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 $x_0, x_1, \ldots, x_n$ są ich wartościami. Niech f będzie funkcją, którą chcemy interpolować na przedziale $[x_0, x_n]$. Wielomian interpolacyjny $P(x)$ dla f w węzłach $x_0, x_1, \ldots, x_n$ to unikalny wielomian stopnia co najwyżej n, który spełnia $P(x_i) = f(x_i)$ dla $i = 0, 1, \ldots, n$.

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:*

$$\begin{aligned}
L_0(x) &= \frac{(x-4)(x-9)(x-16)}{(1-4)(1-9)(1-16)} = -\frac{1}{360}(x^3 - 29x^2 + 244x - 576), \\
L_1(x) &= \frac{(x-1)(x-9)(x-16)}{(4-1)(4-9)(4-16)} = \frac{1}{180}(x^3 - 26x^2 + 169x - 144), \\
L_2(x) &= \frac{(x-1)(x-4)(x-16)}{(9-1)(9-4)(9-16)} = -\frac{1}{280}(x^3 - 21x^2 + 84x - 64), \\
L_3(x) &= \frac{(x-1)(x-4)(x-9)}{(16-1)(16-4)(16-9)} = \frac{1}{1260}(x^3 - 14x^2 + 49x - 36).
\end{aligned}$$

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:*

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

Aby sprawdzić, jak dobrze wybrany podzbiór danych z tabeli przybliża 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 przybliża 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 użytkowania

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 polecenia 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 linijkę, czy zawiera dwie liczby całkowite oddzielone spacją, a następnie dodaje te liczby do listy `data`.

Jeśli program napotka pustą linijkę, 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:

$$(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n) \rightarrow wynik$$

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

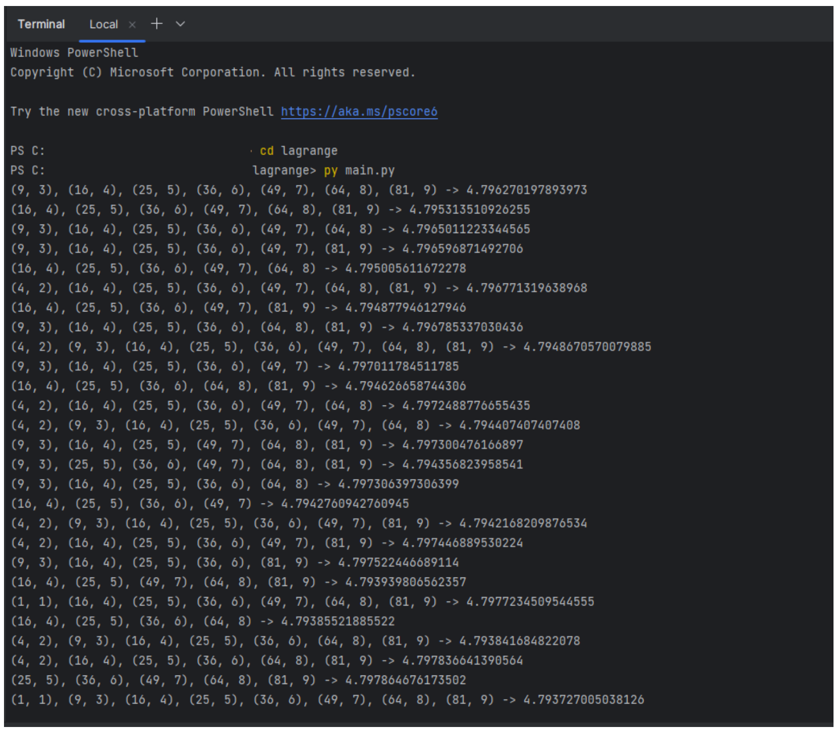
```
program
├── lagrange
│   ├── common_formula.py
│   ├── data.py
│   ├── data.txt
│   ├── getData.py
│   ├── lagrange_multiplier.py
│   ├── lagrange_multipliers.py
│   ├── main.py
│   ├── multiplier.py
│   ├── plot.py
│   ├── polynome.py
│   ├── polynomials.py
│   ├── report.py
│   └── tests.py
└── 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 $\sqrt{23}$.

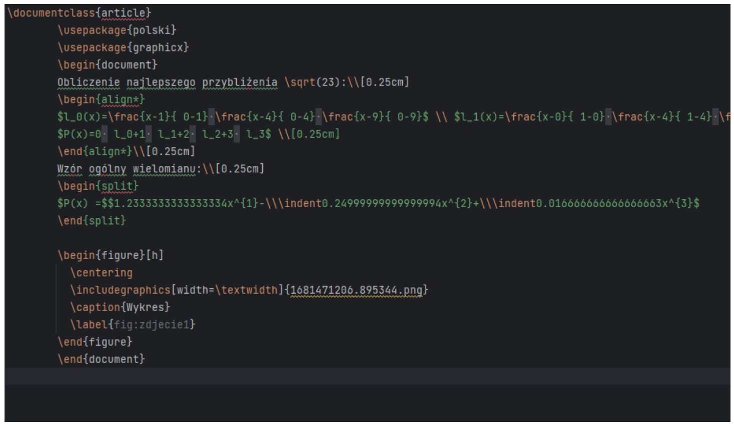


Rysunek 1: Terminal

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

Rysunek 2: Nowe pliki w folderze reports

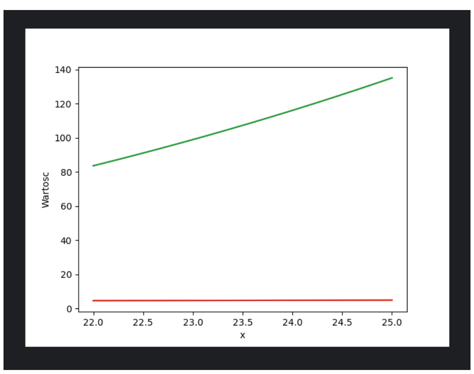
Zawartość pliku `.tex` przed kompilacją do `.pdf` w zależności od podanych danych będzie wyglądać podobnie do zrzutu ekranu poniżej:



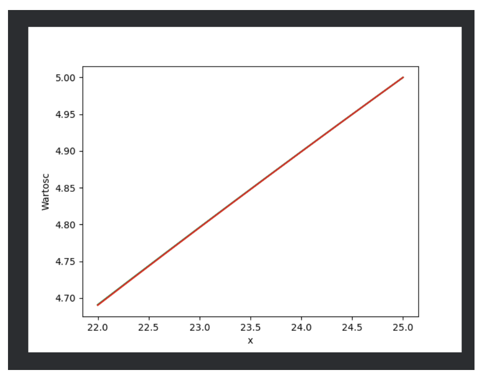
```
\documentclass{article}
        \usepackage{polski}
        \usepackage{graphicx}
        \begin{document}
        Obliczenie najlepszego przybliżenia \sqrt(23):\\[0.25cm]
        \begin{align*}
        $l_0(x)=\frac{x-1}{ 0-1}·\frac{x-4}{ 0-4}·\frac{x-9}{ 0-9}$ \\ $l_1(x)=\frac{x-0}{ 1-0}·\frac{x-4}{ 1-4}·\f
        $P(x)=0· l_0+1· l_1+2· l_2+3· l_3$ \\[0.25cm]
        \end{align*}\\[0.25cm]
        Wzór ogólny wielomianu:\\[0.25cm]
        \begin{split}
        $P(x) =$$1.2333333333333334x^{1}-\\\indent0.24999999999999994x^{2}+\\\indent0.016666666666666663x^{3}$
        \end{split}

