

Dokumentacja projektu zespołowego nr 1

Anna Ćwiklińska, Krystian Gronkowski, Ihor Malyi

Marzec 2023

1 Treść zadania

Obliczyć $\sqrt{23}$ za pomocą wielomianów interpolacyjnych dla danych z tabeli:

x	0	1	4	9	16	25	36	49	64	81
$f(x)$	0	1	2	3	4	5	6	7	8	9

Sprawdzić, jaki podzbiór danych z tabeli daje najlepsze przybliżenie dokładnej wartości pierwiastka (czyli dla jakiego zestawu tych węzłów wielomian Lagrange’a przebiega najbliżej punktu $(23, \sqrt{23})$).

2 Teoretyczny opis metody

W realizacji naszego projektu, w celu obliczenia przybliżonej wartości pierwiastka zastosujemy wielomian interpolacyjny Lagrange’a. Wielomian ten będzie konstruowany na podstawie wybranego podzbioru danych z tabeli, które zostaną wybrane w sposób optymalny, tak aby przybliżenie było jak najbardziej dokładne.

2.1 Wielomian interpolacyjny w postaci Lagrange’a

Niech $n + 1$ będzie liczbą węzłów interpolacyjnych, a x_0, x_1, \dots, x_n są ich wartościami. Niech f będzie funkcją, którą chcemy interpolować na przedziale $[x_0, x_n]$. Wielomian interpolacyjny $P(x)$ dla f w węzłach x_0, x_1, \dots, x_n to

unikalny wielomian stopnia co najwyżej n , który spełnia $P(x_i) = f(x_i)$ dla $i = 0, 1, \dots, n$.

Metoda Lagrange'a polega na wyznaczeniu wielomianu interpolacyjnego za pomocą wzoru:

$$P(x) = \sum_{i=0}^n f(x_i) L_i(x)$$

gdzie $L_i(x)$ to i -ty wielomian Lagrange'a, zdefiniowany jako:

$$L_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}$$

Przykład 1 *Przyjmijmy, że wybieramy podzbiór danych*

$$(1, 1), (4, 2), (9, 3), (16, 4)$$

czyli $x_0 = 1, x_1 = 4, x_2 = 9$ i $x_3 = 16$, oraz odpowiadające im wartości funkcji $f(x)$. W celu skonstruowania wielomianu interpolacyjnego Lagrange'a dla tego podzbioru danych, należy obliczyć:

$$L_0(x) = \frac{(x-4)(x-9)(x-16)}{(1-4)(1-9)(1-16)} = -\frac{1}{360}(x^3 - 29x^2 + 244x - 576),$$

$$L_1(x) = \frac{(x-1)(x-9)(x-16)}{(4-1)(4-9)(4-16)} = \frac{1}{180}(x^3 - 26x^2 + 169x - 144),$$

$$L_2(x) = \frac{(x-1)(x-4)(x-16)}{(9-1)(9-4)(9-16)} = -\frac{1}{280}(x^3 - 21x^2 + 84x - 64),$$

$$L_3(x) = \frac{(x-1)(x-4)(x-9)}{(16-1)(16-4)(16-9)} = \frac{1}{1260}(x^3 - 14x^2 + 49x - 36).$$

Wielomian interpolacyjny dla wybranego podzbioru danych z tabeli to $P(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) + f(x_3)L_3(x)$, czyli:

$$\begin{aligned}
P(x) &= 1 \cdot \left(-\frac{1}{360}(x^3 - 29x^2 + 244x - 576) \right) \\
&\quad + 2 \cdot \frac{1}{180}(x^3 - 26x^2 + 169x - 144) \\
&\quad + 3 \cdot \left(-\frac{1}{280}(x^3 - 21x^2 + 84x - 64) \right) \\
&\quad + 4 \cdot \frac{1}{1260}(x^3 - 14x^2 + 49x - 36) \\
&= \frac{1}{1260}x^3 - \frac{1}{36}x^2 + \frac{41}{90}x + \frac{4}{7}.
\end{aligned}$$

Aby sprawdzić, jak dobrze wybrany podzbiór danych z tabeli przybliży dokładną wartość pierwiastka $\sqrt{23}$, możemy porównać wartość wielomianu $P(x)$ dla $x = 23$ z wartością $\sqrt{23}$:

$$P(23) = \frac{1}{1260}(23)^3 - \frac{1}{36}(23)^2 + \frac{41}{90}(23) + \frac{4}{7} \approx 6,0111111s,$$

co jest dość odległym przybliżeniem wartości $\sqrt{23}$.

