Uniwersytet Jagielloński w Krakowie

Wydział Fizyki, Astronomii i Informatyki Stosowanej

**Adrian Szumski**

Nr albumu: 1094975

# Implementacja wielomianów w języku Python

Praca magisterska na kierunku Informatyka

Praca wykonana pod kierunkiem
dra hab. Andrzeja Kapanowskiego
Instytut Fizyki

Kraków 2016

1:3994305594

## Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Kraków, dnia                                                                 Podpis autora pracy

## Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Kraków, dnia                                                                 Podpis kierującego pracą



2:5339814146

*Dziękuję Panu doktorowi habilitowanemu Andrzejowi Kapanowskiemu za ogromny wkład oraz zaangażowanie w powstanie mojej pracy magisterskiej.*



3:1012427392

**Streszczenie**

W pracy przedstawiono implementację w języku Python wielomianów jednej i wielu zmiennych. Współczynniki wielomianów mogą być przede wszystkim całkowite i ułamkowe, ale obsługiwane są także współczynniki rzeczywiste i zespolone. Stworzono generatory kilku wielomianów ortogonalnych (Hermite'a, Czebyszewa, Legendre'a), oraz pewnych prostych wielomianów.

Podano podstawy matematyczne potrzebne do zrozumienia teorii baz Gröbnera. Wyjaśniono pojęcia pierścienia wielomianów, porządku dopuszczalnego jednomianów, redukcji wielomianowej, S-wielomianów i baz Gröbnera.

Zaimplementowano algorytm Euklidesa dzielenia wielomianów i algorytm Euklidesa wyznaczania największego wspólnego podzielnika dwóch wielomianów jednej zmiennej. Zaimplementowano algorytm dzielenia uogólnionego wielomianów wielu zmiennych. Zaimplementowano algorytm Buchbergera i jego modyfikacje do obliczania bazy Gröbnera. Algorytmy te są obecne w większości systemów algebry komputerowej.

Przedstawiono zastosowania algorytmu bazy Gröbnera do rozwiązywania układów równań wielomianowych wielu zmiennych.

**Słowa kluczowe:** wielomiany, pierścień wielomianów, baza Gröbnera, algorytm Buchbergera, algorytm Euklidesa, schemat Hornera

4:1023230034

**English title:** Python implementation of polynomials


### Abstract

Python implementation of univariate and multivariate polynomials is presented. The polynomial coefficients are mainly integer or fractions, but float and complex types are also supported. Generators of several orthogonal polynomials are provided (Hermite, Chebyshev, Legendre) together with some simple polynomials.

The mathematical background for understanding the theory of Gröbner basis is given. The concepts of a polynomial ring, an admissible monomial ordering, polynomial reduction, S-polynomials, and a Gröbner basis are explained.

The Euclidean division of polynomials and the Euclidean algorithm for computing the greatest common divisor of two univariate polynomials is implemented. The algorithm for generalized division of multivariate polynomials is also shown. The Buchberger's algorithm for computing Gröbner bases and a modified version of it are given. They are used in most computer algebra systems.

An application of the Gröbner basis algorithm for solving systems of polynomial equations in several variables is presented.



**Keywords:** polynomials, polynomial ring, Gröbner basis, Buchberger's algoritm, Euclidean algorithm, Horner scheme



5:2235682339

# Spis treści


**Spis tabel** . . . . . 4

**Spis rysunków** . . . . . 5

**Listings** . . . . . 6

**1. Wstęp** . . . . . 7

**2. Wprowadzenie do Pythona** . . . . . 8

- 2.1. Typy, arytmetyka i struktury danych . . . . . 9
  - 2.1.1. Typy danych . . . . . 9
  - 2.1.2. Operacje arytmetyczne i bitowe . . . . . 9
  - 2.1.3. Operatory logiczne . . . . . 10
- 2.2. Instrukcje sterujące programem . . . . . 10
  - 2.2.1. Instrukcja warunkowa . . . . . 10
  - 2.2.2. Pętla for . . . . . 11
  - 2.2.3. Pętla while . . . . . 12
- 2.3. Funkcje . . . . . 12
- 2.4. Wyrażenia lambda i generatory . . . . . 13
- 2.5. Moduły i pakiety . . . . . 14
- 2.6. Klasy . . . . . 14
- 2.7. Wyjątki . . . . . 15
- 2.8. Biblioteka standardowa . . . . . 15

**3. Matematyka wielomianów** . . . . . 17

- 3.1. Podstawowe definicje . . . . . 17

**4. Bazy Gröbnera** . . . . . 19

- 4.1. Porządek dopuszczalny . . . . . 19
- 4.2. Pierścień wielomianów . . . . . 20
- 4.3. Baza i zredukowana baza Gröbnera . . . . . 21
- 4.4. Redukcja wielomianowa . . . . . 22
- 4.5. S-wielomiany . . . . . 22
- 4.6. Kryterium Buchbergera . . . . . 22
- 4.7. Zastosowania . . . . . 23

**5. Wielomiany jednej zmiennej** . . . . . 24

- 5.1. Interfejs wielomianów . . . . . 24

**6. Wielomiany wielu zmiennych** . . . . . 28

- 6.1. Interfejs wielomianów . . . . . 28
- 6.2. Zamiana kolejności zmiennych . . . . . 29

**7. Algorytmy** . . . . . 31

- 7.1. Schemat Hornera . . . . . 31
- 7.2. Wielomiany ortogonalne . . . . . 31
- 7.3. Algorytm Euklidesa . . . . . 32
- 7.4. Algorytm dzielenia uogólnionego . . . . . 33




6:4308728465

7.5. Algorytm Buchbergera . . . . . . . . . . . . . . . . . . . . . . . . 34
7.6. Wyznaczanie zredukowanej bazy Gröbnera . . . . . . . . . . . . . . 36
7.7. Ulepszony algorytm Buchbergera . . . . . . . . . . . . . . . . . . . 37
7.8. Algorytmy Faugére . . . . . . . . . . . . . . . . . . . . . . . . . . 40

**8. Systemy algebry komputerowej** . . . . . . . . . . . . . . . . . . . . 41

8.1. Singular . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
8.2. CoCoA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
8.3. Przykładowe obliczenia . . . . . . . . . . . . . . . . . . . . . . . . 41

**9. Podsumowanie** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

**A. Testy wybranych algorytmów** . . . . . . . . . . . . . . . . . . . . . 46

A.1. Działania na wielomianach jednej zmiennej . . . . . . . . . . . . . 46
A.2. Działania na wielomianach wielu zmiennych . . . . . . . . . . . . . 46
A.3. Testy algorytmu Buchbergera . . . . . . . . . . . . . . . . . . . . . 46

**Bibliografia** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49


7:3341908137

# Spis tabel

5.1 Interfejs wielomianów jednej zmiennej. . . . . . . . . . . . . . . . . . . . 26

A.1 Wyniki testów algorytmu Buchbergera. . . . . . . . . . . . . . . . . . . . 48



8:7900106230

# Spis rysunków

A.1 Dodawanie wielomianów jednej zmiennej. . . . . . . . . . . . . . . . . . . 47
A.2 Odejmowanie wielomianów jednej zmiennej. . . . . . . . . . . . . . . . . . 47
A.3 Mnożenie wielomianów jednej zmiennej. . . . . . . . . . . . . . . . . . . 48



9:8450149017

# Listings


2.1 Skrypt hello. . . . 8
2.2 Typy danych w języku Python. . . . 9
2.3 Operacje arytmetyczne w języku Python. . . . 9
2.4 Operacje bitowe w języku Python. . . . 10
2.5 Podstawowe operatory logiczne. . . . 10
2.6 Instrukcja warunkowa. . . . 11
2.7 Pętla **for**. . . . 11
2.8 Pętla **while**. . . . 12
2.9 Funkcje w Pythonie. . . . 12
2.10 Wyrażenia lambda. . . . 13
2.11 Przykład generatora. . . . 13
2.12 Przykłady klas wraz z dziedziczeniem. . . . 14
2.13 Wyjątek ZeroDivisionError. . . . 15
5.1 Sesja interaktywna z wielomianami jednej zmiennej. . . . 27
6.1 Sortowanie wielomianów wielu zmiennych. . . . 28
6.2 Sesja interaktywna z wielomianami wielu zmiennych. . . . 29
6.3 Zamiana kolejności zmiennych. . . . 29
7.1 Schemat Hornera. . . . 31
7.2 Sesja interaktywna z wielomianami ortogonalnymi. . . . 32
7.3 Moduł edivision z algorytmem Euklidesa. . . . 32
7.4 Moduł gdivision z dzieleniem uogólnionym. . . . 33
7.5 Moduł groebner3 z algorytmem Buchbergera. . . . 35
7.6 Moduł groebner4 (wyznaczanie zredukowanej bazy Gröbnera). . . . 36
7.7 Moduł groebner6 (ulepszony algorytm Buchbergera dla bazy zredukowanej). . . . 38
8.1 Obliczenia z użyciem naszego kodu. . . . 42
8.2 Obliczenia w programie SINGULAR. . . . 43
8.3 Obliczenia w programie CoCoA. . . . 43




10:2887995421

# 1. Wstęp

Celem niniejszej pracy jest implementacja wielomianów jednej i wielu zmiennych w języku Python [1]. Wielomian (ang. *polynomial*) to wyrażenie algebraiczne złożone ze zmiennych i stałych, które są połączone działaniami dodawania, odejmowania, mnożenia i podnoszenia do potęgi o stałym wykładniku naturalnym [2].

Wielomiany są używane w wielu działach matematyki. W analizie matematycznej pewne funkcje mogą być przedstawiane w postaci ciągu wielomianów. Wielomiany służą też kodowaniu własności rozmaitych obiektów. Przykładowo wielomian charakterystyczny [3] zawiera informacje o niektórych własnościach macierzy kwadratowej (wartości własne, wyznacznik, ślad). W teorii grafów wielomian chromatyczny [4] zlicza liczbę kolorowań grafu jako funkcję liczby kolorów.

Wielomiany są zamknięte ze względu na operację składania wielomianów, stąd znalazły zastosowanie w teorii złożoności obliczeniowej (ang. *computational complexity theory*), w analizie algorytmów. Problemy są uważane za łatwe, jeżeli istnieją algorytmy o wielomianowym czasie działania, pozwalające te problemy rozwiązać.

Drugim celem pracy, oprócz implementacji wielomianów, jest implementacja algorytmów związanych z wielomianami, w szczególności algorytmów teorii baz Gröbnera (lub Groebnera), zwanych też bazami standardowymi [5]. Bazy te są wykorzystywane do rozwiązywania układów równań wielomianowych wielu zmiennych. Jest to uogólnienie metody eliminacji Gaussa, oraz algorytmu Euklidesa obliczania największego wspólnego dzielnika dwóch wielomianów jednej zmiennej. Algorytmy baz Gröbnera wykorzystywane są w algebrze komputerowej. Popularne programy do obliczeń matematycznych (Mathematica, Maple) pozwalają na obliczanie baz Gröbnera dla różnych porządków jednomianów. Istnieją również wyspecjalizowane programy o większych możliwościach (Singular, CoCoA).

Treść pracy jest zorganizowana w następujący sposób. Rozdział 1 zawiera wprowadzenie do całej pracy. Rozdział 2 omawia najważniejsze elementy języka Python. W rozdziale 3 przypomniano elementarne definicje związane z wielomianami, a rozdział 4 zawiera wprowadzenie do teorii baz Gröbnera. W rozdziałach 5 i 6 przedstawiono implementację odpowiednio wielomianów jednej i wielu zmiennych. W rozdziale 7 zebrano najważniejsze algorytmy omawiane i implementowane w niniejszej pracy. Rozdział 8 przedstawia krótko dwa systemy algebry symbolicznej, mianowicie Singular i CoCoA. Rozdział 9 zawiera podsumowanie pracy.



11:9888688527

# 2. Wprowadzenie do Pythona

Python został stworzony w latach 90-tych przez Guido van Rossuma, holenderskiego programistę i pracownika naukowego, zatrudnionego m.in. w National Research Institute for Mathematics and Computer Science w Amsterdamie, a później w firmie Google. Holender, tworząc język Python jako następcę języka ABC, chciał stworzyć jezyk łatwy i intuicyjny, ale nie odbiegający funkcjonalnie od konkurencyjnych języków programowania. Kod miał być zrozumiały w języku angielskim, co na tamte czasy jak i obecne jest standardem w projektach informatycznych. Język miał być przydatny do różnych celów, od programowania hobbystycznego do dużych projektów komercyjnych lub naukowych. Licencja *open source* miała umożliwiać każdemu rozwijanie języka. Obecnie kody źródłowe języka są dostępne bez opłat na oficjalnej stronie Pythona. Ciekawostką jest to, że nazwa *Python* kojarząca się z wężem, pochodzi z angielskiego serialu komediowego *Latający cyrk Monty Pythona*, którego twórca języka był ogromnym fanem.