        \begin{figure}[h]
          \centering
          \includegraphics[width=\textwidth]{1681471206.895344.png}
          \caption{Wykres}
          \label{fig:zdjecie1}
        \end{figure}
        \end{document}
```

Rysunek 3: Nieskompilowany plik .tex

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



(a) Wykres 1.

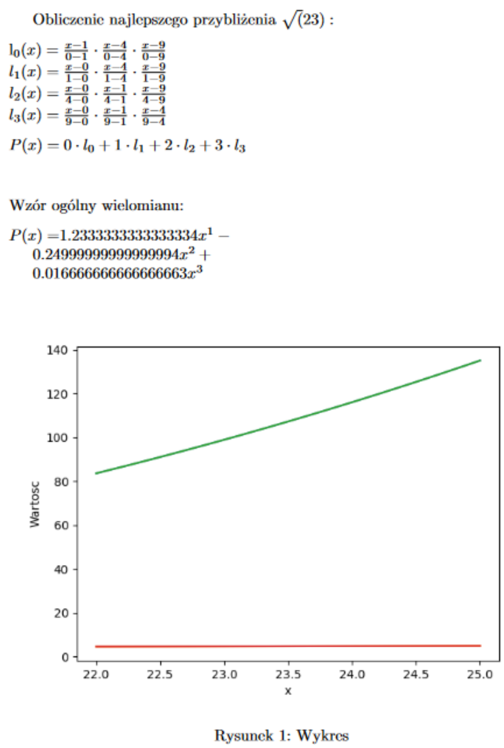


(b) Wykres 2.

Rysunek 4: Porównanie wykresów

Wykres 1. przedstawia źle dopasowany wielomian, podczas gdy na Wykresie 2. wielomian jest dopasowany tak dobrze, że linie praktycznie się pokrywają.

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



Obliczenie najlepszego przybliżenia $\sqrt(23)$:

$l_0(x) = \frac{x-1}{0-1} \cdot \frac{x-4}{0-4} \cdot \frac{x-9}{0-9}$
$l_1(x) = \frac{x-0}{1-0} \cdot \frac{x-4}{1-4} \cdot \frac{x-9}{1-9}$
$l_2(x) = \frac{x-0}{4-0} \cdot \frac{x-1}{4-1} \cdot \frac{x-9}{4-9}$
$l_3(x) = \frac{x-0}{9-0} \cdot \frac{x-1}{9-1} \cdot \frac{x-4}{9-4}$

$P(x) = 0 \cdot l_0 + 1 \cdot l_1 + 2 \cdot l_2 + 3 \cdot l_3$

Wzór ogólny wielomianu:

$P(x) = 1.2333333333333334x^1 - 0.24999999999999994x^2 + 0.016666666666666663x^3$

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

$$\begin{aligned} P(x) = 1.1402714932126838+ \\ 0.2606892943657601x^1- \\ 0.007481072735362144 5x^2+ \\ 0.00018834897741147431x^3- \\ 2.9299914410208595 \times 10^{-06}x^4+ \\ 2.4627016048587456 \times 10^{-08}x^5- \\ 8.501846737141977 \times 10^{-11}x^6 \end{aligned} \tag{1}$$

uzyskany dzięki interpolacji tych węzłów przebiega najbliżej 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

```
class Pair:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"({self.x}, {self.y})"

    def __repr__(self):
        return f"{self.x}, {self.y}"


class DataSets:
    def __init__(self, data_raw: list):
```

```
        self.data_raw = data_raw
        self.subsets = []

        self.__generate_datasets()
        self._class_size = len(self.subsets)
        self._current_index = 0

    def __len__(self):
        return len(self.subsets)

    def __iter__(self):
        return self

    def __next__(self):
        if self._current_index < self._class_size - 1:
            self._current_index += 1
            return self.subsets[self._current_index]
        raise StopIteration

    def __generate_datasets(self):
        for i in range(2, len(self.data_raw)):
            subsets = self.__findsubsets(i)
            for subset in subsets:
                self.subsets.append(Data(subset))
        return self.subsets

    def __findsubsets(self, n: int) -> list:
        return list(itertools.combinations(self.data_raw, n))


class Data:
    def __init__(self, data_raw):
        self.data = []
        for x_y in data_raw:
            self.data.append(Pair(x_y[0], x_y[1]))

    def get_data(self, i):
```

```python
        return self.data[:i] + self.data[i+1:]

    def __str__(self):
        return ', '.join(list(map(lambda row: str(row), self.data)))

    def __repr__(self):
        return '; '.join(list(map(lambda row: str(row), self.data)))

    def to_list(self):
        tmp = []
        for pair in self.data:
            tmp.append((pair.x, pair.y,))
        return tmp
```

8.2 getData.py