3 Opis programu

Program został napisany w języku Python w wersji 3. Celem programu jest znalezienie wielomianu Lagrange’a, który najlepiej przybliży dane wejściowe dla danego punktu $(23, \sqrt{23})$. Wyniki są porównywane do wartości funkcji `sqrt(23)` zaimportowanej z biblioteki `math`.

Program korzysta z kilku bibliotek oraz modułów, takich jak:

- `math`
- `itertools`
- `matplotlib`
- `os`
- `unittest`

3.1 Opis implementacji algorytmu

1. W funkcji `main()` program rozpoczyna się od wczytania danych z pliku tekstowego o nazwie `"data.txt"` za pomocą funkcji `import_data()` i zapisuje je w zmiennej `data`.
2. Następnie tworzony jest obiekt klasy `Polynomials` na podstawie wczytanych danych.
3. Tworzony jest również obiekt klasy `DataSets` i przekazywane do niego wczytane dane.
4. W klasie `DataSets` generowane są wszystkie możliwe kombinacje podzbiorów danych o różnych rozmiarach za pomocą funkcji `generate_subsets()`.
5. Dla każdego wygenerowanego podzbioru danych tworzony jest obiekt klasy `Data`, który reprezentuje dany podzbiór. Klasa `Data` przetwarza ten podzbiór na listę obiektów klasy `Pair`.
6. W funkcji `main()` wywoływana jest metoda `find_best()` na obiekcie klasy `Polynomials`, która zwraca listę wielomianów posortowaną od najlepszego dopasowania do najgorszego.
7. Tworzony jest obiekt klasy `Polynome`, który reprezentuje najlepszy wielomian Lagrange'a. Do tego obiektu przekazywane są współczynniki oraz dane wejściowe.
 - (a) Tworzenie obiektu klasy `LagrangeMultipliers`: Program tworzy obiekt klasy `LagrangeMultipliers` za pomocą konstruktora tej klasy, który przyjmuje obiekt klasy `Data` jako argument wejściowy. Wewnątrz konstruktora tworzony jest zestaw wielomianów Lagrange'a za pomocą obiektów klasy `LagrangeMultiplier`, które są przechowywane jako lista w atrybucie `self.multipliers` w obiekcie klasy `LagrangeMultipliers`.
 - (b) Tworzenie obiektów klasy `LagrangeMultiplier`: Dla każdego indeksu `i` w zakresie od 0 do długości danych wejściowych, program tworzy obiekt klasy `LagrangeMultiplier` za pomocą konstruktora tej klasy, który przyjmuje indeks `i` oraz obiekt klasy `Data` jako dane wejściowe. Wewnątrz konstruktora `LagrangeMultiplier`

tworzony jest pojedynczy wielomian Lagrange’a za pomocą obiektów klasy `Multiplier`, które są przechowywane jako lista w atrybucie `self.multipliers` w obiekcie klasy `LagrangeMultiplier`.

- (c) Tworzenie obiektów klasy `Multiplier`: Dla każdej pary danych (x, y) w danych wejściowych dla danego indeksu i , program tworzy obiekt klasy `Multiplier` za pomocą konstruktora tej klasy, który przyjmuje wartości `x_k` i `x_i` jako argumenty wejściowe, gdzie `x_k` jest wartością x dla danej pary, a `x_i` jest wartością x dla indeksu, dla którego tworzony jest wielomian Lagrange’a.
 - (d) Obliczanie wartości interpolowanego wielomianu Lagrange’a: Po utworzeniu obiektów klasy `LagrangeMultiplier`, program w ywołuje w klasie `Polynome` metodę `calc(x)` na każdym z obiektów klasy `Multiplier`, przekazując jej wartość 23, dla której ma zostać obliczona wartość interpolowanego wielomianu Lagrange’a.
8. Na koniec program wypisuje na ekranie najlepszy wielomian Lagrange’a wraz z jego współczynnikami, na podstawie obiektu klasy `Polynomial` oraz generuje raport w pliku `.tex` i wykres funkcji.
 9. Program kończy swoje działanie.

4 Instrukcja użytkownika

Do prawidłowego działania programu wymagany jest zainstalowany interpreter języka Python 3. Należy również upewnić się, że struktura plików programu jest kompletna, a w szczególności, że plik z danymi wejściowymi (opisany szczegółowo poniżej) znajduje się we właściwym miejscu.