Python w roku 2016 przynosi wersję 3.5. Język wspiera obiektowy, imperatywny, oraz funkcyjny paradygmat programowania. Oznacza to, że w Pythonie możemy programować proceduralnie, czyli pisząc ciąg instrukcji do wykonania przez procesor, lub obiektowo, tworząc obiekty składające się na całą aplikację, które komunikują się ze sobą za pomocą metod. Python jest często używany jako język skryptowy przystosowany do pracy na różnych systemach operacyjnych. Dostępny jest również tryb interaktywny, przydatny szczególnie do krótkich obliczeń i uzyskiwania pomocy.

Cechą charakterystyczną języka Python jest „brak wąsów” (nawiasów klamrowych {}) przy instrukcjach złożonych, kojarzących nam się z większością języków programowania (C/C++, Java, Pike ...), które tutaj zostały zastąpione poprzez wcięcia. Drugą charakterystyczną cechą jest brak średników kończących instrukcje (średniki separują instrukcje w wierszu). Listing 2.1 ukazuję typowy program hello wyświetlający napis na ekranie.

Listing 2.1. Skrypt hello.

```
#!/usr/bin/python
# To jest komentarz.

# Definicja funkcji.
def hello_world():
    a = "Hello"
    b = "World!"
    return a + " " + b

print hello_world()
# Wypisanie na ekranie napisu "Hello World!"
```



12:1959083492

## 2.1. Typy, arytmetyka i struktury danych

Python posiada dynamiczny system typów co oznacza, że wartości są przypisywane do zmiennych w trakcie działania programu. Nie ma deklaracji typów zmiennych, bo w czasie działania programu zmienna może przechowywać wartości różnych typów. Dużym udogodnieniem dla programisty jest automatyczne odśmiecanie pamięci (ang. *garbage collection*).

### 2.1.1. Typy danych

Typy danych w języku Python znamy już z innych języków programowania. Podstawowe typy liczbowe to liczby całkowite, zmiennoprzecinkowe, zespolone. Nowością są łańcuchy znakowe, ograniczane parą pojedyńczych lub podwójnych apostrofów. Dzięki temu można wygodnie zagnieżdżać stringi.

Część typów to tak zwane kolekcje (listy, krotki, słowniki, zbiory) zapewniające szybkie przeszukiwanie danych. Listing 2.2 przedstawia przykłady typów danych.

Listing 2.2. Typy danych w języku Python.

```
nothing = None         # nic odpowiednik null
integer = 7            # liczba calkowita
real = 7.5             # liczba zmiennoprzecinkowa
complex = 3 + 4j       # liczba zespolona
logical = True         # typ logiczny
string = 'hello'       # napis
string2 = "witaj"      # inny napis
alist = ["one", "two", "Truck", "Milion"]              # lista
aset = set(["Orange", "Yellow", "Green"])             # zbior zmienny
afrozenset = frozenset({4.0, 'string', True})         # zbior niezmienny
atuple = ("1", "Mass", "Beryl", "continental")        # krotka
adict = {"name":"Jacob", "age":23}                    # slownik
```

### 2.1.2. Operacje arytmetyczne i bitowe

Przy operacjach arytmetycznych należy wspomnieć, że w Pythonie wszystko jest obiektem. Możemy dziedziczyć i to wielokrotnie praktycznie z każdej klasy. Python posiada umiarkowaną kontrolę typowania. Oznacza to, że nie jest bardzo restrykcyjny, ale też nie pozwala na rzutowanie wszystkiego na wszystko. Python pozwoli nam na mnożenie liczby zespolonej przez liczbę całkowitą, ale już nie pozwoli dodać nam tekstu do liczby. Jeśli zdarzy się przypadek, że do operacji zostanie użyty nie właściwy typ, zostanie rzucony wyjątek TypeError.

Listing 2.3 ukazuje nam podstawowe operacje arytmetyczne, a listing 2.4 podstawowe operacje bitowe.

Listing 2.3. Operacje arytmetyczne w języku Python.

```
# Operacje arytmetyczne dwuargumentowe.

2 + 4      # dodawanie
8 - 17     # odejmowanie
2 * 2      # mnozenie
```



13:7841891343

```
8 / 2       # dzielenie
64 // 8     # dzielenie calkowite
6 % 4       # modulo (reszta z dzielenia)
2 ** 4      # potegowanie (dwa do potegi czwartej)

# Operacje arytmetyczne jednoargumentowe.

z = 4
i = 10
x = 1
-z          # minus, wartosc przeciwna
+i          # plus, zapewnia niezmienna wartosc przekazanego argumentu
~x          # inwersja bitowa
```

Listing 2.4. Operacje bitowe w języku Python.

```
3 & 5       # koniunkcja
2 | 4       # alternatywa
4 ^ 3       # alternatywa wykluczajaca
2 << 3      # przesuniecie bitowe w lewo
8 >> 7      # przesuniecie bitowe w prawo
~4          # negacja
```

### 2.1.3. Operatory logiczne

Podstawowe operatory logiczne w języku Python to **not** (zaprzeczenie), **and** (koniunkcja), oraz **or** (alternatywa). Przykłady ich użycia oraz pozostałe operatory ukazuje listing 2.5

Listing 2.5. Podstawowe operatory logiczne.

```
not (5 > 1)            # False
4 < 10 and 1 <= 1      # True
10 != 10 or 9 == 9     # True

a == b      # jest rowne
a != b      # nie rowne
a < b       # mniejsze od
a <= b      # mniejsze lub rowne
a > b       # wieksze od
a >= b      # wieksze lub rowne
```

## 2.2. Instrukcje sterujące programem

Instrukcje sterujące to mechanizmy pozwalające na nieliniowe przetwarzanie programu. W zależności od rodzaju mechanizmu sterującego i sprawdzanych warunków program może zmienić kolejność przetwarzanych instrukcji.

### 2.2.1. Instrukcja warunkowa

Instrukcja warunkowa to najbardziej popularny mechanizm sterujący. Polega on na wyborze bloku **if** (warunek jest prawdziwy) lub bloku **else** (warunek jest fałszywy). Wzbogaceniem jest tutaj jeden lub więcej bloków **elif**,



14:2310317983

które pozwalają na ponowne sprawdzenie prawdziwości warunku, gdy został odrzucony przez poprzedzający go blok **if** lub **elif**. Listing 2.6 przedstawia przykład zastosowania instrukcji warunkowej. Zauważmy, że wiersze nagłówkowe kończą się dwukropkiem, a instrukcje w bloku są wcięte o stałą szerokość (zwykle cztery spacje).

Listing 2.6. Instrukcja warunkowa.

```
x = 10
y = 8

if 10 < y:
    print "10 jest mniejsze od", y
elif y == 10:
    print y, "jest rowne 10"
elif y < 10:
    print y, "jest mniejsze od 10"
else:
    print "blad"
# Wynik: napis "8 jest mniejsze od 10"

if x != y:
    print "liczby rozne"
else:
    print "liczby rowne"
# Wynik: napis "liczby rozne"
```

### 2.2.2. Pętla for

Pętla **for** to mechanizm pozwalający na powtarzanie instrukcji zawartych w bloku aż do wyczerpania sekwencji obiektów. Listing 2.7 ukazuje pętlę **for** w działaniu.

Listing 2.7. Pętla **for**.

```
# Petla po znakach z napisu.
word = 'Master'
for letter in word:
    print 'Now loop state :', letter
# Wynik:
#Now loop state : M
#Now loop state : a
#Now loop state : s
#Now loop state : t
#Now loop state : e
#Now loop state : r

# Petla po elementach listy.
articles = ['chleb', 'mleko', 'kalafior']
for item in articles:
    print 'Now i eat : ', item

# Wynik:
#Now i eat : chleb
#Now i eat : mleko
#Now i eat : kalafior
```



15:6504542736

```
# Petla po zakresie liczb od 5 do 10.
for i in range(5, 11):
    print i
# Wynik: 5 6 7 8 9 10
```

Przy podawaniu zakresu należy pamiętać, że liczba odpowiadająca za element maksymalny musi być o jeden wyższa niż zakładana przez programistę, aby spełniony był warunek. Liczba 10 jest mniejsze od 11, dlatego 10 zostanie wyświetlona, ale 11 już nie jest mniejsze od 11. Dlatego w tym momencie pętla kończy swój bieg.

### 2.2.3. Pętla while

Pętla **while** przetwarza instrukcje w bloku dopóki warunek jest prawdziwy. Jeżeli warunek od początku będzie fałszywy, to instrukcje z bloku nie będą ani razu wykonane. Listing 2.8 ukazuje działanie pętli **while**.

Listing 2.8. Pętla **while**.

```
i = 0
f = 1
while i < 5:
    i += 1
    f *= i
print f                # factorial 5!
```

Znawcy innych języków programowania nie znajdą w Pythonie pętli typu do-while. Można ją jednak zastąpić instrukcjami zawierającymi pętlę **while**.

## 2.3. Funkcje

Funkcje to pewien blok kodu posiadający swoją nazwę, oraz pozwalający na wywoływanie go z pewnych ograniczonych miejsc w programie. Zazwyczaj celem użycia funkcji jest uniknięcie powielania kodu oraz ułatwienie w poprawianiu błędów (jedno miejsce do sprawdzenia). Funkcję wywołujemy poprzez jej nazwę, z argumentami lub bez. Python posiada funkcje anonimowe, funkcje wbudowane, oraz funkcje stworzone przez użytkownika. Funkcje deklarujemy poprzez słowo kluczowe **def**. Funkcja zwraca wartość poprzez **return**. Python również obsługuje funkcje o nieokreślonej liczbie argumentów. Listning 2.9 ukazuje działanie przykładowych funkcji, w tym przypadku zdefiniowanych przez programistę.

Listing 2.9. Funkcje w Pythonie.

```
# Funkcja bez argumentow.
def function_one():
    print "function one is active ..."

function_one()

# Zalazek aplikacji bazujacej na slowniku polsko-angielskim.
dict_pl_en = {}
```



16:3924373306

```
def modify_word(x, y):     # funkcja dwuargumentowa
    dict_pl_en[x] = y

def remove_word(x):    # funkcja jednoargumentowa
    del dict_pl_en[x]

def return_word(x):    # przyklad funkcji zwracajacej wartosc
    return dict_pl_en[x]

modify_word("kot", "cat")
modify_word("ciac", "cut")
modify_word("bialy", "black")
remove_word("bialy")

print len(dict_pl_en)           # 2
print return_word("kot")        # cat
```

## 2.4. Wyrażenia lambda i generatory

Wyrażenia lambda to zapożyczone z języka Lisp funkcje anonimowe, mogące zawierać tylko proste wyrażenia. Najczęściej są one jednolinijkowe. Listing 2.10 ukazuje sposób użycia wyrażenia lambda.

Listing 2.10. Wyrażenia lambda.

```
a = lambda x: x * 2 + 4
b = lambda y: y + 4

print a(2)          # 8
print b(3)          # 7
```

Początkowo generatory w Pythonie były dostępne po wywołaniu instrukcji **from** __future__ **import** generators. Od Pythona 2.3 generatory są już dostępne domyślnie bez żadnych importów. Generator to funkcja, która zapamiętuje swój wewnętrzny stan. W przypadku zwykłej funkcji jej pola są niszczone przy wyjściu z niej, natomiast po wyjściu z generatora jej pola istnieją nadal. Można więc dowolnie generator zatrzymywać i go wznawiać. Oszczędzamy przy tym inicjalizację dużych obiektów. Generator używa **yield** zamiast **return**, które w wersji 2.3 weszło do zbióru słów kluczowych języka Python. Listing 2.11 ukazuje przykład generatora oraz jego wywołanie.

Listing 2.11. Przykład generatora.

```
# Generator zwraca kolejne liczby podzielne przez 3 lub 5.
def get_int35():
    i = 0
    while True:
        if (i % 3 == 0) or (i % 5 == 0):
            yield i
        i += 1

for x in get_int35():
    if x > 10:
```



17:7504002243

```
        break
    print x
# Wynik: 0 3 5 6 9 10
```

## 2.5. Moduły i pakiety

Moduły w Pythonie to najczęściej pliki z rozszerzeniem .py, zawierające instrukcje w języku Python, np. definicje zmiennych, funkcji, klas, do ponownego wykorzystania. Są bardzo przydatne przy tworzeniu rozbudowanych aplikacji. Aby załadować moduł wystarczy wydać polecenie **import** nazwa_modulu. Często mamy do czynienia z sytuacją, w której jeden moduł importuje kilka następnych. Każdy moduł jest importowany tylko raz.

Pakiety to przestrzenie nazw zawierające w sobie moduły, niekiedy nawet znów pakiety. Każdy pakiet w Pythonie jest folderem zawierającym plik inicjalizujący __init__.py i inne pliki lub foldery. Plik inicjalizujący może być pusty. Zatem jeśli chcemy stworzyć pakiet mathematica z modułem algebra, to tworzymy folder mathematica, a w nim plik algebra.py. Import modułu będzie korzystał z notacji z kropką: **import** mathematica.algebra.