```python
def import_data():
    data = []
    try:
        with open("./data.txt") as f:
            for line in f:
                line = line.strip()
                if not line:
                    continue
                values = line.split()
                if len(values) != 2:
                    print("Błąd: wiersz nie zawiera dwóch liczb oddzielonych spacją!")
                    exit()
                try:
                    x, y = int(values[0]), int(values[1])
                except ValueError:
                    print("Błąd: nie można przekonwertować wartości na liczby całkowite!")
                    exit()
```

```
                data.append([x, y])
    except FileNotFoundError:
        print("Błąd: plik nie istnieje!")
        exit()
    return data
```

8.3 lagrange_multiplier.py

```
class LagrangeMultiplier:
    def __init__(self, i: int, data: Data):
        self.multipliers = []
        self.i = i
        for pair in data.get_data(i):
            x_i = data.data[i].x
            self.multipliers.append(Multilplier(pair.x, x_i))

    def calc(self, x: int):
        res = 1
        for multiplier in self.multipliers:
            res = res * multiplier.calc(x)
        return res

    def __to_str(self):
        mult = list(map(lambda m: str(m), self.multipliers))
        res = "\cdot".join(mult)
        res = f"l_{self.i}(x)={res}"
        res = res.replace('--', '+')
        return res

    def __str__(self):
        return self.__to_str()

    def __repr__(self):
        return self.__to_str()
```

8.4 lagrange_multipliers.py

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

8.5 main.py

```python
def main():
    p = Polynomials(import_data())
    best = p.find_best()
    print(f"sqrt(23) = {math.sqrt(23)}")
    for b in best:
        print(b)
    best[0].report.generate()
    best[0].report.save()


if __name__ == "__main__":
    main()
```

8.6 multiplier.py

```python
class Multilplier:
    def __init__(self, x_k, x_i):
        self.x_k = x_k
        self.x_i = x_i
        self.value = None

    def calc(self, x):
        self.value = (x - self.x_k) / (self.x_i - self.x_k)
```

```
        return self.value

    def __str__(self):
        return f"\\frac{{x-{self.x_k}}}{{ {self.x_i}-{self.x_k}}}"

    def __repr__(self):
        return f"\\frac{{x-{self.x_k}}}{{ {self.x_i}-{self.x_k}}}"
```

8.7 plot.py

```
class Plot:
    def __init__(self, polynome):
        self.polynome = polynome
        self.__filename = None

    @property
    def filename(self):
        if not self.__filename:
            self.__filename = str(datetime.datetime.now().timestamp()) +
            ↪  '.png'
        return self.__filename

    def calc_points(self, func):
        end = 25
        x = []
        y = []
        i = 22
        while i < end:
            x.append(i)
            y.append(func(i))
            i += 0.0001
        return x, y

    def plot(self):
        for f in [self.polynome.calc, math.sqrt]:
```

```python
                x, y = self.calc_points(f)
                plt.plot(x, y, label=f)
                plt.ylabel('Wartosc')
                plt.xlabel('x')

        with open('../reports/' + self.filename, 'wb') as f:
            plt.savefig(f)
```

8.8 polynome.py

```python
class Polynome:
    def __init__(self, data: Data):
        self.data = data
        self.report = Report(self)
        self.common_formula = CommonPolynomeFormula(self.data).policz()
        self.multipliers = LagrangeMultipliers(data)
        self.plot = Plot(self)

    def format_common_formula(self):
        formatted_terms = []
        for term in self.common_formula.split():
            if 'e' in term:
                coeff, exp = term.split('e')
                formatted_term = f'{coeff} \\times 10^{{{exp}}}'
            else:
                formatted_term = term
            formatted_terms.append(formatted_term)

        formatted_formula = ''
        for i, term in enumerate(formatted_terms):
            formatted_formula += term
            if formatted_formula.endswith(('+', '-')):
                formatted_formula += '\\\\\\indent'
        return formatted_formula

```

```
    def calc(self, x):
        tmp = 0
        for multiplier, data_ in zip(self.multipliers.multipliers,
        ↪  self.data.data):
            tmp = tmp + multiplier.calc(x) * data_.y
        return tmp