Aby uruchomić program, należy:

1. Upewnić się, że mamy zainstalowane biblioteki wymagane do prawidłowego uruchomienia programu (w szczególności `matplotlib`), oraz że program jest kompletny i istnieje folder `reports`.
2. Otworzyć terminal lub wiersz poleceń na swoim komputerze.
3. Przejść do katalogu, w którym znajduje się plik `main.py`.
4. (Opcjonalnie) Zmienić dane wejściowe w pliku `data.txt`

5. Uruchomić program komputerowy za pomocą: `python main.py` (lub `py main.py`)

Jeśli wszystko się powiodło, program wczyta dane z pliku i wykona się. W terminalu powinny wyświetlić się dane wyjściowe (opisane poniżej). W folderze `reports` powstanie nowy plik w formacie `.tex`, którego tytuł będzie połączeniem słowa `report` oraz daty i godziny wykonania programu, a także plik `.png` z wygenerowanym wykresem.

4.1 Dane wejściowe

Program oczekuje na dane wejściowe dostarczone w pliku `data.txt`, który powinien znajdować się w tym samym katalogu co program. Domyślnie w pliku znajdują się pełne dane z [tabeli](#) z punktu 1. Poprzez modyfikację pliku `dane.txt` użytkownik może zmieniać zakres istniejących danych wejściowych oraz dodawać nowe.

Dane wejściowe powinny być podane w formacie `x y`, gdzie `x` i `y` są odpowiednio współrzędnymi (x, y) danego węzła i są oddzielone spacją. Przykładowy format danych wejściowych przedstawia się następująco:

```
1 1
4 2
9 3
16 4
25 5
```

4.2 Walidacja danych wejściowych

Za proces importu danych do programu oraz ich walidację odpowiada funkcja `import_data` zlokalizowana w pliku `getData.py`.

Funkcja sprawdza każdą linię, czy zawiera dwie liczby całkowite oddzielone spacją, a następnie dodaje te liczby do listy `data`.

Jeśli program napotka pustą linię, lub spacji w linijce jest więcej niż jedna, nie wpływa to na działanie programu.

Jeśli plik nie istnieje lub linijka nie zawiera dwóch liczb, funkcja wypisuje odpowiedni komunikat o błędzie i program kończy działanie.

W przypadku niepowodzenia konwersji wartości na liczby całkowite, również wypisuje komunikat o błędzie i kończy działanie.

Gdy wszystko się powiedzie, funkcja na końcu zwraca listę z danymi.

4.3 Dane wyjściowe

Program generuje raport w formacie `.tex` w którym znajdują się wypisane kolejne mnożniki Lagrange’a, wzór ogólny oraz wykres prezentujący odległość wielomianu od funkcji pierwiastka drugiego stopnia. Zwraca również wyniki w postaci tekstu w terminalu – zestawu danych w kolejności od najlepszego do najgorszego dopasowania.

Wynik w terminalu zostanie zwrócony w postaci:

$$(x_1, y_1), (x_2, y_2) \dots (x_n, y_n) \rightarrow \text{wynik}$$

Na przykład:

`(1, 1), (4, 2), (64, 8) -> 6.785185185185185`

gdzie:

`(1, 1), (4, 2), (64, 8)` to dane wejściowe, które najlepiej przybliżają funkcję `sqrt`,

`6.785185185185185` to wynik obliczenia wielomianu Lagrange’a.

5 Struktura plików

```
program
├── lagrange
│   ├── common_formula.py
│   ├── data.py
│   ├── data.txt
│   ├── getData.py
│   ├── lagrange_multiplier.py
│   ├── lagrange_multipliers.py
│   ├── main.py
│   ├── multiplier.py
│   ├── plot.py
│   ├── polynome.py
│   ├── polynomials.py
│   ├── report.py
│   └── tests.py
└── reports
    └── ...
```