## 2.6. Klasy

W XXI wieku metodologia programowania obiektowego jest już rozpowszechniona. Większość rozbudowanych aplikacji jest pisanych z wykorzystaniem klas. Jest mnóstwo powodów, aby tak robić, ale najważniejszym z nich to koszt. Programy obiektowe są po prostu tańsze w utrzymaniu i łatwiej się je serwisuje.

Klasy to podstawowe narzędzia do tworzenia własnych struktur danych. Instancje klasy komunikują się ze światem za pomocą metod. Klasy i instancje zawierają pola (dane, atrybuty). W Pythonie nie ma enkapsulacji danych tzn. wszystkie atrybuty są dostępne publicznie. Klasę w Pythonie tworzymy poprzez słowo kluczowe **class**. Listning 2.12 ukazuje przykład definicji klasy wraz z dziedziczeniem.

Listing 2.12. Przykłady klas wraz z dziedziczeniem.

```
class Car:

    def __init__(self):
        pass

    def get_name(self):
        raise NotImplementedError("virtual method")

    def get_color(self):
        raise NotImplementedError("virtual method")

class FamilyCar(Car):    # dziedziczenie z klasy Car

    def __init__(self, name, color):
        self.name = name
```



18:8802290729

```
        self.color = color

    def get_name(self):
        return self.name

    def get_color(self):
        return self.color

my_car = FamilyCar("Sniezynka", "Bialy")
print my_car.getName()          # Sniezynka
print my_car.getColor()         # Bialy
```

## 2.7. Wyjątki

W Pythonie wyjątki to po prostu klasy. Służą do raportowania błędów lub sytuacji wyjątkowych. Wyjątki mogą być przechwytywane i obsługiwane w żądany przez nas sposób. Listing 2.13 ukazuje wyjątek typu ZeroDivisionError (dzielenie przez zero), oraz własną klasę obsługującą napotkany wyjątek.

Listing 2.13. Wyjątek ZeroDivisionError.

```
print 25 / 0          # sprowokowanie wyjatku
# Przykladowe komunikaty:
#Traceback (most recent call last):
#  File "introduction_to_python.py", line 74, in <module>
#    25 / 0
#ZeroDivisionError: integer division or modulo by zero

# Wlasna klasa obslugujaca wyjatek.
class MyError(ZeroDivisionError):

    def __init__(self, message):
        self.message = message

    def __str__(self):
        return "Exception: " + self.message

raise MyError("dzielenie przez zero")    # rzucenie wyjatku
```

## 2.8. Biblioteka standardowa

Język Python posiada obszerną bibliotekę standardową. Jest to zestaw pakietów do wielu różnych zastosowań. W Internecie są dostępne dodatkowe pakiety stworzone przez firmy lub pojedyńczych programistów, które można użyć jako podstawę do dalszego rozwoju aplikacji dopasowanej do indywidualnych potrzeb.

Przykłady pakietów z biblioteki standardowej:
- math, zbiór funkcji matematycznych,
- fractions, obsługa liczb wymiernych,
- random, m.in. generowanie liczb pseudolosowych,
- pickle, serializacja obiektów,



19:7811935032

— copy, kopiowanie dowolnych obiektów,
— re, obsługa wyrażeń regularnych,
— timeit, narządzia do pomiaru czasu wykonywania kodu,
— unittest, narzędzia do testowania kodu.

# 3. Matematyka wielomianów

W tym rozdziale przypominamy elementarne definicje dotyczące głównie wielomianów jednej zmiennej.

## 3.1. Podstawowe definicje

Wielomianem jednej zmiennej stopnia $n$ nazywamy wyrażenie postaci [2]

$$f(x) = \sum_{k=0}^{n} a_k x^k \quad \text{(postać kanoniczna)}, \tag{3.1}$$

gdzie $a_k$ są stałymi współczynnikami, a $x$ jest zmienną niezależną. Współczynniki $a_k$ mogą być dowolnymi liczbami (całkowite, wymierne, rzeczywiste, zespolone). $f(x)$ może być jednym wyrazem lub sumą skończonej liczby wyrazów $a_k x^k$.

Wielomian wielu zmiennych jest skończoną sumą wyrazów, a każdy wyraz jest iloczynem stałego współczynnika i zmiennych niezależnych. Przykładowy wielomian dwóch zmiennych ma postać

$$f(x, y) = 3x^2 - 4xy + 5y^2 + 6x - 7y + 8, \tag{3.2}$$

gdzie zmienne niezależne są oznaczone jako $x$ i $y$.

*Stopniem zmiennej* w wyrazie nazywa się wykładnik przy danej zmiennej. *Stopniem (niezerowego) wyrazu* nazywamy sumę stopni wszystkich zmiennych tego wyrazu. *Stopniem (niezerowego) wielomianu* $f$ nazywa się największy stopień wyrazu [oznaczenie $\deg(f)$]. *Wyraz wiodący* wielomianu to wyraz o największym stopniu, przy czym należy doprecyzować jak sortowane będą wyrazy o jednakowym stopniu. *Jednomian*, *dwumian* i *trójmian* to nazwy wielomianów składających się odpowiednio z jednego, dwóch i trzech wyrazów.

Wielomiany mogą być tworzone ze stałych i zmiennych wyłącznie za pomocą dwóch działań: dodawanie i mnożenia, ponieważ odejmowanie jest równoważne dodawaniu liczby przeciwnej, a potęgowanie to wielokrotne mnożenie. Własności działań na wielomianach:

— Suma i iloczyn wielomianów jest wielomianem.
— Pochodna wielomianu jest wielomianem.
— Całka (funkcja pierwotna) wielomianu jest wielomianem.
— Złożenie wielomianów jest wielomianem.

**Dzielenie wielomianów jednej zmiennej:** Każdy wielomian $f$ jednej zmiennej można przedstawić w postaci $f = gh + r$, gdzie $g$, $h$, $r$ są wielomianami,



21:8606356081

$\deg(r) < \deg(g)$, $h$ i $r$ są wyznaczone jednoznacznie. Jeżeli $r = 0$, to mówimy, że $f$ jest podzielny przez $g$. Do wyznaczenia $h$ i $r$ stosuje się *algorytm Euklidesa* [6].



22:1036474563

# 4. Bazy Gröbnera

Bazy standardowe i dowód ich istnienia pojawił się w pracy Hironaki (1964) o usuwaniu osobliwości [7]. Algorytmy umożliwiające wyznaczanie bazy podał Buchberger (1965) w pracy doktorskiej, której promotorem był Gröbner [5]. Teoria baz Gröbnera została uogólniona w wielu kierunkach i znaleziono jej różne zastosowania. Pierwsza w Polsce książka Dumnickiego i Winiarskiego o bazach Gröbnera ukazała się w roku 2007 (drugie wydanie w roku 2009) [8]. Z tej książki pochodzi większość definicji i twierdzeń tego rozdziału. Literatura w języku angielskim jest bardzo bogata, szczególnie przydatny dla nas okazał się raport z pseudokodami algorytmów baz Gröbnera [9].

## 4.1. Porządek dopuszczalny

W zbiorze liczb naturalnych $N$ (z zerem) istnieje naturalne uporządkowanie, które przenosi się do zbioru jednomianów jednej zmiennej $T$ ($1 < x < x^2 < \ldots$). Mnożenie jednomianów odpowiada dodawaniu w zbiorze $N$, przy czym jeżeli $x^j < x^k$, to $x^j x^s < x^k x^s$.

W przypadku $n$ zmiennych możemy rozważyć zbiór $N^n$ z elementami $\alpha = (\alpha_1, \ldots, \alpha_n)$, oraz powiązany z nim zbiór jednomianów $T^n$ z elementami $x^\alpha = x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$. Stopniem jednomianu $x^\alpha$ nazywamy liczbę

$$\deg(x^\alpha) = |\alpha| = \alpha_1 + \ldots + \alpha_n. \tag{4.1}$$

**Definicja:** Porządek $\leqslant$ jest *dopuszczalny* (ang. *admissible order*) w $N^n$, jeżeli dla dowolnych $\alpha, \beta, \gamma \in N^n$
(i) $\alpha \leqslant \beta$ lub $\beta \leqslant \alpha$,
(ii) $0 \leqslant \alpha$,
(iii) $\alpha \leqslant \beta \Rightarrow \alpha + \gamma \leqslant \beta + \gamma$.

W zbiorze $T$ istnieje tylko jeden porządek dopuszczalny, opisany wcześniej. Dla wielu zmiennych istnieje kilka możliwych porządków dopuszczalnych. Podamy przykłady dla trzech zmiennych $z < y < x$.

— Porządek leksykograficzny (ang. *(pure) lexicographic order*, *lex*, *plex*), np.
  $x^2y^5z^3 <_{lex} x^3y^2z$ ($x^2 < x^3$),
  $x^2y^3z^2 <_{lex} x^2y^5z$ ($x^2 = x^2$, $y^3 < y^5$).
— Porządek stopniowo leksykograficzny (ang. *graded lex order*, *grlex*, *deglex*), np.
  $x^3y^2z <_{deglex} x^2y^4z$ ($3+2+1 < 2+4+1$),
  $x^2y^2z^2 <_{deglex} x^3yz^2$ ($2+2+2 = 3+1+2 = 6$, $x^2 < x^3$).
— Porządek stopniowo leksykograficzny odwrotny (ang. *graded reverse lex order*, *grevlex*, *degrevlex*), np.



23:5920875943

$x^3y^2z <_{degrevlex} x^2y^3z^2$ $(3+2+1 < 2+3+2)$,
$xy^2z^2 <_{degrevlex} xy^3z$ $(1+2+2 = 1+3+1 = 5,\ z^2 > z)$.

**Twierdzenie:** W dowolnym niepustym podzbiorze $T^n$ istnieje jednomian najmniejszy [8]. Stąd wynika, że jeżeli jakiś algorytm będzie generował silnie malejący ciąg jednomianów, to ten algorytm musi kiedyś skończyć działanie (*własność stopu*).

## 4.2. Pierścień wielomianów

**Definicja:** Wielomianem $n$ zmiennych o współczynnikach z ciała $K$ nazywamy *skończoną* sumę postaci [8]

$$f(x) = \sum_{\alpha} c_\alpha x^\alpha,\ c_\alpha \in K,\ x^\alpha \in T^n. \tag{4.2}$$

**Definicja:** Zbiór wielomianów $n$ zmiennych o współczynnikach z ciała $K$, z naturalnymi działaniami dodawania i mnożenia, tworzy *pierścień wielomianów* $K[x]$.

**Definicja:** Dla ustalonego porządku dopuszczalnego i niezerowego wielomianu $f$ definiujemy
— Nośnik $f$, $\text{supp}(f) = \{x^\alpha \in T^n : c_\alpha \neq 0\}$,
— Jednomian wiodący $f$ (ang. *leading monomial*),
$LM(f) = \max\{\text{supp}(f)\}$,
— Współczynnik wiodący $f$ (ang. *leading coefficient*),
$LC(f) = c_\alpha$ dla $x^\alpha = LM(f)$,
— Wyraz wiodący $f$ (ang. *leading term*),
$LT(f) = LC(f)LM(f)$,
— Ogon $f$, $\text{tail}(f) = f - LT(f)$.

**Definicja:** *Monoid generowany przez* $A \subset T^n$ jest to zbiór

$$A \cdot T^n = \{am : a \in A,\ m \in T^n\}. \tag{4.3}$$

*Przestrzeń liniowa generowana przez* $A$ jest to zbiór

$$\text{Span}(A) = \left\{f \in K[x] : f = \sum c_\alpha x^\alpha,\ x^\alpha \in A\right\}. \tag{4.4}$$

Jeżeli $A \cdot T^n = A$, to $A$ nazywamy *monoidem.*

**Stwierdzenie:** Jeżeli $f \in K[x]$, to $f \in \text{Span}(\text{supp}(f))$.

**Lemat Dickinsona:** Dla dowolnego zbioru $A \subset T^n$ istnieje *skończony* zbiór $B \subset T^n$ taki, że $A \cdot T^n = B \cdot T^n$ (równe monoidy generowane).

**Definicja:** Dla ustalonego porządku dopuszczalnego i niepustego zbioru wielomianów $F \subset K[x]\backslash\{0\}$ definiujemy



24:4240300262

— Zbiór jednomianów wiodących $LM(F) = \{LM(f) : f \in F\}$,
— Monoid wiodący $\mathcal{L}(F) = LM(F) \cdot T^n$,
— Jednomiany standardowe $\mathcal{D}(F) = T^n \backslash \mathcal{L}(F)$.

