {}
ograniczenia = []
```



9. Wnioski końcowe

Praca laboratoryjna skupiła się na zaprojektowaniu i implementacji aplikacji do rozwiązywania zadań z zakresu programowania liniowego, co jest ważną częścią badań operacyjnych. Aplikacja, zrealizowana w języku Python z wykorzystaniem narzędzia Jupyter Notebook, została wyposażona w funkcjonalność umożliwiającą użytkownikom definiowanie problemów optymalizacyjnych poprzez wprowadzanie zmiennych, ustawianie ograniczeń i określanie funkcji celu. W trakcie pracy z aplikacją użytkownik może generować różne rodzaje wykresów, w tym wykresy nierówności, liniowe, kołowe oraz słupkowe, które nie tylko wizualizują problem, ale również ułatwiają zrozumienie zależności i wyników optymalizacji. Ponadto aplikacja umożliwia zapisywanie wykonanych działań i wyników do formatu PDF, co jest przydatne dla dokumentacji i dalszej analizy. Zasadniczo, praca ta ukazuje, jak narzędzia takie jak Python i Jupyter Notebook mogą być wykorzystywane do tworzenia aplikacji o szerokim zastosowaniu praktycznym, co ma istotne znaczenie nie tylko w kontekście akademickim, ale i profesjonalnym, szczególnie w dziedzinach wymagających optymalizacji zasobów, jak logistyka, finanse czy inżynieria. Podsumowując, aplikacja stworzona w ramach pracy laboratorium stanowi uniwersalne narzędzie, które umożliwia kompleksowe podejście do problemów optymalizacyjnych - od ich definiowania, przez rozwiązywanie, aż po analizę i prezentację wyników. Narzędzie to może być użyteczne dla studentów, inżynierów, analityków i naukowców zajmujących się optymalizacją liniową w różnych aplikacjach praktycznych.