5.1 Opis struktury plików

- Folder **program** – folder nadrzędny, zawiera wszystkie pliki programu
 - Folder **lagrange** – zawiera pliki odpowiadające za funkcjonowanie programu
 - * Plik **common_formula.py** pozwala przedstawić wzór wielomianu w postaci ogólnej w wygenerowanym raporcie.
 - * Plik **data.py** zawiera definicje klas **Pair**, **DataSets** i **Data**.
 - * **data.txt** to plik tekstowy służący do przechowywania danych wejściowych programu. Ten plik należy zmodyfikować, chcąc użyć innych danych niż domyślne.
 - * Plik **getData.py** zawiera funkcję importującą dane z pliku oraz sprawdzającą ich poprawność.
 - * Plik **lagrange_multiplier.py** zawiera klasę reprezentującą wielomian Lagrange’a dla jednego punktu danych. Oblicza wartość wielomianu Lagrange’a dla danego x .
 - * Plik **lagrange_multipliers.py** zawiera klasę reprezentującą wielomiany Lagrange’a dla całego zbioru danych
 - * Plik **main.py** zawiera główną funkcję programu.
 - * Plik **multiplier.py** zawiera klasę, która reprezentuje obiekt do obliczania wartości na podstawie podanego wzoru matematycznego.
 - * Plik **plot.py** służy do generowania wykresu.
 - * Plik **polynome.py** zawiera klasę **Polynome**, która reprezentuje wielomian interpolacyjny Lagrange’a dla danego zbioru danych.
 - * Plik **polynomials.py** zawiera klasę reprezentującą zbiór wielomianów interpolacyjnych Lagrange’a dla różnych zbiorów danych.
 - * Plik **report.py** odpowiada za generowanie raportów dla najlepszych dopasowań wielomianów.
 - * Plik **tests.py** zawiera kod do testowania funkcji w innych modułach
 - Folder **reports** – zawiera raporty w formacie **.tex** wygenerowane przez kolejne uruchomienia programu.

6 Raport z demonstracji

Po przejściu do folderu i uruchomieniu programu (w środowisku Windows) w terminalu wyświetlają się dane. Są to wyżej wspomniane węzły i wartość $\sqrt{23}$ uzyskana w wyniku interpolacji tych węzłów. Uszeregowane są rosnąco według różnicy pomiędzy ich wynikiem a rzeczywistą wartością $\sqrt{23}$.

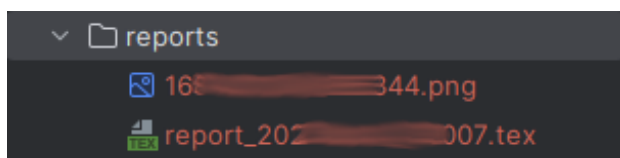
```
Terminal Local x + v
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\> . cd lagrange
PS C:\> lagrange> py main.py
(9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9) -> 4.796270197893973
(16, 4), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9) -> 4.795313510926255
(9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8) -> 4.7965011223344565
(9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (81, 9) -> 4.796596871492706
(16, 4), (25, 5), (36, 6), (49, 7), (64, 8) -> 4.795005611672278
(4, 2), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9) -> 4.796771319638968
(16, 4), (25, 5), (36, 6), (49, 7), (81, 9) -> 4.794877946127946
(9, 3), (16, 4), (25, 5), (36, 6), (64, 8), (81, 9) -> 4.796785337030436
(4, 2), (9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9) -> 4.7948670570079885
(9, 3), (16, 4), (25, 5), (36, 6), (49, 7) -> 4.797011784511785
(16, 4), (25, 5), (36, 6), (64, 8), (81, 9) -> 4.794626658744306
(4, 2), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8) -> 4.7972488776655435
(4, 2), (9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8) -> 4.794407407407408
(9, 3), (16, 4), (25, 5), (49, 7), (64, 8), (81, 9) -> 4.797300476166897
(9, 3), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9) -> 4.794356823958541
(9, 3), (16, 4), (25, 5), (36, 6), (64, 8) -> 4.797306397306399
(16, 4), (25, 5), (36, 6), (49, 7) -> 4.7942760942760945
(4, 2), (9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (81, 9) -> 4.7942168209876534
(4, 2), (16, 4), (25, 5), (36, 6), (49, 7), (81, 9) -> 4.797446889530224
(9, 3), (16, 4), (25, 5), (36, 6), (81, 9) -> 4.79752246689114
(16, 4), (25, 5), (49, 7), (64, 8), (81, 9) -> 4.793939806562357
(1, 1), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9) -> 4.7977234509544555
(16, 4), (25, 5), (36, 6), (64, 8) -> 4.79385521885522
(4, 2), (9, 3), (16, 4), (25, 5), (36, 6), (64, 8), (81, 9) -> 4.793841684822078
(4, 2), (16, 4), (25, 5), (36, 6), (64, 8), (81, 9) -> 4.797836641390564
(25, 5), (36, 6), (49, 7), (64, 8), (81, 9) -> 4.797864676173502
(1, 1), (9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9) -> 4.793727005038126
```