**Definicja:** Niech $F = \{f_1, \ldots, f_s\}$ będzie skończonym zbiorem wielomianów z $K[x]$. Definiujemy *ideał generowany przez F* jako

$$\langle F \rangle = \left\{ \sum_{i=1}^{s} h_i f_i : h_i \in K[x] \right\}. \tag{4.5}$$

Zbiór $F$ nazywamy bazą ideału $\langle F \rangle$ i jest to baza skończona, więc ideał $\langle F \rangle$ nazywamy *skończenie generowanym*. Dla nieskończonego zbioru $F \subset K[x]$ ideał $\langle F \rangle$ definiuje się podobnie, ale tworzymy jedynie skończone kombinacje wielomianów. Jeżeli $F = \langle F \rangle$, to $F$ nazywamy *ideałem*.

Zauważmy, że baza ideału $\langle F \rangle$ nie jest wyznaczona jednoznacznie. Wśród różnych baz szczególnie użyteczna jest baza Gröbnera.

**Przykład:** Niech $F = \{x, y^2\}$ zawiera się w $K[x, y]$ [8]. Wtedy do ideału $\langle F \rangle$ należą tylko wielomiany, dla których $c_{(0,0)} = c_{(0,1)} = 0$.

**Stwierdzenie:** Dla niepustego zbioru $A \subset T^n$ zachodzi
(1) $A \cdot T^n \subset \langle A \rangle$ (monoid zawiera się w ideale),
(2) $\langle A \rangle = \mathrm{Span}(A \cdot T^n)$.

**Twierdzenie Hilberta o bazie:** Każdy ideał w pierścieniu wielomianów $K[x]$ nad pierścieniem noetherowskim $K$ jest skończenie generowany. Pierścieniem noetherowskim (ang. *Noetherian ring*) jest każde ciało oraz np. pierścień liczb całkowitych. W naszych zastosowaniach pierścień $K$ będzie odpowiadał liczbom całkowitym, wymiernym, rzeczywistym lub zespolonym. Twierdzenie Hilberta o bazie można wysłowić następująco: jeżeli pierścień $K$ jest noetherowski, to jego pierścień wielomianów $K[x]$ również jest noetherowski.

## 4.3. Baza i zredukowana baza Gröbnera

**Definicja:** Dla ustalonego porządku dopuszczalnego i skończonego podzbioru $G \subset K[x]$, $G$ nazywamy *bazą Gröbnera*, jeżeli $0 \notin G$ oraz $\mathcal{L}(\langle G \rangle) = \mathcal{L}(G)$.

**Definicja:** Niezerowy wielomian $f$ z $K[x]$ jest *unitarny* względem danego porządku dopuszczalnego, jeżeli $LC(f) = 1$. Zbiór $F \subset K[x]$ jest unitarny, jeżeli każdy wielomian z $F$ jest unitarny.

**Definicja:** Bazę Gröbnera $G$ względem ustalonego porządku dopuszczalnego nazywamy *zredukowaną*, jeżeli
(1) $G$ jest minimalną bazą Gröbnera (najmniejsza liczność),
(2) $G$ jest unitarna,
(3) $\mathrm{supp}(\mathrm{tail}(g)) \subset \mathcal{D}(G)$ dla wszystkich $g \in G$.



25:4262908244

## 4.4. Redukcja wielomianowa

Redukcja wielomianowa jest istotną częścią algorytmu baz Gröbnera, a zarazem częścią najbardziej kosztowną obliczeniowo. Dla wielomianu jednej zmiennej $f$, redukcja za pomocą wielomianu $g$ odpowiada szukaniu reszty $r$ w algorytmie Euklidesa, gdzie $f = gh + r$. Dla wielomianu wielu zmiennych $f$, redukuje się go za pomocą zbioru wielomianów $F$, aby wyznaczyć resztę $r$. Realizuje to algorytm dzielenia uogólnionego.

**Definicja:** Niech $f, g$ należą do $K[x]$. Jeżeli $LM(g)$ dzieli $LM(f)$, to *redukcję* $f$ do $r$ za pomocą $g$ określamy następująco

$$r = f - \frac{LT(f)}{LT(g)}g. \tag{4.6}$$

Czasem określa się operację redukcji nie tylko dla $LT(f)$, ale dla dowolnego wyrazu wielomianu $f$, który jest podzielny przez $LM(g)$.

Algorytm dzielenia uogólnionego wykonuje wielokrotnie redukcję $f$ za pomocą wielomianów ze zbioru $F$. Jako wynik końcowy otrzymuje się resztę $r$ całkowicie zredukowaną ze względu na $F$.

## 4.5. S-wielomiany

**Definicja:** Dla ustalonego porządku dopuszczalnego i niezerowych wielomianów $f$, $g$ należących do $K[x]$, określamy najmniejszą wspólną wielokrotność jednomianów

$$J = \mathrm{lcm}(LM(f), LM(g)). \tag{4.7}$$

Wtedy wielomian

$$S(f,g) = \frac{J}{LT(f)}f - \frac{J}{LT(g)}g \tag{4.8}$$

nazywamy *S-wielomianem* wielomianów $f$ i $g$. Zauważmy, że wyrazy wiodące $f$ i $g$ kasują się. Niektórzy autorzy określają S-wielomiany z innym mnożnikiem

$$\tilde{S}(f,g) = LC(f)LC(g)S(f,g) = LC(g)\frac{J}{LM(f)}f - LC(f)\frac{J}{LM(g)}g. \tag{4.9}$$

**Stwierdzenie:** $LM(S(f,g)) \leqslant LM(f)LM(g)$.

## 4.6. Kryterium Buchbergera

**Twierdzenie:** Niech $G = \{g_1, \ldots, g_s\}$ będzie zbiorem niezerowych wielomianów z $K[x]$. Wtedy następujące warunki są równoważne:

— $G$ jest bazą Gröbnera.



26:1036167091

— Dla dowolnych $1 \leqslant j < k \leqslant s$ wielomian $S(g_j, g_k)$ redukuje się do zera ze względu na $G$.
— Dla każdego wielomianu $f$ z ideału $\langle G \rangle$, redukcja $f$ ze względu na $G$ daje zero.

Na tym twierdzeniu opiera się algorytm Buchbergera wyznaczania bazy Gröbnera.

W celu wyznaczenia zredukowanej bazy Gröbnera trzeba usunąć pewną nadmiarowość z posiadanej bazy Gröbnera $G$. Po pierwsze, każdy wielomian w bazie musi być unitarny, czyli wystarczy podzielić go przez współczynnik wiodący. Po drugie, każdy element $g$ bazy $G$ należy zredukować używając zbioru $G \backslash \{g\}$. Jeżeli w wyniku redukcji otrzymamy zero, to dany element $g$ można usunąć z bazy.

## 4.7. Zastosowania

Przedstawimy listę wybranych zastosowań teorii baz Gröbnera. We wszystkich zastosowaniach udało się sformułować dany problem jako pytanie dotyczące zbioru wielomianów wielu zmiennych. Po wykonaniu obliczeń baz Gröbnera i wykorzystaniu kluczowych własności tych baz udało się otrzymać rozwiązanie problemu.

— Kolorowanie wierzchołków grafu (problem trzech kolorów [8]).
— Dowodzenie twierdzeń geometrycznych (wzór Herona [8], twierdzenie Apoloniusza [10]).
— Obliczanie całek funkcji specjalnych [10].
— Rozwiązywanie układów liniowych równań diofantycznych [8].
— Rozwiązywanie zagadek logicznych, np. Mastermind [8].
— Rozwiązywanie pewnych problemów optymalizacyjnych całkowitoliczbowych, np. wydawanie monet [10].
— Zastosowanie w teorii kodowania sygnałów do korekty błędów transmisji [10].
— Zastosowanie w robotyce do znajdowania położeń ramion robota [10].
— Zastosowanie w inżynierii oprogramowania do znajdowania niezmienników pętli, przy automatycznej weryfikacji poprawności programów [10].
— Optymalizacja wydobycia ropy naftowej z platform wiertniczych [10].

27:7491765701

# 5. Wielomiany jednej zmiennej

W tym rozdziale przedstawiono interfejs i implementacje wielomianów jednej zmiennej (klasa Poly).

## 5.1. Interfejs wielomianów

Interfejs wielomianów jednej zmiennej został przedstawiony w tabeli 5.1. Przygotowano dwie implementacje wielomianów jednej zmiennej, które realizują podany interfejs.

Pierwsza implementacja jest oparta na wewnętrznej liście. Wszystkie współczynniki są tam obecne, przez co ta implementacja jest zalecana przy wielomianach z dużą liczbą niezerowych współczynników. Pomysł jest zaczerpnięty z książki Sedgewicka [11].

Druga implementacja jest oparta na wewnętrznym słowniku, w którym kluczami są potęgi niezerowych wyrazów, a wartościami są niezerowe współczynniki. Ta implementacja zajmuje mniej pamięci, jeżeli pracujemy z wielomianiami o wysokich potęgach, a małą liczbą wyrazów. W obu implementacjach wzięto pod uwagę następujące zagadnienia.

— Wielomiany przede wszystkim są dostosowane do obliczeń dokładnych na liczbach całkowitych (int, long) i ułamkach (Fraction), ale pracują również z liczbami float i complex. Działania na wielomianach o współczynnikach całkowitych lub ułamkowych nie generują liczb float lub complex.
— Ułamki z mianownikiem równym jeden są zamieniane na liczby całkowite w metodzie __repr__, aby poprawić czytelność wyświetlania wyników.
— Zerowe współczynniki przy najwyższych potęgach zmiennej niezależnej, które mogą pojawić się podczas wykonywania działań na wielomianach, są usuwane. Wykonuje to metoda prywatna _cancel(), która jest wywoływana po wykonaniu dodawania, odejmowania i mnożenia. W implementacji słownikowej w ogóle nie przechowuje się zerowych współczynników.
— Konstruktor klasy Poly tworzy jednomian z zadanym współczynnikiem przy podanej potędze zmiennej niezależnej. Metoda klasy Poly.from_list tworzy wielomian z podanej listy kolejnych współczynników.
— Obliczanie wartości wielomianu dla danego argumentu jest wykonywane algorytmem Hornera.
— Potęgowanie wielomianu jest wykonywane metodą potęgowania binarnego.
— Wielomiany są niezmienne (ang. *immutable*), działania zawsze tworzą nowy obiekt wielomianu (z wyjątkiem operatora jednoargumentowego +p).



28:1052364723

— Pewne operacje są zdefiniowane tylko dla jednomianów: dzielenie jednomianów, obliczanie najmniejszej wspólnej wielokrotności jednomianów (wynik ma zawsze współczynnik wiodący równy jeden).
— Obliczanie wyrazu wiodącego, jednomianu wiodącego i współczynnika wiodącego jest możliwe tylko dla niezerowego wielomianu. Wymagane jest podanie metody określającej porządek dopuszczalny, np. key=Poly.key_lex. Dla wielomianów jednej zmiennej istnieje tylko jeden porządek dopuszczalny.
— Sortowanie listy wielomianów jednej zmiennej można wykonać za pomocą metody list.sort, przy czym należy podać argument key=Poly.key_deg. Odpowiada to jedynemu istniejącemu porządkowi dopuszczalnemu. Zdefiniowano również inne metody, które odpowiadają pewnym porządkom dopuszczalnym. Są to Poly.key_lex (porządek leksykograficzny) i Poly.key_deglex (porządek stopniowo leksykograficzny). Dla wielomianów jednej zmiennej te metody są aliasami do metody Poly.key_deg, ale dla wielomianów wielu zmiennych są to istotnie różne porządki.



29:9344831703

Tabela 5.1. Interfejs wielomianów jednej zmiennej. $p$ i $q$ są wielomianami, $n$ jest liczbą całkowitą nieujemną, $a, b, c$ to liczby (int, long, float, complex) lub ułamki (Fraction).

| **Operacja** | **Znaczenie** | **Metoda** |
|---|---|---|
| p = Poly(c, n) | tworzenie jednomianu | __init__ |
| q = Poly(c) | tworzenie jednomianu | __init__ |
| Poly() | wielomian zerowy | __init__ |
| Poly.from_list(L) | tworzenie wielomianu z listy | from_list |
| **print** q | wyświetlanie wielomianu | __repr__ |
| p + q, p + a | dodawanie wielomianów | __add__ |
| a + p | dodawanie wielomianów | __radd__ |
| p − q, p − a | odejmowanie wielomianów | __sub__ |
| a − p | odejmowanie wielomianów | __rsub__ |
| p * q, p * a | mnożenie wielomianów | __mul__ |
| a * p | mnożenie wielomianów | __rmul__ |
| p / a | dzielenie przez liczbę | __div__ |
| p / q | dzielenie jednomianów | __div__ |
| +p | operator jednoargumentowy | __pos__ |
| −p | operator jednoargumentowy | __neg__ |
| p.is_zero() | czy wielomian jest zerowy | is_zero |
| p.degree() | stopień wielomianu | degree |
| len(p) | liczba niezerowych wyrazów | __len__ |
| p.key_deg() | porządek stopniowy | key_deg |
| p.key_lex() | porządek leksykograficzny | key_lex |
| p.key_deglex() | porządek stopniowo leksykograficzny | key_deglex |
| p[n] | $n$-ty współczynnik wielomianu | __getitem__ |
| p == q | porównywanie wielomianów | __eq__ |
| p != q | porównywanie wielomianów | __ne__ |
| p.eval(a) | wartość wielomianu | eval |
| p(a) | wartość wielomianu | __call__ |
| p.combine(q) | składanie wielomianów | combine |
| p(q) | składanie wielomianów | __call__ |
| p ** n, pow(p, n) | potęgowanie wielomianu | __pow__ |
| p.leading_term(key) | wyraz wiodący | leading_term |
| p.leading_monomial(key) | jednomian wiodący | leading_monomial |
| p.leading_coefficient(key) | współczynnik wiodący | leading_coefficient |
| p.iterterms() | generator wyrazów wielomianu | iterterms |
| p.diff() | różniczkowanie wielomianu | diff |
| p.integrate() | całkowanie wielomianu | integrate |
| p.lcm(q) | najmniejsza wspólna wielokrotność | lcm |



30:1151007186

Przykładowa sesja interaktywna prezentuje możliwości wielomianów jednej zmiennej (listing 5.1).