    @property
    def value(self):
        return self.calc(23)

    def __str__(self):
        return f"{self.data} -> {self.value}"

    def __repr__(self):
        return f"{self.data} -> {self.value}"
```

8.9 polynomials.py

```
class Polynomials:

    def __init__(self, datasets: list):
        self.datasets = DataSets(datasets)

    def find_best(self):
        polynomials = []
        for data in self.datasets:
            p = Polynome(data)
            polynomials.append(p)
        best = sorted(polynomials, key=lambda p: abs(p.value - sqrt(23)))
        return best
```

8.10 report.py

```
class Report:
    def __init__(self, polynome):
        self.polynome = polynome

    def save(self):
        now = datetime.now()
        current_datetime = now.strftime("%Y%m%d_%H%M%S")
        filename = f"report_{current_datetime}.tex"
        file_path = os.path.join('../reports/', filename)
        with open(file_path, 'w') as f:
            f.write(self.generate())

    def __generate_multipliers(self):
        multipliers_view = []

        for multiplier in self.polynome.multipliers.multipliers:
            multipliers_view.append(f"${multiplier}$")
        multipliers_view = ' \\\ '.join(multipliers_view)
        return multipliers_view

    def __generate_p(self):
        res = "$P(x)="
        terms = []
        i = 0
        for x_y in self.polynome.data.data:
            terms.append(f"{x_y.y}\cdot l_{i}")
            i += 1
        res += "+".join(terms)
        res += "$"
        return res

    def generate(self):
        self.__generate_multipliers()
        self.polynome.plot.plot()
        content = f"""\documentclass{{article}}
```

```
        \\usepackage{{polski}}
        \\usepackage{{graphicx}}
        \\begin{{document}}

        Obliczenie najlepszego przybliżenia \sqrt(23):\\\\[0.25cm]
        \\begin{{align*}}
        {self.__generate_multipliers()} \\\\[0.25cm]
        {self.__generate_p()} \\\\[0.25cm]
        \\end{{align*}}\\\\[0.25cm]
        Wzór ogólny wielomianu:\\\\[0.25cm]
        \\begin{{split}}
        $P(x) =${self.polynome.format_common_formula()}
        \\end{{split}}

        \\begin{{figure}}[h]
          \\centering
↪   \\includegraphics[width=\\textwidth]{{{self.polynome.plot.filename}}}
          \\caption{{Wykres}}
          \\label{{fig:zdjecie1}}
        \\end{{figure}}
        \\end{{document}}"""
        return content

    def __str__(self):
        return self.generate()
```

8.11 tests.py

```
class TestMultipliers(unittest.TestCase):
    def setUp(self):
        data_raw = [
            [-1, -1],
            [0, 0],
            [1, 1],
```

```
            [2, 2]
        ]

        self.data = Data(data_raw)
        self.multipliers = []

        for i in range(len(data_raw) - 1):
            self.multipliers.append(LagrangeMultiplier(i, self.data))

        self.x = 10
        self.true_answ = [-120, 396, -440, 165]

    def test_multipliers(self):
        i = 0
        for m in self.multipliers:
            self.assertEqual(self.true_answ[i], m.calc(self.x))
            i += 1


class TestMultiplier(unittest.TestCase):
    def setUp(self) -> None:
        pass

    def test_multiplier(self):
        self.multiplier = (Multilplier(0, -1))
        self.x = 10
        self.true_answ = -10
        self.assertEqual(self.multiplier.calc(self.x), self.true_answ)


class TestDataSets(unittest.TestCase):
    def setUp(self) -> None:
        self.data_raw = [
            [-1, -1],
            [0, 0],
            [1, 1],
            [2, 2]
```

```
        ]
        self.data_sets = DataSets(self.data_raw)

    def test_subsets(self):
        self.assertEqual(len(self.data_sets), 10)


if __name__ == '__main__':
    unittest.main()
```

Bibliografia

Literatura

[1] Jaruszewska-Walczak, D. Wykład z Algorytmów numerycznych.

[2] Fortuna, Z., Macukow, B., & Wąsowski, J. *Metody numeryczne*.

[3] Kincaid, D., & Cheney, W. *Analiza numeryczna*.