Rysunek 1: Terminal

Można zauważyć, że w katalogu reports po uruchomieniu programu znalazły się dwa nowe pliki - raport w formacie `.tex` oraz wygenerowany wykres w formacie `.png`



Rysunek 2: Nowe pliki w folderze reports

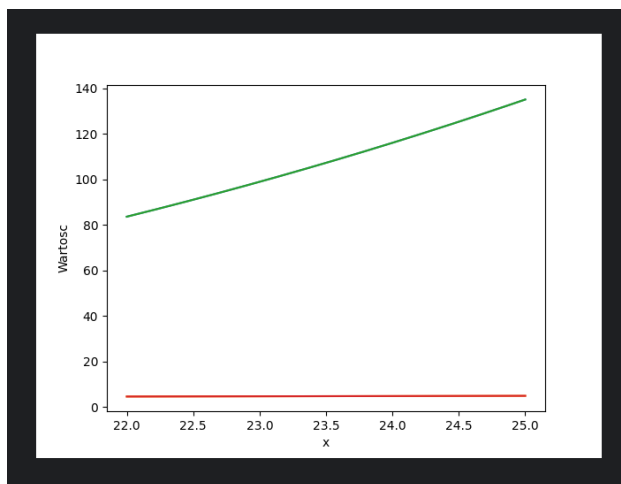
Zawartość pliku `.tex` przed kompilacją do `.pdf` w zależności od podanych danych będzie wyglądać podobnie do zrzutu ekranu poniżej:

```
\documentclass{article}
\usepackage{polski}
\usepackage{graphics}
\begin{document}
Obliczenie najlepszego przybliżenia \sqrt{25}:\\[0.25cm]
\begin{align*}
& \$l_0(x)=\frac{x-1}{0-1}\frac{0}{0-4}\frac{0}{0-9}\frac{0}{0-9} \\
& \$l_1(x)=\frac{x-0}{1-0}\frac{0}{1-4}\frac{0}{1-9}\frac{0}{1-9} \\
& \$P(x)=0 \\
& \$l_2(x)=\frac{x-1}{0-1}\frac{0}{0-4}\frac{0}{0-9}\frac{0}{0-9} \\
& \$l_3(x)=\frac{x-0}{1-0}\frac{0}{1-4}\frac{0}{1-9}\frac{0}{1-9}
\end{align*}
Wzór ogólny wielomianu:\\[0.25cm]
\begin{split}
& \$P(x) = \$1.23333333333333334x^{1} - \$0.26999999999999994x^{2} + \$0.01666666666666663x^{3}
\end{split}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{1681471206.895344.png}
\caption{Wykres}
\label{fig:zajeciel}
\end{figure}
\end{document}
```

Rysunek 3: Nieskompilowany plik `.tex`

Powstaje również plik w formacie `.png` przedstawiający wykres wielomianu oraz wykres funkcji pierwiastka drugiego stopnia.



Rysunek 4: Wykres

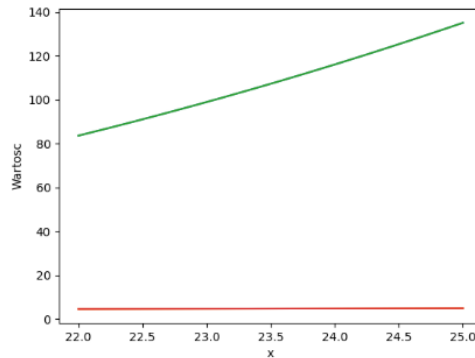
Wcześniej wspomniany plik `.tex` po skompilowaniu w odpowiednim programie (z użyciem właściwej ścieżki do pliku z wykresem) przedstawia się następująco:

Obliczenie najlepszego przybliżenia $\sqrt{23}$:

$$\begin{aligned} l_0(x) &= \frac{x-1}{0-1} \cdot \frac{x-4}{0-4} \cdot \frac{x-9}{0-9} \\ l_1(x) &= \frac{x-0}{1-0} \cdot \frac{x-4}{1-4} \cdot \frac{x-9}{1-9} \\ l_2(x) &= \frac{x-0}{4-0} \cdot \frac{x-1}{4-1} \cdot \frac{x-9}{4-9} \\ l_3(x) &= \frac{x-0}{9-0} \cdot \frac{x-1}{9-1} \cdot \frac{x-4}{9-4} \\ P(x) &= 0 \cdot l_0 + 1 \cdot l_1 + 2 \cdot l_2 + 3 \cdot l_3 \end{aligned}$$