Listing 5.1. Sesja interaktywna z wielomianami jednej zmiennej.

```
>>> from fractions import Fraction
>>> from polys import Poly
>>> p = Poly(2, 3) + 4          # 2 * x ** 3 + 4
>>> p
Poly(4) + Poly(2, 3)            # bez potegi 0 przy 4
>>> p.is_zero()
False
>>> len(p)                      # liczba wyrazow
2
>>> p * Poly(3, 5)              # mnozenie
Poly(12, 5) + Poly(6, 8)
>>> p ** 2                      # potegowanie
Poly(16) + Poly(16, 3) + Poly(4, 6)
>>> p.diff()                    # pochodna
Poly(6, 2)                      # 6 * x ** 2
>>> p.integrate()               # calkowanie
Poly(4, 1) + Poly(Fraction(1, 2), 4)    # 4 * x + x ** 4 / 2
>>> p.combine(Poly(5, 7))       # skladanie wielomianow
Poly(4) + Poly(250, 21)
>>> p.eval(Fraction(3, 5))      # wartosc w punkcie
Fraction(554, 125)
>>> p.eval(0.6)                 # przejscie do float
4.432
>>> p.leading_term(key=Poly.key_deg)
Poly(2, 3)                      # wyraz wiodacy
>>> p.leading_monomial(key=Poly.key_deg)
Poly(1, 3)                      # jednomian wiodacy
>>> p.leading_coefficient(key=Poly.key_deg)
2                               # wspolczynnik wiodacy
>>> Poly(6, 5) / Poly(3, 2)     # dzielenie jednomianow
Poly(2, 3)
>>> Poly(6, 3).lcm(Poly(3, 2))
Poly(1, 3)
>>> list(p.iterterms())         # lista wyrazow (jednomianow)
[Poly(4), Poly(2, 3)]
>>> sorted(term.degree() for term in p.iterterms())
[0, 3]                          # posortowana lista stopni
>>> [p[i] for i in range(8)]    # lista osmiu wspolczynnikow
[4, 0, 0, 2, 0, 0, 0, 0]
>>> p.data          # [4, 0, 0, 2] implementacja listowa
>>> p.data          # {0: 4, 3: 2} implementacja slownikowa
# Zapis wielomianu do pliku.
>>> import pickle
>>> afile = open("data.pickle", "w")
>>> pickle.dump(p, afile)
>>> afile.close()
# Odczyt wielomianu z pliku.
>>> afile = open("data.pickle", "r")
>>> q = pickle.load(afile)
>>> afile.close()
>>> p == q
True
```

31:7123277256

# 6. Wielomiany wielu zmiennych

W tym rozdziale przedstawiono interfejs i implementację wielomianów wielu zmiennych. Implementacja klasy Poly zawarta jest w module mpolys. Jednakowa nazwa klasy pozwala wymiennie stosować różne implementacje do przypadku wielomianów jednej zmiennej.

## 6.1. Interfejs wielomianów

Interfejs wielomianów wielu zmiennych zawiera w sobie wszystkie elementy interfejsu wielomianów jednej zmiennej. Dodatkowo pojawiają się nowe elementy, ponieważ niektóre operacje mogą dotyczyć jednej z kilku zmiennych niezależnych. W przypadku wielu zmiennych w ramach niniejszej pracy przygotowano jedynie implementację słownikową klasy Poly. Oto zestawienie najważniejszych zagadnień implementacyjnych.

- W implementacji słownikowej dla wielu zmiennych, w wewnętrznym słowniku kluczem jest krotka z potęgami kolejnych zmiennych niezależnych, a wartością jest niezerowy współczynnik. Każdy klucz ma długość co najmniej jeden.
- Zmienne niezależne są numerowane od zera, a numery zmiennych są potrzebne w wielu operacjach, aby nie było niejednoznaczności (pochodna, całka, podstawianie). Domyślnie działania są wykonywane na zmiennej o indeksie zero, co zapewnia kompatybilność interfejsu z wielomianami jednej zmiennej. Wygodnie jest w obliczeniach nazywać kolejne zmienne $x$ (var=0), $y$ (var=1), $z$ (var=2), itd.
- Dla wielomianów wielu zmiennych, a w szczególności jednomianów, porównywanie można wykonać wg wielu możliwych porządków dopuszczalnych. Przygotowano metody generujące klucze do wykorzystania podczas korzystania z sortowania listy wielomianów (listing 6.1). Łatwo można dodać inny porządek do tych już istniejących.

Listing 6.1. Sortowanie wielomianów wielu zmiennych.

```
>>> poly_list = [ ... ]                      # lista wielomianow
>>> poly_list.sort(key=Poly.key_deg)         # wykorzystanie stopnia
>>> poly_list.sort(key=Poly.key_lex)         # porzadek leksykograficzny
>>> poly_list.sort(key=Poly.key_deglex)      # porzadek stopniowo leks.
```



32:5887147910

Przykładowa sesja interaktywna prezentuje możliwości wielomianów wielu zmiennych (listing 6.2).

Listing 6.2. Sesja interaktywna z wielomianami wielu zmiennych.

```
>>> from fractions import Fraction
>>> from mpolys import Poly
# Wygodne podstawienia.
>>> x = Poly(1, 1)
>>> y = Poly(1, 0, 1)
>>> z = Poly(1, 0, 0, 1)
>>> p = 3 * (x ** 2) + 5 * y * z
>>> p ** 2
Poly(25, 0, 2, 2) + Poly(30, 2, 1, 1) + Poly(9, 4)
>>> list(p.iterterms())         # lista wyrazow
[Poly(3, 2), Poly(5, 0, 1, 1)]
>>> p.degree()                  # stopien wielomianu
2
>>> len(p)                      # liczba wyrazow
2
>>> p.leading_term(key=Poly.key_deglex)
Poly(3, 2)                      # wyraz wiodacy
>>> p.leading_monomial(key=Poly.key_deglex)
Poly(1, 2)                      # jednomian wiodacy
>>> p.leading_coefficient(key=Poly.key_deglex)
3                               # wspolczynnik wiodacy
>>> (x * y).lcm(y * z)
Poly(1, 1, 1, 1)                # x * y * z
>>> Poly(3, 5, 4, 2) / Poly(2, 2, 2, 2)
Poly(Fraction(3, 2), 3, 2)      # dzielenie jednomianow
>>> p[0, 1, 1]                  # wspolczynnik przy y * z
5
>>> p[1, 0, 1]                  # wspolczynnik przy x * z
0
>>> p.diff(var=1)               # pochodna po y
Poly(5, 0, 0, 1)
>>> p.integrate(var=2)          # calka po z
Poly(Fraction(5, 2), 0, 1, 2) + Poly(3, 2, 0, 1)
>>> p.combine(y * y, var=2)     # podstawienie z = y * y
Poly(3, 2) + Poly(5, 0, 3)
>>> p.combine(Poly(7), var=1) # podstawienie y = 7
Poly(3, 2) + Poly(35, 0, 0, 1)
>>> p.data
{(2,): 3, (0, 1, 1): 5}         # implementacja slownikowa
```

## 6.2. Zamiana kolejności zmiennych

Podczas obliczania baz Gröbnera czasem zachodzi potrzeba eksperymentowania z różnymi porządkami dopuszczalnymi, czy z różną kolejnością zmiennych w porządku leksykograficznym. Załóżmy, że mamy dany wielomian w zmiennych $x, y, z$, a interesuje nas porządek leksykograficzny i kolejność $x > z > y$. W naszej implementacji rozwiązaniem jest zamiana kolejności zmiennych $y, z$, wykonana przy pomocy zmiennej $t$ (listing 6.3).



33:9178275983

Listing 6.3. Zamiana kolejności zmiennych.

```
>>> from mpolys import Poly
>>> x = Poly(1, 1)
>>> y = Poly(1, 0, 1)
>>> z = Poly(1, 0, 0, 1)
>>> t = Poly(1, 0, 0, 0, 1)
>>> p = 7 * x**2 + 8 * y**3 + 9 * z**4
>>> p
Poly(7, 2) + Poly(8, 0, 3) + Poly(9, 0, 0, 4)
# Zamiana miejscami y i z.
>>> p = p.combine(t, var=1)    # y na t
>>> p = p.combine(y, var=2)    # z na y
>>> p = p.combine(z, var=3)    # t na z
>>> p
Poly(7, 2) + Poly(8, 0, 0, 3) + Poly(9, 0, 4)
```

# 7. Algorytmy

Ten rozdział jest poświęcony algorytmom wykorzystywanym przy implementacji wielomianów lub pojawiających się podczas wykonywania działań na wielomianach.

## 7.1. Schemat Hornera

Schemat Hornera (ang. *Horner's method*, *Horner scheme*) jest to sposób obliczania wartości wielomianu jednej zmiennej, dla danej wartości argumentu, wykorzystujący minimalną liczbę mnożeń [12]. Jest to również algorytm dzielenia wielomianu przez dwumian.

Listing 7.1 przedstawia funkcję wykorzystującą schemat Hornera do obliczenia wartości wielomianu w punkcie $x$, przy czym wielomian jest dany jako lista współczynników `data`. W niniejszej pracy wykorzystujemy schemat Hornera przy obliczaniu wartości wielomianu w punkcie [metoda `eval()`], oraz przy składaniu wielomianów [metoda `combine()`].

Listing 7.1. Schemat Hornera.

```
def Horner_scheme(data, x):
    """Horner scheme for polynomials."""
        result = 0
        for item in reversed(data):
            result = result * x + item
        return result
```

Warto zauważyć, że uogólnieniem schematu Hornera jest algorytm Clenshawa [13]. Jest to rekurencyjna metoda obliczania liniowej kombinacji wielomianów Czebyszewa. Stosuje się również do dowolnej klasy funkcji definiowalnych za pomocą trójwyrazowego równania rekurencyjnego.

## 7.2. Wielomiany ortogonalne

W pracy zaimplementowano kilka rodzajów wielomianów ortogonalnych jednej zmiennej. Wykorzystano relacje rekurencyjne dla wielomianów i technikę programowania dynamicznego zstępującego. Wielomiany po obliczeniu są zapamiętywane w słowniku, co przyspiesza późniejsze obliczenia. Oto związki rekurencyjne dla zaimplementowanych wielomianów ortogonalnych:

— Wielomiany Czebyszewa, $T_0(x) = 1$, $T_1(x) = x$,

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x). \tag{7.1}$$



35:5400496148

— Wielomiany Hermite'a, $H_0(x) = 1$, $H_1(x) = 2x$,

$$H_k(x) = 2xH_{k-1}(x) - 2(k-1)H_{k-2}(x). \tag{7.2}$$

— Wielomiany Legendre'a, $P_0(x) = 1$, $P_1(x) = x$,

$$kP_k(x) = (2k-1)xP_{k-1}(x) - (k-1)P_{k-2}(x). \tag{7.3}$$

Przykładowa sesja interaktywna prezentuje korzystanie z wielomianów ortogonalnych (listing 7.2).

Listing 7.2. Sesja interaktywna z wielomianami ortogonalnymi.

```
>>> from fractions import Fraction
>>> from polys import Poly
>>> from factory import PolyFactory
>>> pf = PolyFactory(Poly)
>>> pf.legendre(3)              # wielomian Legendre'a P_3
Poly(Fraction(-3, 2), 1) + Poly(Fraction(5, 2), 3)
>>> pf.hermite(4)               # wielomian Hermite'a H_4
Poly(12) + Poly(48, 2) + Poly(16, 4)
>>> pf.chebyshev(5)             # wielomian Czebyszewa T_5
Poly(5, 1) + Poly(-20, 3) + Poly(16, 5)
>>> pf.HERMITE                  # zapamietane wielomiany Hermite'a
{0: Poly(1), 1: Poly(2, 1),
 2: Poly(2) + Poly(4, 2),
 3: Poly(12, 1) + Poly(8, 3),
 4: Poly(12) + Poly(48, 2) + Poly(16, 4)}
>>> pf.LEGENDRE                 # zapamietane wielomiany Legendre'a
>>> pf.CHEBYSHEV                # zapamietane wielomiany Czebyszewa
```

## 7.3. Algorytm Euklidesa

Algorytm Euklidesa jest wykorzystywany do znalezienia największego wspólnego podzielnika dwóch wielomianów jednej zmiennej [6].