Wzór ogólny wielomianu:

$$\begin{aligned} P(x) &= 1.233333333333334x^1 - \\ &0.2499999999999994x^2 + \\ &0.01666666666666663x^3 \end{aligned}$$



Rysunek 1: Wykres

Rysunek 5: Skompilowany plik .tex

7 Wnioski i interpretacja wyników

Poprzez użycie programu można również zbadać, które kombinacje węzłów interpolacyjnych dają najlepsze przybliżenie do dokładnej wartości pierwiastka $\sqrt{23}$.

Zestaw węzłów interpolacyjnych, dla którego wielomian interpolacyjny Lagrange'a najlepiej przybliża wartość pierwiastka $\sqrt{23}$, można uznać za najlepszy.

Według uzyskanych wyników programu dla podstawowych danych z treści polecenia, najlepszym dopasowaniem jest pierwszy od góry zbiór węzłów wyświetlający się w terminalu:

(9, 3), (16, 4), (25, 5), (36, 6), (49, 7), (64, 8), (81, 9)

Oznacza to, że wielomian o wzorze ogólnym

$$\begin{aligned} P(x) = & 1.1402714932126838 + \\ & 0.2606892943657601x^1 - \\ & 0.0074810727353621445x^2 + \\ & 0.00018834897741147431x^3 - \\ & 2.9299914410208595 \times 10^{-06}x^4 + \\ & 2.4627016048587456 \times 10^{-08}x^5 - \\ & 8.501846737141977 \times 10^{-11}x^6 \end{aligned} \tag{1}$$

uzyskany dzięki interpolacji tych węzłów przebiega najbliższej punktu $(23, \sqrt{23})$ ze wszystkich możliwych wielomianów dla tych danych (co można również zobaczyć na generowanym wykresie).

8 Kod programu

8.1 data.py

```
1 class Pair:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     def __str__(self):
7         return f"({self.x}, {self.y})"
8
9     def __repr__(self):
10        return f"{self.x}, {self.y}"
11
12
13 class DataSets:
14     def __init__(self, data_raw: list):
15         self.data_raw = data_raw
16         self.subsets = []
17
```

```

18         self.__generate_datasets()
19         self._class_size = len(self.subsets)
20         self._current_index = 0
21
22     def __len__(self):
23         return len(self.subsets)
24
25     def __iter__(self):
26         return self
27
28     def __next__(self):
29         if self._current_index < self._class_size - 1:
30             self._current_index += 1
31             return self.subsets[self._current_index]
32         raise StopIteration
33
34     def __generate_datasets(self):
35         for i in range(2, len(self.data_raw)):
36             subsets = self.__findsubsets(i)
37             for subset in subsets:
38                 self.subsets.append(Data(subset))
39         return self.subsets
40
41     def __findsubsets(self, n: int) -> list:
42         return list(itertools.combinations(self.data_raw, n))
43
44
45 class Data:
46     def __init__(self, data_raw):
47         self.data = []
48         for x_y in data_raw:
49             self.data.append(Pair(x_y[0], x_y[1]))
50
51     def get_data(self, i):
52         return self.data[:i] + self.data[i+1:]
53
54     def __str__(self):

```

```

55         return ', '.join(list(map(lambda row: str(row), self.data)))
56
57     def __repr__(self):
58         return '; '.join(list(map(lambda row: str(row), self.data)))
59
60     def to_list(self):
61         tmp = []
62         for pair in self.data:
63             tmp.append((pair.x, pair.y))
64         return tmp

```

8.2 getData.py

```

1  def import_data():
2      data = []
3      try:
4          with open("./data.txt") as f:
5              for line in f:
6                  line = line.strip()
7                  if not line:
8                      continue
9                  values = line.split()
10                 if len(values) != 2:
11                     print("Błąd: wiersz nie zawiera dwóch liczb
12                        ↪ oddzielonych spacją!")
13                     exit()
14                 try:
15                     x, y = int(values[0]), int(values[1])
16                 except ValueError:
17                     print("Błąd: nie można przekonwertować wartości na
18                        ↪ liczby całkowite!")
19                     exit()
20                 data.append([x, y])
21     except FileNotFoundError:
22         print("Błąd: plik nie istnieje!")