Listing 7.3. Moduł edivision z algorytmem Euklidesa.

```
#!/usr/bin/python
#
# Based on pseudocode from:
#
# http://en.wikipedia.org/wiki/Polynomial_long_division

class EuclideanDivision:
    """The Euclidean division of polynomials."""

    def __init__(self, first_poly, second_poly):
        """The algorithm initialization."""
        if second_poly.is_zero():
            raise ValueError("poly is zero")
        self.cls = first_poly.__class__
        self.A = first_poly
        self.B = second_poly
        self.Q = self.cls()    # the quotient
```



36:4796728332

```
        self.R = self.A    # the remainder

    def run(self):
        """Calculate Q and R, A = B * Q + R, R.degree() < B.degree()."""
        # At each step A = B * Q + R.
        while (not self.R.is_zero()) and (
            self.R.degree() >= self.B.degree()):
            # Divide the leading term.
            T = (self.R.leading_term(self.cls.key_deg) /
                 self.B.leading_term(self.cls.key_deg))
            self.Q = self.Q + T
            self.R = self.R - T * self.B


class PolyGCD:
    """The greatest common divisor (GCD) of polynomials."""

    def __init__(self, first_poly, second_poly):
        """The algorithm initialization."""
        self.A = first_poly
        self.B = second_poly
        self.gcd = None

    def run(self):
        """Calculate GCD using Euclidean algorithm."""
        a = self.A
        b = self.B
        while not b.is_zero():
            algorithm = EuclideanDivision(a, b)
            algorithm.run()
            a = b
            b = algorithm.R
        self.gcd = a / a[a.degree()]
```

## 7.4. Algorytm dzielenia uogólnionego

Algorytm dzielenia uogólnionego jest uogólnieniem algorytmu Euklidesa na przypadek wielomianów wielu zmiennych [9]. Jest to w istocie redukcja wielomianu $f$ ze względu na zbiór wielomianów $F = \{f_1, \ldots, f_s\}$ z $K[x]$, przy ustalonym porządku dopuszczalnym. Na wyjściu otrzymujemy zbiór wielomianów $h_i$ i wielomian $r$, przy czym $f = \sum_{i=1}^{s} f_i h_i + r$. Wielomian $r$ nie dzieli się przez żadne $LT(f_i)$. W ogólnym przypadku wielomiany $h_i$ i $r$ nie są wyznaczone jednoznacznie, chyba że $F$ jest bazą Gröbnera. Algorytm dzielenia uogólnionego jest wykorzystywany przy wyznaczaniu bazy Gröbnera.

Listing 7.4. Moduł gdivision z dzieleniem uogólnionym.

```
#!/usr/bin/python

class GeneralizedDivision:
    """Generalized division of polynomials."""

    def __init__(self, first_poly, base, key=None):
        """The algorithm initialization."""
```



37:6657590247

```
        if len(base) == 0:
            raise ValueError("base is empty")
        self.cls = first_poly.__class__
        self.A = first_poly
        self.B = base
        self.Q = [self.cls() for item in base]   # the quotients
        self.R = self.cls()   # the remainder
        self.sort_key = key

    def run(self):
        """Calculate Q and R, A = \sum_i B[i] * Q[i] + R."""
        poly = self.A
        base_len = len(self.B)
        while not poly.is_zero():
            dividing = True
            i = 0
            while i < base_len and dividing:
                try:
                    item = self.B[i]
                    mono = (poly.leading_term(self.sort_key) /
                            item.leading_term(self.sort_key))
                    self.Q[i] += mono
                    poly -= mono * item
                    dividing = False
                except ValueError:
                    i += 1
            if dividing:   # ERROR in [Ajwa, Liu, Wang, 2003]
                p_lt = poly.leading_term(self.sort_key)
                self.R += p_lt
                poly -= p_lt
```

## 7.5. Algorytm Buchbergera

Algorytm Buchbergera jest oryginalnym algorytmem podanym przez Buchbergera w jego pracy doktorskiej. Algorytm służy do wyznaczenia bazy Gröbnera $G$ na bazie podanego zbioru wielomianów $F$ [9], [14], [15]. Idea algorytmu jest stosunkowo prosta. Inicjalizujemy bazę Gröbnera $G$ jako równą zbiorowi $F$. Tworzymy zbiór $M$ par różnych wielomianów z $G$. Dla każdej pary $(p, q)$ z $M$ obliczamy S-wielomian, który następnie redukujemy modulo $G$ do wielomianu $h$. Jeżeli wielomian $h$ jest niezerowy, to dodajemy go bazy $G$, oraz uaktualniamy zbiór par $M$. Czynności kontynuujemy do momentu wyczerpania zbioru par $M$.

W naszej implementacji wielomiany ze zbioru $F$ dodajemy pojedyńczo do bazy $G$ (metoda insert). Po każdym dodaniu wielomianu następuje sekwencja tworzenia par, obliczania S-wielomianów (metoda _spoly) i redukcji (metoda _reduce).

**Złożoność:** Złożoność obliczeniowa algorytmu Buchbergera jest trudna do oszacowania. Dla zredukowanej bazy Gröbnera pokazano [16], że stopnie ele-



38:6777161940

mentów w bazie są ograniczone przez wyrażenie

$$2\left(\frac{d^2}{2}+d\right)^{2^{n-1}}, \tag{7.4}$$

gdzie $n$ jest liczbą zmiennych, a $d$ największym stopniem wielomianów wejściowych. Jest to zależność podwójnie wykładnicza, która obrazuje prawdziwą trudność problemu. Okazuje się jednak, że w wielu praktycznych zastosowaniach algorytm działa znacznie wydajniej. Może to oznaczać, że oczekiwany czas działania algorytmu zależy nie od $n$ i $d$, a od wewnętrznych własności algebraicznych danych wejściowych.

**Uwagi:** W podstawowej postaci algorytm Buchbergera nie precyzuje sposobu pobierania par ze zbioru $M$. Na listingu 7.5 $M$ to kolejka FIFO. Sprawdzaliśmy także kolejkę LIFO, czyli stos.

Listing 7.5. Moduł groebner3 z algorytmem Buchbergera.

```
#!/usr/bin/python

from Queue import Queue

class Groebner:
    """Buchberger's algoritm for finding a Groebner basis."""

    def __init__(self, key=None):
        """Load up an algorithm instance."""
        self.base = list()
        self.sort_key = key

    def insert(self, poly):
        """Insert a poly to the base."""
        M = Queue()
        for item in self.base:
            M.put((item, poly))
        self.base.append(poly)
        while not M.empty():
            a, b = M.get()
            poly = self._spoly(a, b)
            poly = self._reduce(poly, self.base)   # normal form
            if not poly.is_zero():
                for item in self.base:
                    M.put((item, poly))
                self.base.append(poly)

    def _reduce(self, poly, base):
        """Return the normal form of the poly."""
        remainder = poly.__class__()
        base_len = len(base)
        while not poly.is_zero():
            dividing = True
            i = 0
            while i < base_len and dividing:
                try:
                    item = base[i]
                    mono = (poly.leading_term(self.sort_key) /
```



39:8941893298

```
                    item.leading_term(self.sort_key))
                poly -= mono * item
                dividing = False
            except ValueError:
                i += 1
        if dividing:    # ERROR in [Ajwa, Liu, Wang, 2003]
            p_lt = poly.leading_term(self.sort_key)
            remainder += p_lt
            poly -= p_lt
    return remainder

def _spoly(self, a, b):
    """Return S-poly."""
    a_lt = a.leading_term(self.sort_key)
    b_lt = b.leading_term(self.sort_key)
    LCM = a_lt.lcm(b_lt)
    return (LCM / a_lt) * a - (LCM / b_lt) * b
```

## 7.6. Wyznaczanie zredukowanej bazy Gröbnera

W praktycznych obliczeniach najwygodniejsza jest zredukowana baza Gröbnera, ponieważ jest wyznaczona jednoznacznie dla danego porządku dopuszczalnego i jest najmniej liczna.

Listing 7.6. Moduł groebner4 (wyznaczanie zredukowanej bazy Gröbnera).

```
#!/usr/bin/python

from Queue import Queue

class Groebner:
    """Buchberger's algoritm for finding a reduced Groebner basis."""

    def __init__(self, key=None):
        """Load up an algorithm instance."""
        self.base = list()
        self.sort_key = key

    def insert(self, poly):
        """Insert a poly to the base."""
        M = Queue()
        if len(self.base) > 0:
            poly = self._reduce(poly, self.base)
        if poly.is_zero():
            return
        poly /= poly.leading_coefficient(self.sort_key)
        for item in self.base:
            M.put((item, poly))
        self.base.append(poly)
        while not M.empty():
            a, b = M.get()
            poly = self._spoly(a, b)
            poly = self._reduce(poly, self.base)    # normal form
            if not poly.is_zero():
                poly /= poly.leading_coefficient(self.sort_key)
```



40:7724705950

```
                for item in self.base:
                    M.put((item, poly))
                self.base.append(poly)
        # Redukcja bazy Groebnera.
        new_base = []
        base_len = len(self.base)
        for i in xrange(base_len-1):
            poly = self.base[i]
            poly = self._reduce(poly, new_base + self.base[i+1:])
            if not poly.is_zero():
                poly /= poly.leading_coefficient(self.sort_key)
                new_base.append(poly)
        poly = self.base[-1]    # ostatni element
        poly = self._reduce(poly, new_base)
        if not poly.is_zero():
            poly /= poly.leading_coefficient(self.sort_key)
            new_base.append(poly)
        self.base = new_base

    def _reduce(self, poly, base):
        """Return the normal form of the poly."""
        remainder = poly.__class__()
        base_len = len(base)
        while not poly.is_zero():
            dividing = True
            i = 0
            while i < base_len and dividing:
                try:
                    item = base[i]
                    mono = (poly.leading_term(self.sort_key) /
                        item.leading_term(self.sort_key))
                    poly -= mono * item
                    dividing = False
                except ValueError:
                    i += 1
            if dividing:    # ERROR in [Ajwa, Liu, Wang, 2003]
                p_lt = poly.leading_term(self.sort_key)
                remainder += p_lt
                poly -= p_lt
        return remainder

    def _spoly(self, a, b):
        """Return S-poly."""
        a_lt = a.leading_term(self.sort_key)
        b_lt = b.leading_term(self.sort_key)
        LCM = a_lt.lcm(b_lt)
        return (LCM / a_lt) * a - (LCM / b_lt) * b
```

## 7.7. Ulepszony algorytm Buchbergera

W literaturze znane są różne usprawnienia oryginalnego algorytmu Buchbergera. Są to kryteria usuwania (ang. *deletion criteria*) i strategie wyboru (ang. *selection stategies*).



41:9686141427

— W podstawowowym algorytmie Buchbergera mamy swobodę wyboru pary wielomianów $(p, q)$ ze zbioru $M$. Okazuje się, że pewne strategie wyboru mogą prowadzić do szybszego zakończenia pracy algorytmu, choć nie w każdym przypadku. Stosowano przykładowo następujące heurystyki: (1) wybieranie pary o najmniejszej wartości LCM, (2) wybieranie pary o najmniejszym rozmiarze całkowitym wielomianów, (3) wybieranie najstarszej pary.
— Pewne pary wielomianów $(p, q)$ mogą być usunięte ze zbioru $M$, ponieważ i tak będą zredukowane do zera. Tak jest przykładowo wtedy, gdy $LM(p)$ i $LM(q)$ nie mają wspólnych zmiennych. Wtedy zachodzi związek $\mathrm{lcm}(LM(f), LM(g)) = LM(f)LM(g)$.