```

```
21         exit()
22     return data
```

8.3 lagrange_multiplier.py

```
1  class LagrangeMultiplier:
2      def __init__(self, i: int, data: Data):
3          self.multipliers = []
4          self.i = i
5          for pair in data.get_data(i):
6              x_i = data.data[i].x
7              self.multipliers.append(Multiplier(pair.x, x_i))
8
9      def calc(self, x: int):
10         res = 1
11         for multiplier in self.multipliers:
12             res = res * multiplier.calc(x)
13         return res
14
15     def __to_str(self):
16         mult = list(map(lambda m: str(m), self.multipliers))
17         res = "\cdot".join(mult)
18         res = f"l_{self.i}(x)={res}"
19         res = res.replace('--', '+')
20         return res
21
22     def __str__(self):
23         return self.__to_str()
24
25     def __repr__(self):
26         return self.__to_str()
```

8.4 lagrange_multipliers.py

```
1 class LagrangeMultipliers:
2     def __init__(self, data: Data):
3         self.multipliers = []
4         for i in range(len(data.data)):
5             self.multipliers.append(LagrangeMultiplier(i, data))
```

8.5 main.py

```
1 def main():
2     p = Polynomials(import_data())
3     best = p.find_best()
4     print(f"sqrt(23) = {math.sqrt(23)}")
5     for b in best:
6         print(b)
7     best[0].report.generate()
8     best[0].report.save()
9
10
11 if __name__ == "__main__":
12     main()
```

8.6 multiplier.py

```
1 class Multiplier:
2     def __init__(self, x_k, x_i):
3         self.x_k = x_k
4         self.x_i = x_i
5         self.value = None
6
7     def calc(self, x):
8         self.value = (x - self.x_k) / (self.x_i - self.x_k)
```



```

9         return self.value
10
11     def __str__(self):
12         return f"\\frac{{{x-{{self.x_k}}}}{{ {{self.x_i}}-{{self.x_k}}}}}"
13
14     def __repr__(self):
15         return f"\\frac{{{x-{{self.x_k}}}}{{ {{self.x_i}}-{{self.x_k}}}}}"

```

8.7 plot.py

```

1 class Plot:
2     def __init__(self, polynome):
3         self.polynome = polynome
4         self.__filename = None
5
6     @property
7     def filename(self):
8         if not self.__filename:
9             self.__filename = str(datetime.datetime.now().timestamp()) +
10                 ↪ '.png'
11             return self.__filename
12
13     def calc_points(self, func):
14         end = 25
15         x = []
16         y = []
17         i = 22
18         while i < end:
19             x.append(i)
20             y.append(func(i))
21             i += 0.0001
22         return x, y
23
24     def plot(self):
25         for f in [self.polynome.calc, math.sqrt]:

```

```

25         x, y = self.calc_points(f)
26         plt.plot(x, y, label=f)
27         plt.ylabel('Wartosc')
28         plt.xlabel('x')
29
30     with open('../reports/' + self.filename, 'wb') as f:
31         plt.savefig(f)

```

8.8 polynome.py

```

1 class Polynome:
2     def __init__(self, data: Data):
3         self.data = data
4         self.report = Report(self)
5         self.common_formula = CommonPolynomeFormula(self.data).policz()
6         self.multipliers = LagrangeMultipliers(data)
7         self.plot = Plot(self)
8
9     def format_common_formula(self):
10        formatted_terms = []
11        for term in self.common_formula.split():
12            if 'e' in term:
13                coeff, exp = term.split('e')
14                formatted_term = f'{coeff} \\times 10^{{{exp}}}'
15            else:
16                formatted_term = term
17            formatted_terms.append(formatted_term)
18
19        formatted_formula = ''
20        for i, term in enumerate(formatted_terms):
21            formatted_formula += term
22            if formatted_formula.endswith(('+', '-')):
23                formatted_formula += '\\\\indent'
24        return formatted_formula
25

```

```

26     def calc(self, x):
27         tmp = 0
28         for multiplier, data_ in zip(self.multipliers.multipliers,
29                                     ↪ self.data.data):
29             tmp = tmp + multiplier.calc(x) * data_.y
30         return tmp
31
32     @property
33     def value(self):
34         return self.calc(23)
35
36     def __str__(self):
37         return f"{self.data} -> {self.value}"
38
39     def __repr__(self):
40         return f"{self.data} -> {self.value}"

```

8.9 polynomials.py

```

1 class Polynomials:
2
3     def __init__(self, datasets: list):
4         self.datasets = DataSets(datasets)
5
6     def find_best(self):
7         polynomials = []
8         for data in self.datasets:
9             p = Polynome(data)
10            polynomials.append(p)
11            best = sorted(polynomials, key=lambda p: abs(p.value - sqrt(23)))
12            return best

```

8.10 report.py

```
1 class Report:
2     def __init__(self, polynome):
3         self.polynome = polynome
4
5     def save(self):
6         now = datetime.now()
7         current_datetime = now.strftime("%Y%m%d_%H%M%S")
8         filename = f"report_{current_datetime}.tex"
9         file_path = os.path.join('../reports/', filename)
10        with open(file_path, 'w') as f:
11            f.write(self.generate())
12
13    def __generate_multipliers(self):
14        multipliers_view = []
15
16        for multiplier in self.polynome.multipliers.multipliers:
17            multipliers_view.append(f"${multiplier}$")
18        multipliers_view = ' \\\ ' .join(multipliers_view)
19        return multipliers_view
20
21    def __generate_p(self):
22        res = "$P(x)="
23        terms = []
24        i = 0
25        for x_y in self.polynome.data.data:
26            terms.append(f"{x_y.y}\cdot l_{i}")
27            i += 1
28        res += "+" .join(terms)
29        res += "$"
30        return res
31
32    def generate(self):
33        self.__generate_multipliers()
34        self.polynome.plot.plot()
35        content = f"""\documentclass{{article}}
```

```

36         \\usepackage{{polski}}
37         \\usepackage{{graphicx}}
38         \\begin{{document}}
39
40         Obliczenie najlepszego przybliżenia  $\sqrt{23}$ :\\\\[0.25cm]
41         \\begin{{align*}}
42         {self.__generate_multipliers()} \\\\ [0.25cm]
43         {self.__generate_p()} \\\\ [0.25cm]
44         \\end{{align*}}\\\\ [0.25cm]
45         Wzór ogólny wielomianu:\\\\ [0.25cm]
46         \\begin{{split}}
47         $P(x) = ${self.polynome.format_common_formula()}
48         \\end{{split}}
49
50         \\begin{{figure}}[h]
51             \\centering
52
53         ↪ \\includegraphics[width=\\textwidth]{{{self.polynome.plot.filename}}}}
54             \\caption{{Wykres}}
55             \\label{{fig:zdjecie1}}
56         \\end{{figure}}
57         \\end{{document}}""
58         return content
59
60     def __str__(self):
61         return self.generate()

```

8.11 tests.py

```

1 class TestMultipliers(unittest.TestCase):
2     def setUp(self):
3         data_raw = [
4             [-1, -1],
5             [0, 0],
6             [1, 1],

```

```

7         [2, 2]
8     ]
9
10    self.data = Data(data_raw)
11    self.multipliers = []
12
13    for i in range(len(data_raw) - 1):
14        self.multipliers.append(LagrangeMultiplier(i, self.data))
15
16    self.x = 10
17    self.true_answ = [-120, 396, -440, 165]
18
19    def test_multipliers(self):
20        i = 0
21        for m in self.multipliers:
22            self.assertEqual(self.true_answ[i], m.calc(self.x))
23            i += 1
24
25
26    class TestMultiplier(unittest.TestCase):
27        def setUp(self) -> None:
28            pass
29
30        def test_multiplier(self):
31            self.multiplier = (Multiplier(0, -1))
32            self.x = 10
33            self.true_answ = -10
34            self.assertEqual(self.multiplier.calc(self.x), self.true_answ)
35
36
37    class TestDataSets(unittest.TestCase):
38        def setUp(self) -> None:
39            self.data_raw = [
40                [-1, -1],
41                [0, 0],
42                [1, 1],
43                [2, 2]

```

```
44         ]
45         self.data_sets = DataSets(self.data_raw)
46
47     def test_subsets(self):
48         self.assertEqual(len(self.data_sets), 10)
49
50
51 if __name__ == '__main__':
52     unittest.main()
```

Bibliografia

Danuta Jaruszevska-Walczak - wykład Algorytmy numeryczne