Listing 7.7. Moduł groebner6 (ulepszony algorytm Buchbergera dla bazy zredukowanej).

```
#!/usr/bin/python

from Queue import PriorityQueue

class Groebner:
    """Buchberger's algoritm for finding a reduced Groebner basis."""

    def __init__(self, key=None):
        """Load up an algorithm instance."""
        self.base = list()
        self.sort_key = key
        self.time = 0       # time stamp

    def priority_number1(self, a, b):
        """Return a priority number."""
        a_lt = a.leading_term(self.sort_key)
        b_lt = b.leading_term(self.sort_key)
        LCM = a_lt.lcm(b_lt)
        return self.sort_key(LCM)

    def priority_number2(self, a, b):
        """Return a priority number."""
        a_lt = a.leading_term(self.sort_key)
        b_lt = b.leading_term(self.sort_key)
        LCM = a_lt.lcm(b_lt)
        return LCM.degree()

    def priority_number3(self, a, b):
        """Return a priority number."""
        return len(a) + len(b)    # the number of terms

    def priority_number4(self, a, b):
        """Return a priority number."""
        return max(a.degree(), b.degree())    # max degree

    priority_number = priority_number1

    def insert(self, poly):
        """Insert a poly to the base."""
        # Podejscie wg [Ajwa, Liu, Wang, 2003].
```



42:7799997564

```
        M = PriorityQueue()    # kolejka par wielomianow do przetworzenia
        if len(self.base) > 0:
            poly = self._reduce(poly, self.base)
        if poly.is_zero():    # wielomian nalezy do idealu
            return
        # Tworzymy wielomian unitarny.
        poly /= poly.leading_coefficient(self.sort_key)
        poly_lm = poly.leading_monomial(self.sort_key)
        for item in self.base:
            item_lm = item.leading_monomial(self.sort_key)
            LCM = poly_lm.lcm(item_lm)
            if LCM == poly_lm * item_lm:
                # Jednomiany wzglednie pierwsze, wiec nie wstawiamy.
                continue
            self.time += 1
            pri = self.priority_number(item, poly)
            M.put((pri, self.time, item, poly))
        self.base.append(poly)
        while not M.empty():
            pri, time, a, b = M.get()
            poly = self._spoly(a, b)
            poly = self._reduce(poly, self.base)    # normal form
            if not poly.is_zero():
                # Tworzymy wielomian unitarny.
                poly /= poly.leading_coefficient(self.sort_key)
                for item in self.base:
                    self.time += 1
                    pri = self.priority_number(item, poly)
                    M.put((pri, self.time, item, poly))
                self.base.append(poly)
        # Teraz chcemy zredukowac posiadana baze Groebnera.
        new_base = []
        base_len = len(self.base)
        for i in xrange(base_len-1):    # bez ostatniego
            poly = self.base[i]
            poly = self._reduce(poly, new_base + self.base[i+1:])
            if not poly.is_zero():
                poly /= poly.leading_coefficient(self.sort_key)
                new_base.append(poly)
        # Ostatni przetwarzam osobno (nie wiem, czy to potrzebne).
        poly = self.base[-1]
        poly = self._reduce(poly, new_base)
        if not poly.is_zero():
            poly /= poly.leading_coefficient(self.sort_key)
            new_base.append(poly)
        self.base = new_base

    def _reduce(self, poly, base):
        """Return the normal form of the poly."""
        # Redukowanie wzgledem calej bazy.
        remainder = poly.__class__()    # nie ma jawnie nazwy Poly
        base_len = len(base)
        while not poly.is_zero():
            dividing = True
            i = 0
            while i < base_len and dividing:
                try:
```



43:9706536810

```
                item = base[i]
                mono = (poly.leading_term(self.sort_key) /
                        item.leading_term(self.sort_key))
                poly -= mono * item
                dividing = False
            except ValueError:
                i += 1
        if dividing:    # ERROR in [Ajwa, Liu, Wang, 2003]
            p_lt = poly.leading_term(self.sort_key)
            remainder += p_lt
            poly -= p_lt
    return remainder

def _spoly(self, a, b):
    """Return S-poly."""
    a_lt = a.leading_term(self.sort_key)
    b_lt = b.leading_term(self.sort_key)
    LCM = a_lt.lcm(b_lt)
    return (LCM / a_lt) * a - (LCM / b_lt) * b
```

## 7.8. Algorytmy Faugére

Od momentu odkrycia zaproponowano wiele wariantów algorytmu Buchbergera. Za najbardziej efektywne uważa się obecnie algorytmy F4 i F5, których autorem jest Faugére [17]. Oba algorytmy wykonują działania na macierzach rzadkich. Wiele redukcji jest wykonywanych równolegle. Algorytm F5 był w stanie rozwiązać kilka problemów kryptograficznych.

Algorytm F4 [18] zaimplementowano w programach FGb, Maple i Magma. Algorytm F5 [19] zaimplementowano w programach Singular i Sage.

# 8. Systemy algebry komputerowej

Przedstawimy krótko dwa systemy algebry komputerowej, które są dostępne bez żadnych opłat. Wykorzystamy je do niezależnych testów naszych wyników, oraz do porównania składni.

## 8.1. Singular

Program SINGULAR służy do obliczeń wielomianowych [20]. Jest rozwijany na Wydziale Matematyki Uniwersytetu Technicznego w Kaiserslautern (Niemcy). Pierwsze prace nad programem zaczęły się w roku 1984, a wersja 1.0 pojawiła się w roku 1997. Obecną wersję 4 można instalować na platformach Unix/Linux, Windows, Macintosh.

## 8.2. CoCoA

Program CoCoA (ang. *Computations in Commutative Algebra*) służy do obliczeń z pierścieniami wielomianów wielu zmiennych [21]. Są tutaj dwa składniki: system CoCoA-5 (napisany w języku interpretowanym CoCoALanguage, podobnym do Pascala) i biblioteka CoCoALib (napisana w C++). Projekt jest rozwijany od roku 1987. Autorzy pracują głównie na Wydziale Matematyki (DIMA) Uniwersytetu w Genui (Włochy). Obecnie CoCoA działa na wielu platformach, m.in. Linux, MacOS X, Microsoft Windows.

## 8.3. Przykładowe obliczenia

Rozważymy dwa proste problemy prowadzące do układów wielomianowych. Przedstawimy rozwiązywanie tych problemów przy użyciu naszego kodu (listing 8.1), oraz przy użyciu programów SINGULAR (listing 8.2) i CoCoA (listing 8.3).

1. Problem kolorowania wierzchołków grafu trzema kolorami [8]. Zmienne $x_j$ oznaczają kolory wierzchołków, które mogą być jednym z trzech zespolonych pierwiastków trzeciego stopnia z 1. Dla każdego wierzchołka dodajemy równanie $x_j^3 - 1 = 0$, a dla każdej krawędzi $(x_j, x_k)$ równanie $x_j^2 + x_j x_k + x_k^2 = 0$. Z końcowych równań dla naszego grafu wynika równość kolorów wierzchołków $x_1 = x_5$, $x_2 = x_6$, $x_3 = x_4$.
2. Udowodnić twierdzenie, że przekątne prostokąta dzielą się na połowy [8]. Dobieramy układ współrzędnych tak, aby wierzchołki prostokąta znala-



45:2406366841

zły się w punktach $(0,0)$, $(a,0)$, $(0,b)$, $(a,b)$. Punkt $(x,y)$ jest punktem przecięcia przekątnych. Tworzymy układ równań

$$ya - xb = 0, \tag{8.1}$$
$$xb + ya - ab = 0, \tag{8.2}$$
$$abt - 1 = 0. \tag{8.3}$$

Pierwsze dwa równania opisują przekątne. Trzecie równanie ze zmienną dodatkową $t$ wymusza niezerowe $a$ i $b$ (unikamy patologii). Końcowe równania dają tezę twierdzenia $2x = a$, $2y = b$.

Listing 8.1. Obliczenia z użyciem naszego kodu.

```
#1
# 0 ---- 1 ---- 2
#   \    / \    / \
#    \  /   \  /   \
#     \/     \/     \
#      3 ---- 4 ---- 5
from mpolys import Poly
from groebner5 import Groebner

N = 6    # liczba wierzcholkow grafu
nodes = range(N)
edges = [(0, 1), (0, 3), (1, 2), (1, 3), (1, 4), (2, 4),
    (3, 4), (2, 5), (4, 5)]
x = N * [None]
for i in range(N):
    alist = [1] + i * [0] + [1]
    x[i] = Poly(*alist)
equations = []
for i in nodes:
    equations.append(x[i] * x[i] * x[i] + Poly(-1))
for (i, j) in edges:
    equations.append(x[i] * x[i] + x[i] * x[j]  + x[j] * x[j])

#ideal = Groebner(key=Poly.key_deglex)
ideal = Groebner(key=Poly.key_lex)
for item in equations:
    ideal.insert(item)
for item in ideal.base:
    print item
# Wynik po uporzadkowaniu.
# x[5]**3 -1 = 0
# -x[4] + x[0] = 0
# x[1] - x[5] = 0
# x[2] + x[4] + x[5] = 0
# x[3] + x[5] + x[4] = 0
# x[4]**2 + x[4] * x[5] + x[5]**2 = 0
#------------------------------------------------------------------
#2
from mpolys import Poly
from groebner5 import Groebner

a = Poly(1, 1)
b = Poly(1, 0, 1)
```



46:6659739794

```
x = Poly(1, 0, 0, 1)
y = Poly(1, 0, 0, 0, 1)
t = Poly(1, 0, 0, 0, 0, 1)
equations = []
equations.append(y * a - x * b)
equations.append(x * b + y * a - a * b)
equations.append(a * b * t + Poly(-1))

ideal = Groebner(key=Poly.key_deglex)
#ideal = Groebner(key=Poly.key_lex)
for item in equations:
    ideal.insert(item)
for item in ideal.base:
    print item
# Wynik po uporzadkowaniu.
# a - 2 * x = 0
# b - 2 * y = 0
# x * y * t + Poly(Fraction(-1, 4)) = 0
```

Listing 8.2. Obliczenia w programie SINGULAR.

```
#1
ring RING=0,x(0..5),lp; option(redSB);
ideal IDEAL; int i;
for (i=0;i<=5;i++) { IDEAL=IDEAL,x(i)^3-1;}
list edges=0, 1, 0, 3, 1, 2, 1, 3, 1, 4, 2, 4, 3, 4, 2, 5, 4, 5;
for (i=1;i<size(edges);i=i+2) { IDEAL=IDEAL,x(edges[i])^2
+x(edges[i])*x(edges[i+1])+x(edges[i+1])^2; }
std(IDEAL);

# Wynik wyswietlony przez program
_[1]=x(5)^3-1
_[2]=x(4)^2+x(4)*x(5)+x(5)^2
_[3]=x(3)+x(4)+x(5)
_[4]=x(2)+x(4)+x(5)
_[5]=x(1)-x(5)
_[6]=x(0)-x(4)
#2
ring RING=0,(a,b,x,y,t),lp; option(redSB);
ideal IDEAL=y*a-x*b,x*b+y*a-a*b,a*b*t-1;
std(IDEAL);

# Wynik wyswietlony przez program
_[1]=4xyt-1
_[2]=b-2y
_[3]=a-2x
```

Listing 8.3. Obliczenia w programie CoCoA.

```
#1
Use ring ::= QQ[x[0..5]], Lex;
I1 := Ideal([X^3-1 | X In Indets(ring)]);
edges := [[0, 1], [0, 3], [1, 2], [1, 3], [1, 4], [2, 4], [3, 4],
[2, 5], [4, 5]];
I2 := Ideal([x[i[1]]^2+x[i[1]]*x[i[2]]+x[i[2]]^2 | i In edges]);
ReducedGBasis(I1+I2);
```



47:4334583894

```
# Wynik wyswietlony przez program (po sformatowaniu
# do czytelniejszej wersji)
[
x[5]^3 -1,
x[4]^2 +x[4]*x[5] +x[5]^2,
x[3] +x[4] +x[5],
x[2] +x[4] +x[5],
x[1] -x[5],
x[0] -x[4]
]
#2
Use ring ::= QQ[a,b,x,y,t], Lex;
I := Ideal(y*a-x*b,x*b+y*a-a*b,a*b*t-1);
ReducedGBasis(I);

# Wynik wyswietlony przez program (po sformatowaniu
# do czytelniejszej wersji)
[
x*y*t -1/4,
b -2*y,
a -2*x
]
```

# 9. Podsumowanie

W pracy zdefiniowano interfejs wielomianów, który pozwala na typowe obliczenia związane z wielomianami. Przygotowano dwie implementacje wielomianów jednej zmiennej (moduł polys), oraz implementację słownikową wielomianów wielu zmiennych (moduł mpolys). Jest możliwość prowadzenia obliczeń z wykorzystaniem porządku leksykograficznego lub porządku stopniowo leksykograficznego.

Przygotowano generatory pewnych typowych wielomianów (moduł factory), a także generatory trzech rodzajów wielomianów ortogonalnych: Czebyszewa, Hermite'a i Legendre'a. Generatory korzystają ze zdefiniowanego interfejsu wielomianów, dlatego pracują z dowolną implementacją wielomianów.

Najważniejszym wynikiem pracy jest implementacja algorytmów baz Gröbnera, które są podstawą systemów algebry komputerowej. Na początku zaimplementowano algorytm Euklidesa dla wielomianów jednej zmiennej (moduł edivision) i algorytm uogólnionego dzielenia wielomianów wielu zmiennych (moduł gdivision). Wreszcie powstała implementacja oryginalnego algorytmu Buchbergera do wyznaczania bazy Gröbnera i zredukowanej bazy Gröbnera (moduły groebner*). Pokazano przykładowe obliczenia z użyciem stworzonej implementacji, a także pokazano analogiczne obliczenia prowadzone przy użyciu programów Singular i CoCoA.



49:2187433101

# A. Testy wybranych algorytmów

W tym dodatku przedstawimy testy implementacji wielomianów i wybranych algorytmów wielomianowych.

## A.1. Działania na wielomianach jednej zmiennej

W ramach pracy sprawdzono wydajność działań na wielomianach jednej zmiennej. Podstawowe działania to dodawanie (rysunek A.1), odejmowanie (rysunek A.2) i mnożenie (rysunek A.3). Do działań użyto dwóch różnych wielomianów stopnia $n$ o współczynnikach całkowitych. Sprawdzono implementację listową i słownikową. Dla obu implementacji dodawanie i odejmowanie przekracza spodziewany czas $O(n)$, a mnożenie przekracza czas $O(n^2)$. Podobne zachowanie [przekroczenie $O(n)$] zaobserwowaliśmy dla operacji różniczkowania i całkowania wielomianów jednej zmiennej, dla implementacji listowej i słownikowej.

Prawdopodobną przyczyną braku liniowości w dodawaniu i odejmowaniu jest brak liniowości ukryty w wewnętrznych mechanizmach Pythona dotyczących obsługi list i słowników. Przykładowo przy zwiększaniu liczby kluczy w słowniku wielkość tablicy mieszającej zwiększa się, a indeksy są ponownie przeliczane. Przy listach natomiast warto pamiętać, że Python rezerwuje zwykle więcej pamięci, aby szybciej obsłużyć powiększanie listy. Jednak po wyczerpaniu rezerwy potrzebna jest nowa alokacja pamięci.

Przy implementacji listowej wielomianów jednej zmiennej można w dodawaniu i odejmowaniu zastąpić pętle **for** przez narzędzie `itertools.izip_longest`. Przynosi to prawie dwukrotny wzrost szybkości.

## A.2. Działania na wielomianach wielu zmiennych

W pracy sprawdzono także działania na wielomianach wielu zmiennych. Zbadano czas dodawania, odejmowania i mnożenia dwóch wielomianów stopnia $n$, jednego w zmiennej $y$, a drugiego w zmiennej $z$. Dodawanie i odejmowanie było rzędu $O(n)$, natomiast mnożenie lekko przekraczało zależność $O(n^2)$.

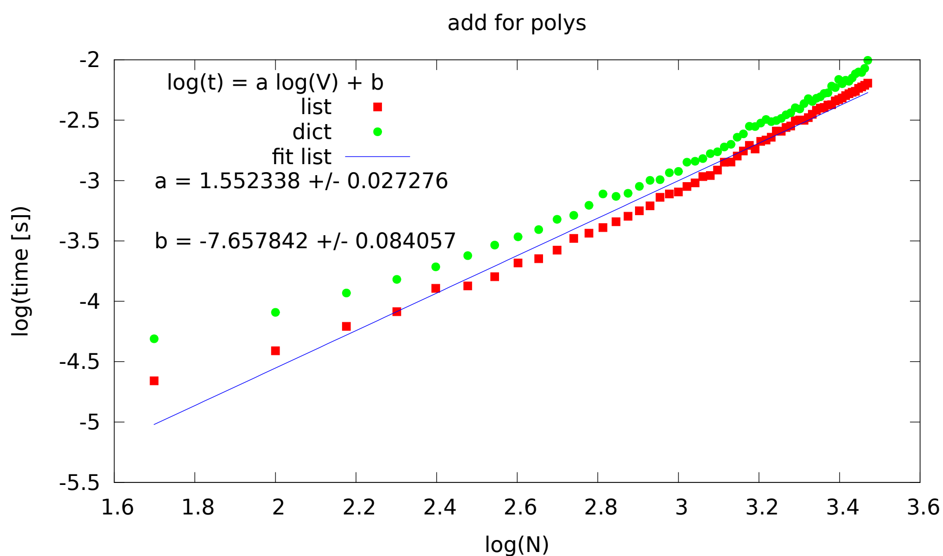
## A.3. Testy algorytmu Buchbergera

Wykonaliśmy kilka testów algorytmu Buchbergera dla następującego zestawu wielomianów [8]:

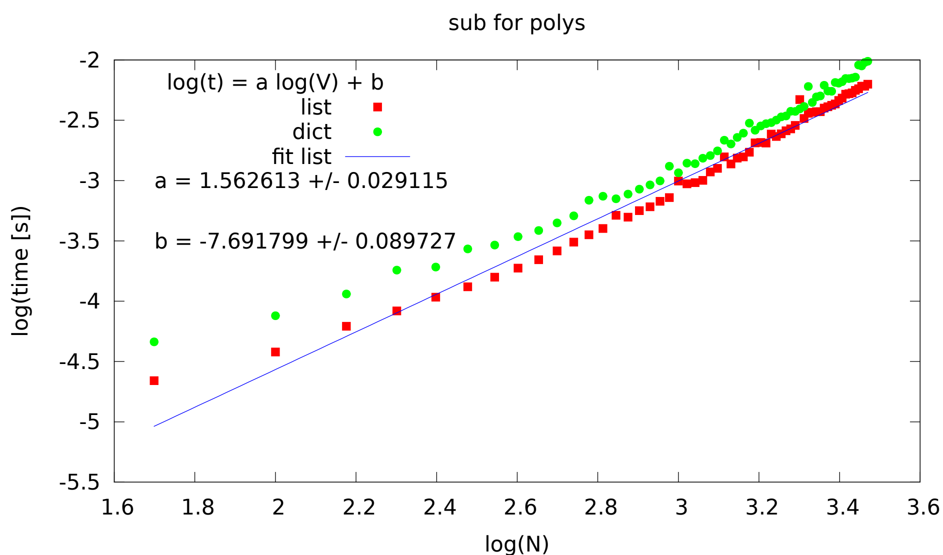
$$x^k - y^k, y^k - z^k, z^k - t^k, t^k - s^k, x^{k-1}y + y^{k-1}z + z^{k-1}t + t^{k-1}s + s^{k-1}x. \quad (A.1)$$



50:4101257929



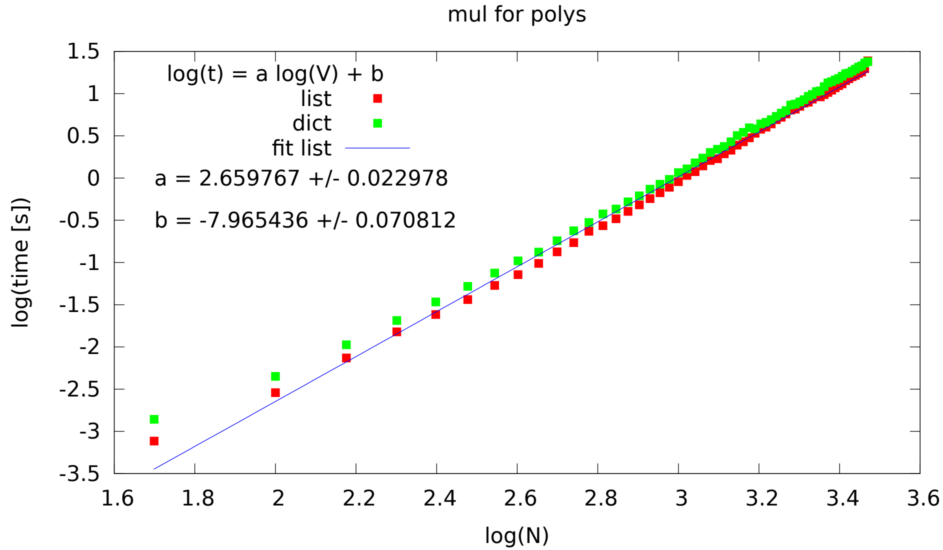

Rysunek A.1. Dodawanie wielomianów jednej zmiennej.




Rysunek A.2. Odejmowanie wielomianów jednej zmiennej.



51:6828333092




Rysunek A.3. Mnożenie wielomianów jednej zmiennej.

Tabela A.1. Wyniki testów algorytmu Buchbergera: czasy wyznaczania zredukowanej bazy Gröbnera. Sprawdzane priorytety par wielomianów: PRI1 to klucz sortowania dla LCM, PRI2 to stopień LCM, PRI3 to suma wyrazów pary, PRI4 to największy stopień z pary.

| **Moduł, porządek** | **PRI1** | **PRI2** | **PRI3** | **PRI4** |
|---|---|---|---|---|
| groebner4 *lex* | 0.29212 | | | |
| groebner5 *lex* | 0.95550 | 0.23351 | 0.89321 | 0.31401 |
| groebner6 *lex* | 1.01833 | 0.24172 | 0.95037 | 0.31154 |
| groebner4 *deglex* | 0.30376 | | | |
| groebner5 *deglex* | 0.24103 | 0.25723 | 0.96792 | 0.32840 |
| groebner6 *deglex* | 0.22806 | 0.24819 | 0.98712 | 0.31972 |

Dla $k = 2$ i porządku *deglex* lub *lex* zredukowana baza Gröbnera liczy 17 elementów. Testy z modułami groebner3 i groebner4 pokazały, że około 6 razy szybciej wykonują się obliczenia, kiedy zbiór $M$ działa jak kolejka FIFO, a nie LIFO (stos).

Algorytmy z modułów groebner5 i groebner6 wykorzystują do realizacji zbioru $M$ kolejkę priorytetową, przy czym priorytety można ustalać na różne sposoby. Sprawdziliśmy cztery różne sposoby przyznawania priorytetów, przy czym kolejka pobiera elementy o najmniejszym priorytecie. Wyniki zebrano w tabeli A.1. Najkrótsze czasy obliczeń przy porządku leksykograficznym otrzymaliśmy przy ustaleniu priorytetu pobierania elementów zbioru $M$ na stopień LCM (PRI2). Przy porządku stopniowo leksykograficznym najlepszy okazał się priorytet będący kluczem sortowania dla wielomianu LCM (PRI1).

# Bibliografia


[1] Python Programming Language - Official Website,
https://www.python.org/.

[2] Wikipedia, Polynomial, 2016,
https://en.wikipedia.org/wiki/Polynomial.

[3] Wikipedia, Characteristic polynomial, 2016,
https://en.wikipedia.org/wiki/Characteristic_polynomial.

[4] Wikipedia, Chromatic polynomial, 2016,
https://en.wikipedia.org/wiki/Chromatic_polynomial.

[5] Wikipedia, Gröbner basis, 2016,
https://en.wikipedia.org/wiki/Grobner_basis.

[6] Wikipedia, Euclidean algorithm, 2016,
https://en.wikipedia.org/wiki/Euclidean_algorithm.

[7] Heisuke Hironaka, *Resolution of singularities of an algebraic variety over a field of characteristic zero, I, II*, Annals of Mathematics, Second Series 79, 205-326 (1964).

[8] Marcin Dumnicki, Tadeusz Winiarski, *Bazy Gröbnera: efektywne metody w układach wielomianowych*, Wydawnictwo Naukowe Uniwersytetu Pedagogicznego, Kraków 2009.

[9] Iyad A. Ajwa, Zhuojun Liu, Paul S. Wang, *Grobner Bases Algorithm*, ICM Technical Reports Series, Ken State University, 1995,
http://icm.mcs.kent.edu/reports/1995/gb.pdf.

[10] Scholarpedia, Groebner basis, 2016,
http://www.scholarpedia.org/article/Groebner_bases.

[11] Robert Sedgewick, *Algorytmy w C++*, Wydawnictwo RM, Warszawa 1999.

[12] Wikipedia, Horner's method, 2016,
https://en.wikipedia.org/wiki/Horner%27s_method.

[13] Wikipedia, Clenshaw algorithm, 2016,
https://en.wikipedia.org/wiki/Clenshaw_algorithm.

[14] Wikipedia, Buchberger's algorithm, 2016,
https://en.wikipedia.org/wiki/Buchberger's_algorithm.

[15] Bruno Buchberger, *A theoretical basis for the reduction of polynomials to canonical forms*, ACM SIGSAM Bulletin 39, 19-29, (1976).

[16] Thomas W. Dubé, *The Structure of Polynomial Ideals and Gröbner Bases*, SIAM Journal on Computing 19, 750-773 (1990).

[17] Wikipedia, Faugére's F4 and F5 algorithms, 2016,
https://en.wikipedia.org/wiki/Faugére's_F4_and_F5_algorithms.

[18] Jean-Charles Faugére, *A new efficient algorithm for computing Gröbner bases (F4)*, Journal of Pure and Applied Algebra (Elsevier Science) 139, 61-88 (1999).

[19] Jean-Charles Faugére, *A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)*, Proceedings of the 2002 international symposium on Symbolic and algebraic computation (ISSAC) (ACM Press), 75-83 (2002).




53:6491027335

[20] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, Hans Schönemann, SINGULAR 4-0-2 — A computer algebra system for polynomial computations, 2015,
http://www.singular.uni-kl.de/.

[21] John Abbott, Anna Maria Bigatti, Giovanni Lagorio, CoCoA-5: a system for doing Computations in Commutative Algebra, 2016,
http://cocoa.dima.unige.it/.



54:5208272526